Heaps
Definition: A data structure that can be defined recursively as a collection of nodes, where each node is a data structure consisting of a value, together with a list of references (edges) to nodes, with the constraints that no reference is duplicated, and none points to the root.
Trees

- Trees Have
  - Nodes
  - Edges
- Trees CANNOT
  - Contain Self-Referencing Edges
  - Have Cycles
  - Be Disjointed
• Binary Tree Structure
• Node’s data must be comparable
• Node’s have at most two children
  – Left Child
  – Right Child
• Max Heap: Children must be less than or equal to the parent
• Min Heap: Children must be greater than or equal to the parent
• Assume Leaves are NULL references
• Array Heap
• Assume Root is at Index 0
• Left Child Index = Parent Index * 2 + 1
• Right Child Index = Parent Index * 2 + 2
• Parent Index = (Child Index - 1) / 2

Array Max Heap

Max Heap

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

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>6</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index
• **Add**
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• **Bubble Up**
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

---

**Example Adding “13”**

![Heaps Diagram]

1. **Add**
   1. Replace the first leaf in breadth order with the new data
   2. From that node “bubble up” the data if necessary

2. **Bubble Up**
   1. If the child’s data is larger than the parent then swap that information
   2. Continue swapping child data with parent data until the parent is larger than the child or we reach the root index
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

Example Adding “13”
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

Example Adding “13”

```
[0] 12
[1] 8
[2] 9
[3] 7
[4] 5
[5] 6
[6] 13
```

```
null
null
null
null
null
null
```
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

Example Adding “13”
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

Example Adding “13”
• **Add**
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• **Bubble Up**
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

**Example Adding “13”**

```
[0]
  [1] 13
  [3] 8
  [4] 7
  [5] 5
  [6] 9
  [7] 6
  [8] null
  [9] null
  [10] null
  [12] null
```
• Add
  – Replace the first leaf in breadth order with the new data
  – From that node “bubble up” the data if necessary

• Bubble Up
  – If the child’s data is larger than the parent then swap that information
  – Continue swapping child data with parent data until the parent is larger than the child or we reach the root index

Example Adding “13”
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds
Heaps

• **Remove**
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• **Bubble Down**
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

**Example Remove**

Return Value: 12
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove

Return Value: 12
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove

Return Value: 12
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove

Return Value: 12
Heaps

- **Remove**
  - Store the data at the Root
  - Replace the Root data with the Data in the last node in Breadth Order
  - Starting from the root, “Bubble Down” that information
  - Return the stored value, previously at the root

- **Bubble Down**
  - Pick the larger of the 2 children
  - If its value is larger than the parent, then swap those values
  - Continue this until the parent’s value is larger or we reach the tree’s bounds

**Example Remove**

Return Value: 12
• **Remove**
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• **Bubble Down**
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

**Example Remove**

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

Return Value: 12
• **Remove**
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• **Bubble Down**
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

---

**Example Remove**

```
[0]
[1] 9
[3] 8
[7] 7

[2]
[4] 6
[5] 5
[6] null
[9] null
[8] null
[10] null
```
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

### Example Remove Again

Return Value: 9
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove Again

Return Value: 9
Remove
- Store the data at the Root
- Replace the Root data with the Data in the last node in Breadth Order
- Starting from the root, “Bubble Down” that information
- Return the stored value, previously at the root

Bubble Down
- Pick the larger of the 2 children
- If its value is larger than the parent, then swap those values
- Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove Again
Return Value: 9
• **Remove**
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• **Bubble Down**
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

---

**Example Remove Again**

Return Value: 9
• **Remove**
  - Store the data at the Root
  - Replace the Root data with the Data in the last node in Breadth Order
  - Starting from the root, “Bubble Down” that information
  - Return the stored value, previously at the root

• **Bubble Down**
  - Pick the larger of the 2 children
  - If its value is larger than the parent, then swap those values
  - Continue this until the parent’s value is larger or we reach the tree’s bounds

**Example Remove Again**

```
Return Value: 9
```

```
[0]
8

[1]
5

[4]
null

[3]
7

[2]
6

[6] null

[5] null

[7] null

[8]
```
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove Again

Return Value: 9
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

Example Remove Again

Return Value: 9
• **Remove**
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• **Bubble Down**
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds

---

**Example Remove Again**

```
8
7
6
5
null
null
null
null
null
null
null
null
null
null
```

Return Value: 9
• Remove
  – Store the data at the Root
  – Replace the Root data with the Data in the last node in Breadth Order
  – Starting from the root, “Bubble Down” that information
  – Return the stored value, previously at the root

• Bubble Down
  – Pick the larger of the 2 children
  – If its value is larger than the parent, then swap those values
  – Continue this until the parent’s value is larger or we reach the tree’s bounds