# Starting algorithms

false

Analyze algos by resource usage (time, space)

RAM model of computation

Instruction set.

Array of registers indexed by 0, 1, 2, ...

Each reg can contain an arbitrary finite sequence of bits, equivalently, an arbitrary integer.

Instructions include:

add $R_i$ to $R_j$, put result into $R_k$

$R_0, R_1, R_2, \ldots$

$R_c := R_i + R_j$

$R_k := R_i + R_j$

$R_k := R_i - R_j$

$R_k := R_i \times R_j$

$R_k := R_i / R_j$ integer div

$R_k := R_i \% R_j$ integer remainder

0 — false

non-0 — true

$R_k := R_i$ [and / or] $R_j$

not $R_j$

floating point operations

$R_k := R_{(R_i)}$

$R_{R_k} := R_i$

If $R_k \neq 0$ then goto L

Read $R_k$

Write $R_k$

Arrays

$A[1 \ldots n]$

stored in contiguous registers

$R_{10}, R_{11}, \ldots, R_{n+9}$

$A[1] \leftrightarrow R_{10}$

$A[2] \leftrightarrow R_{11}$

$A[R_3]$

---

1st approx:

Input of "size" $n$

Time taken by an algo will generally depend on $n$.

Different inputs of size $n$ may give different running times.

Worst case: max running time for input of size $n$

Best case: min .....

Avg case: avg ...

How do we measure time realistically?

---

1st stab:

1 time step for each instruction execution.

$R_{10} := R_{10} * R_{10}$

$10^6$ iterations: $R_{10} = 2^{10^6}$ | not reasonable

One solution: charge according to the number of bits in the register.

Sep 6-12:29 PM

Our solution:
count arithmetic ops as unit cost provided the numbers in the registers never exceed $c \lg n$ ($c$ constant indep of $n$)
↑ # of bits to store the value
$O(\lg n)$

Space requirements similar

Counting unit cost per instruction is reasonable.

---

We only care asymptotically (up to $\Theta(\cdots)$)

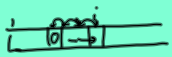
---

Insertion sort

Input: $A[1 \ldots n]$ of numbers

Output: $A[1 \ldots n]$ in ascending order.

```
For i := 2 to n do
    j := i-1
    while j > 0 and A[j] > A[j+1] do
        swap(A[j], A[j+1])
        j := j-1
    end while
end for
```




A

Analysis (worst case running time)

worst-case cost of inner while loop: $\Theta(i)$.

$i$'th iteration of for-loop takes time $\Theta(i)$.

ignore $\Theta$; call this $i$.

Total time for algo

$$\Theta\left(\sum_{i=2}^{n} i\right)$$

$$= \Theta\left(\sum_{i=1}^{n} i\right)$$

$$= \Theta\left(\frac{n(n+1)}{2}\right)$$

$$= \Theta(n^2)$$

---

```
for (i=1; i<n; i*=2)
    for (j=1; j<i; j++)
        1 → sum++;
```

$n \lg n$?
$n$?
$n 2^n$?

---

$$\text{Time}(n) = \sum_{k=0}^{\lfloor \lg n \rfloor} 2^k$$

($i = 2^k$ on $k$th iteration)

$$\frac{2^{\lfloor \lg n \rfloor + 1} - 1}{2 - 1}$$

$$\leq 2^{\lg n + 1} - 1$$

$$= 2n - 1 = O(n)$$

Time is also $\Omega(n)$
(last iteration of for-loop takes time $\geq \frac{n}{2}$.

$\therefore \Theta(n)$

Sep 6-12:55 PM

MergeSort:

Same input & output as insertion sort, except.

Mergesort(A, p, q)
// sorts A[p..q]
If $p \geq q$ then
  quit
else
  $m := \lfloor \frac{p+q}{2} \rfloor$
  Mergesort(A, p, m)  $T(\frac{n}{2})$
  Mergesort(A, m+1, q)  $T(\frac{n}{2})$
  → Merge(A[p..m], A[m+1..q])  $\Theta(q-p+1)$

Let $T(n)$ be the time taken by Mergesort to sort $n$ numbers.

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + \Theta(n)$$

Solve this to get

$$T(n) = \Theta(n \lg n)$$

Definition: A recurrence is an equation involving a function $T$ of $n$ where $T(n)$ is expressed in terms of $T$ of values smaller than $n$.

Example:

$$T(n) = 2T(n-1)$$
$$T(0) = 1$$

uniquely determine $T(n)$ for all natural numbers $n$.

$$\boxed{T(n) = 2^n}$$

solution of the recurrence.

$$T(0) = 0$$
$$T(n) = T(n-1) + n$$
$$T(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Proof that this works is by induction on $n$.

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + 1$$
$$T(n) = \Theta(\lg n)$$

Sep 6-1:29 PM