Searching, Sorting, Complexity Part 02
• Efficiency
  – Producing desired results with little to no waste
  – Well organized and prevents wasteful use of a resource

• Resources
  – Time
  – Space

• How do we measure efficiency?
  – Algorithms do not require computers
Complexity

- Classifies Computational Problems based on inherent difficulty
- Relates problems to each other
- Time and Space

Asymptotic Analysis
- A way to describe a limiting behavior / function
- Limits in math are a value that a function approaches as the input approaches some value
- Time and Space Complexity
• Theoretical upper bound of an algorithm
• The “Worst Case” scenario
• Let f and g be functions defined on some subset of real numbers

\[
 f(n) = O(g(n)) \text { where } n \in \mathbb{R} \text { as } n \to \infty
\]

• Let M be a constant that’s sufficiently large then we can say

\[
 |f(n)| \leq M |g(n)| \text { for all } n \geq n_0
\]
• Common Big O Complexities
  – O(1) – Constant
  – O(log(n)) – Logarithmic
  – O(n) – Linear
  – O(n\log n) – Linearithmic
  – O(n^2) – Quadratic
  – O(2^n) – Exponential “Bad”
  – O(n!) – Factorial “Really Bad”
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

---

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem:
– Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length
Problem:
- Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem:
– Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem:
- Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

**Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem:
- Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the staring index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Problem:
– Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

Selection Sort
1. Start from index 0
2. Assume the starting index has the smallest value and record that index
3. Sequentially check every other value
4. If a value is found that is smaller at another index, then record that current index
5. Once all values have been checked if the recorded index does not match the current index then swap those values
6. Increase the starting index by 1
7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
```
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the staring index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the staring index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
A Few Swaps Later
• Problem:
  – Given any array of integers, develop an algorithm that sorts the values from smallest to largest.

• Selection Sort
  1. Start from index 0
  2. Assume the starting index has the smallest value and record that index
  3. Sequentially check every other value
  4. If a value is found that is smaller at another index, then record that current index
  5. Once all values have been checked if the recorded index does not match the current index then swap those values
  6. Increase the starting index by 1
  7. Repeat 2 through 6 until the starting index >= length

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Descending Order

• Operations
  – Search for smallest value = n
  – Search for the next smallest value = n – 1
  – Search for the next smallest value = n – 2
  – ...
  – Search for the largest element = 1
Selection Sort Complexity

- **Worst Case**
  - Sorted in Descending Order

- **Operations**
  - Search for smallest value = n
  - Search for the next smallest value = n - 1
  - Search for the next smallest value = n - 2
  - ...  
  - Search for the largest element = 1

**Complexity**

$O(n^2)$
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Bubble Sort
1. Start from index 0
2. Check each index with its neighbor (index+1)
3. If that neighbor’s value is smaller then swap with the current index’s value
4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Example
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• **Bubble Sort**
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

### Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

![Swap arrows](image-url)
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>12</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>12</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>11</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td>12</td>
</tr>
</tbody>
</table>
A Few Swaps Later
• Bubble Sort
  1. Start from index 0
  2. Check each index with its neighbor (index+1)
  3. If that neighbor’s value is smaller then swap with the current index’s value
  4. Repeat step 1 until no swaps have been made

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
### Bubble Sort Complexity

<table>
<thead>
<tr>
<th>Worst Case</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted in Descending Order</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubble Up Largest Value = n</td>
<td></td>
</tr>
<tr>
<td>Bubble Up Next Largest Value = n-1</td>
<td></td>
</tr>
<tr>
<td>Bubble Up Next Largest Value = n-2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Smallest Value = 1</td>
<td></td>
</tr>
</tbody>
</table>
Worst Case
  – Sorted in Descending Order

Operations
  – Bubble Up Largest Value = n
  – Bubble Up Next Largest Value = n – 1
  – Bubble Up Next Largest Value = n – 2
  – ...
  – Smallest Value = 1

Complexity

\[ O(n^2) \]
Can we do better?
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
     3. Repeat Step 2 until the final array is reached

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
     3. Repeat Step 2 until the final array is reached

Example

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
Merge Sort

1. Recursively split the array in half until single elements remain
2. Merge two smaller arrays and return the sorted result
   1. Create an array of combined size
   2. Add elements from the two smaller arrays into the combined array in sorted order
3. Repeat Step 2 until the final array is reached

Example

0 1 2 3 4 5 6 7
10 8 7 6 12 5 11 9

0 1 2 3
10 8 7 6
0 1 2 3
12 5 11 9
**Merge Sort**

1. Recursively split the array in half until single elements remain
2. Merge two smaller arrays and return the sorted result
   1. Create an array of combined size
   2. Add elements from the two smaller arrays into the combined array in sorted order
3. Repeat Step 2 until the final array is reached

**Example**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
     3. Repeat Step 2 until the final array is reached

Example

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
    1. Create an array of combined size
    2. Add elements from the two smaller arrays into the combined array in sorted order
    3. Repeat Step 2 until the final array is reached

Example

```
 0 10
 8 0
 7 0
 6 0
 12 0
 5 0
 11 0
 9 0
```

```
0 1
```

```
0 0 0 0 0 0 0 0 0
10 8 7 6 12 5 11 9
```
• **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

**Example**

```
0 0
10 8
0 1
7 6
0 8
12 5
0 11
9
```

```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

0 0 0 0 0 0 0 0 0
10 8 7 6 12 5 11 9
0 1
8 10
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

```
0 0
10 8
```

```
0 0 0 0 0 0 0 0
7 6 12 5 11 9
```

```
0 1
8 10
```

```
0 1
```

```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
</table>
• **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

**Example**

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>9</td>
<td>11</td>
</tr>
</tbody>
</table>

[k] 0 1 2 3
```
- **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

**Example**

```
0 1
8 10
0 1
6 7
0 1
5 12
0 1
9 11
```

```
0 1 2 3
8 10
0 1
6 7
0 1
5 12
0 1
9 11
```

```
0 1 2 3
6
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
    1. Create an array of combined size
    2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

```
0 1
8 10
6 7
5 12
9 11
0 1 2 3
6 7 8 10
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
    1. Create an array of combined size
    2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

\[\begin{array}{c|c|c|c|c|c|c|c}
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
8 & 10 & 6 & 7 & 5 & 12 & 9 & 11 \\
\end{array}\]
• **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

---

**Example**

```
0 1 2 3
6 7 8 10
5 12 9 11
6 7 8 10
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

```
0 1
8 10
0 1
6 7
0 1
5 12
0 1
9 11
0 1 2 3
6 7 8 10
0 1 2 3
5 9 11 12
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>
Merge Sort

1. Recursively split the array in half until single elements remain
2. Merge two smaller arrays and return the sorted result
   1. Create an array of combined size
   2. Add elements from the two smaller arrays into the combined array in sorted order
3. Repeat Step 2 until the final array is reached

Example

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
6 & 7 & 8 & 10 \\
\end{array}
\quad
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
5 & 9 & 11 & 12 \\
\end{array}
\]

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
4 & 5 & 6 & 7 \\
\end{array}
\]
• **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
• **Merge Sort**
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

**Example**

```
0 1 2 3
6 7 8 10
0 1 2 3
5 9 11 12
```

```
0 1 2 3 4 5 6 7
5 6 7 8
```

```
0 1 2 3 4 5 6 7 0 1 2 3
5 6 7 8
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

**Example**

```
0 1 2 3
6 7 8 10

0 1 2 3 4 5 6 7
5 6 7 8 9
```

```
0 1 2 3
5 9 11 12

0 1 2 3 4 5 6 7
5 6 7 8 9
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example

```
0 1 2 3
6 7 8 10
0 1 2 3
5 9 11 12
```

```
0 1 2 3 4 5 6 7
5 6 7 8 9 10
```

```
0 1 2 3
6 7 8 10
5 9 11 12
```
• Merge Sort
  1. Recursively split the array in half until single elements remain
  2. Merge two smaller arrays and return the sorted result
     1. Create an array of combined size
     2. Add elements from the two smaller arrays into the combined array in sorted order
  3. Repeat Step 2 until the final array is reached

Example
Merge Sort

1. Recursively split the array in half until single elements remain
2. Merge two smaller arrays and return the sorted result
   1. Create an array of combined size
   2. Add elements from the two smaller arrays into the combined array in sorted order
   3. Repeat Step 2 until the final array is reached

Example

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Descending Order

• Operations
  – Split
  – Merge

**Complexity Visual**

<table>
<thead>
<tr>
<th></th>
<th>n/2</th>
<th>n/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/4</td>
<td>n/4</td>
<td>n/4</td>
</tr>
<tr>
<td>n/8</td>
<td>n/8</td>
<td>n/8</td>
</tr>
<tr>
<td>n/8</td>
<td>n/8</td>
<td>n/8</td>
</tr>
<tr>
<td>n/8</td>
<td>n/8</td>
<td>n/8</td>
</tr>
<tr>
<td>n/8</td>
<td>n/8</td>
<td>n/8</td>
</tr>
</tbody>
</table>

...
• Worst Case
  – Sorted in Descending Order
• Operations
  – Split
  – Merge

**Complexity Visual**

\[
\begin{array}{cccc}
  n & n/2 & n/2 \\
n/4 & n/4 & n/4 & n/4 \\
  n/8 & n/8 & n/8 & n/8 & n/8 & n/8 & n/8 & n/8 \\
  \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[\lg(n)\]
• Worst Case
  – Sorted in Descending Order
• Operations
  – Split
  – Merge

**Complexity Visual**

- $n$
- $n/2$
- $n/4$
- $n/8$
- $\ldots$
- $1$

- $\lg(n)$
- Worst Case
  - Sorted in Descending Order
- Operations
  - Split
  - Merge

Complexity

$O(n \lg(n))$
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Index: i, j
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>10</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>10</td>
<td>6</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
<td>12</td>
<td>5</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1.
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
• **Quick Sort**
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

**Example**

```
Index: 0 1 2 3 4 5 6 7
Value: 8 7 6 5 12 10 11 9
```

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

If value at j is smaller than the pivot then swap values at i and j and increase i by 1
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

### Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Swap values at i and the pivot
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
</tbody>
</table>

Swap values at i and the pivot
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Swap values at i and the pivot
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition.
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

---

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td></td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

```
Index  | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  |
Value  | 5  | 7  | 6  | 8  | 9  | 10 | 11 | 12 |
```

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large/equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

```
Index 0 1 2 3 4 5 6 7
Value  5  7  6  8  9 10 11 12
```

Recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
### Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

### Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition.
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition.
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
- Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort
1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition.
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
Quick Sort

1. Pick an arbitrary value called a “pivot” from the array
2. Using the pivot value “partition” the array
   1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
   2. Once it has been partitioned the pivot value is where it should be
3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

 recursively do the same for the values to the left of the partition and to the right of the partition
• Quick Sort
  1. Pick an arbitrary value called a “pivot” from the array
  2. Using the pivot value “partition” the array
     1. Reorder the array where smaller values are to the left of the pivot, and large / equal values are to the right
     2. Once it has been partitioned the pivot value is where it should be
  3. Recursively continue now with the array to the left, and the array to the right of the pivot

Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

Recursively do the same for the values to the left of the partition and to the right of the partition
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index
• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index
• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
- **Worst Case**
  - Sorted in Ascending Order
  - Assuming pivot is always picked from the last index

- **Operations**
  - The first index moves $n$ spaces
  - The first index moves $n-1$ spaces
  - The first index moves $n-2$ spaces
  - ... 
  - The first index moves 1 space
• **Worst Case**
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• **Operations**
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

---

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• Operations
  – The first index moves $n$ spaces
  – The first index moves $n-1$ spaces
  – The first index moves $n-2$ spaces
  – ...
  – The first index moves 1 space

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index
• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

Complexity Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
• **Worst Case**
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• **Operations**
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

---

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>
The text on the page is as follows:

- **Worst Case**
  - Sorted in Ascending Order
  - Assuming pivot is always picked from the last index
- **Operations**
  - The first index moves n spaces
  - The first index moves n-1 spaces
  - The first index moves n-2 spaces
  - ...
  - The first index moves 1 space

### Complexity Example

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

**pivot**
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index
• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• Operations
  – The first index moves $n$ spaces
  – The first index moves $n-1$ spaces
  – The first index moves $n-2$ spaces
  – …
  – The first index moves 1 space

---

**Complexity Example**

<table>
<thead>
<tr>
<th>Index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
</tbody>
</table>

The index $j$ starts at 4 and moves to 3, 2, 1, 0, while the index $i$ starts at 3 and moves to 2, 1, 0.
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index
• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space
• Worst Case
  – Sorted in Ascending Order
  – Assuming pivot is always picked from the last index

• Operations
  – The first index moves n spaces
  – The first index moves n-1 spaces
  – The first index moves n-2 spaces
  – ...
  – The first index moves 1 space

Complexity

$O(n^2)$
<table>
<thead>
<tr>
<th>Merge Sort</th>
<th>Quick Sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst Time Complexity = ( O(n \lg(n)) )</td>
<td>Worst Time Complexity = ( O(n^2) )</td>
</tr>
<tr>
<td>Average Time Complexity = ( \Theta(n \lg(n)) )</td>
<td>Average Time Complexity = ( \Theta(n \lg(n)) )</td>
</tr>
<tr>
<td>Worst Space Complexity = ( O(n) ) additional</td>
<td>Worst Space Complexity = ( O(\lg(n)) ) additional</td>
</tr>
</tbody>
</table>