Week 1. Lecture Notes

Topics: Insertion Sort

Analysis of Insertion Sort

Recurrence of Merge sort

Substitution Method

The problem of Sorting

Input: a sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of numbers

Output: a permutation $\langle a_1', a_2', \ldots, a_n' \rangle$

such that

$$a_1' \leq a_2' \leq \ldots \leq a_n'$$

Example:

Input: 9 3 5 0 4 7

Output: 0 3 4 5 7 9

Pseudo Code: Insertion Sort

pseudo-code

```
INSERTION SORT (A, n)      ▷ A[1,... n]

   for j ← 1 to n
      do  key ← A[i]
          i ← j-1
          while  i>0 and A[i] > key
             do  A[i+1] ← A[i]
                 i ← i-1
          A[i+1] = key
```

Example of Insertion Sort

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 6 8 9

done

Running Time

- The running time depends on the input: an already sorted sequence is easier to sort.

- Parameterize the running time by the size of the input, since short sequences are easier to sort than the long ones.

- Generally, we seek upper bounds on the running time, because everybody likes a guarantee.

Types of Analysis

Worst-case: (Usually)

$T(n)$ = maximum time of algorithm on any input of size 'n'

Average-case: (Sometimes)

$T(n)$ = expected time of algorithm on any input of size 'n'

Best Case

Cheat with a slow algorithm that works fast on 'some' input

Machine-Independent Time

What is Insertion sort's worst-case time?

- It depends on the speed of our computer:
  - relative speed (on the same machine)
  - absolute speed (on different machines)

Big Idea

Ignore machine-dependent constants

Look at 'growth' of $T(n)$ as $n \to \infty$

"Asymptotic Analysis"

Θ notation

Math:

$\Theta(g(n)) = \{ f(n) : \exists$ positive constants c_1, c_2 and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ $\forall\, n \geq n_0 \}$

Engineering

Drop low-order terms; ignore leading constants

Example

$3n^3 - 90n^2 + 5n - 1024 = \Theta(n^3)$

O notation

$O(g(n)) = \{ f(n) : \exists$ positive constants c and n_0 such that $f(n) \le c g(n)$ $\forall\, n \geq n_0 \}$

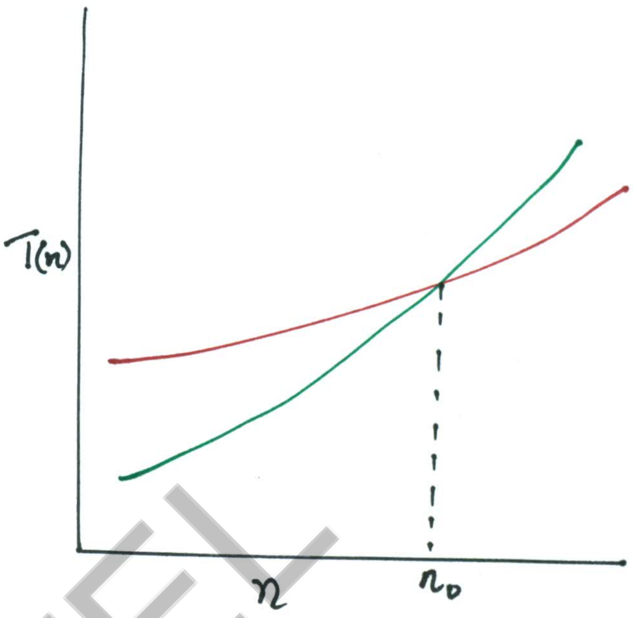
Ω notation

$\Omega(g(n)) = \{ f(n) : \exists$ positive constants c and n_0 such that $f(n) \geq c g(n)$ $\forall\, n \geq n_0 \}$

Asymptotic Performance

When n gets large enough a $\Theta(n^2)$ algorithm 'always' beats a $\Theta(n^3)$ algorithm

- We shouldn't ignore asymptotically slower algorithms
- Real world designs situation often calls for a careful balancing of engineering objectives
- Asymptotic analysis is a useful tool to help to structure our thinking



Insertion Sort Analysis

Worst Case: Input inverse sorted

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2) \quad \text{[arithmetic series]}$$

Average Case: All permutations equally likely

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

- Insertion sort is moderately fast for small 'n'
- It is not at all fast for large 'n'.

Merge Sort

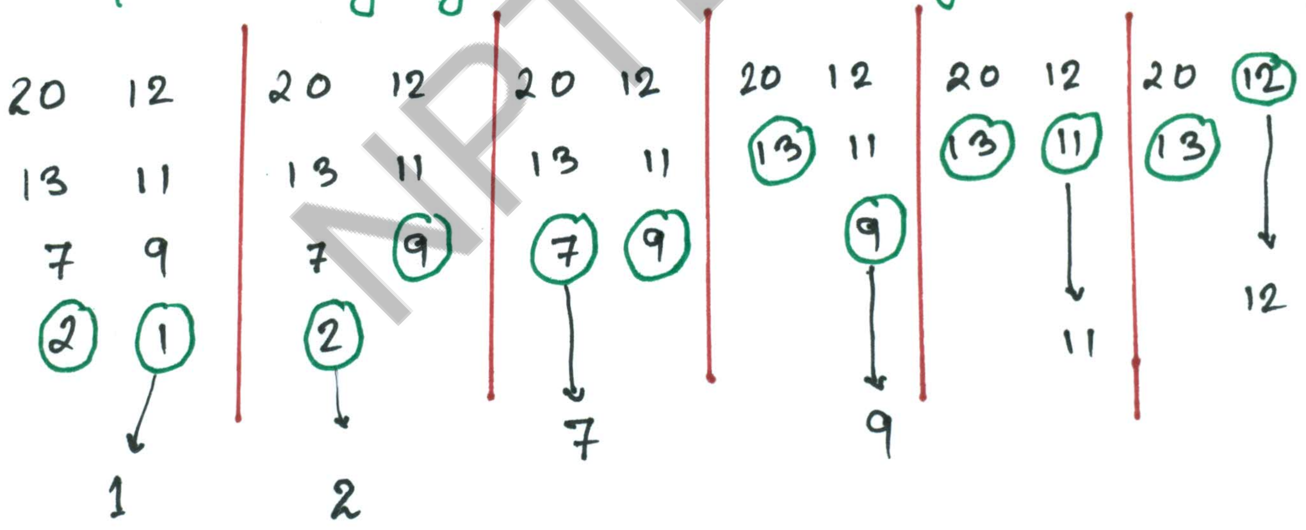
MERGE-SORT $A[1, \ldots, n]$

To sort n numbers

1. If $n=1$, done
2. Recursively sort $A[1, \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1, \ldots, n]$
3. "Merge" the 2 sorted lists

Key subroutine : MERGE

Example: Merging two sorted arrays



Solⁿ: 1 2 7 9 11 12 13 20

Time = $\Theta(n)$ to merge a total of n elements (linear time)

Analyzing Merge Sort

MERGE-SORT (A, n) ▷ $A[1, \ldots, n]$

$T(n)$	To sort n numbers
$\Theta(1)$	1. If $n=1$ done
$2T(n/2)$	2. Recursively sort $A[1, \ldots, \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil + 1, \ldots, n]$
$\Theta(n)$	3. Merge the 2 sorted lists

Should be $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$ but it turns out not to matter asymptotically

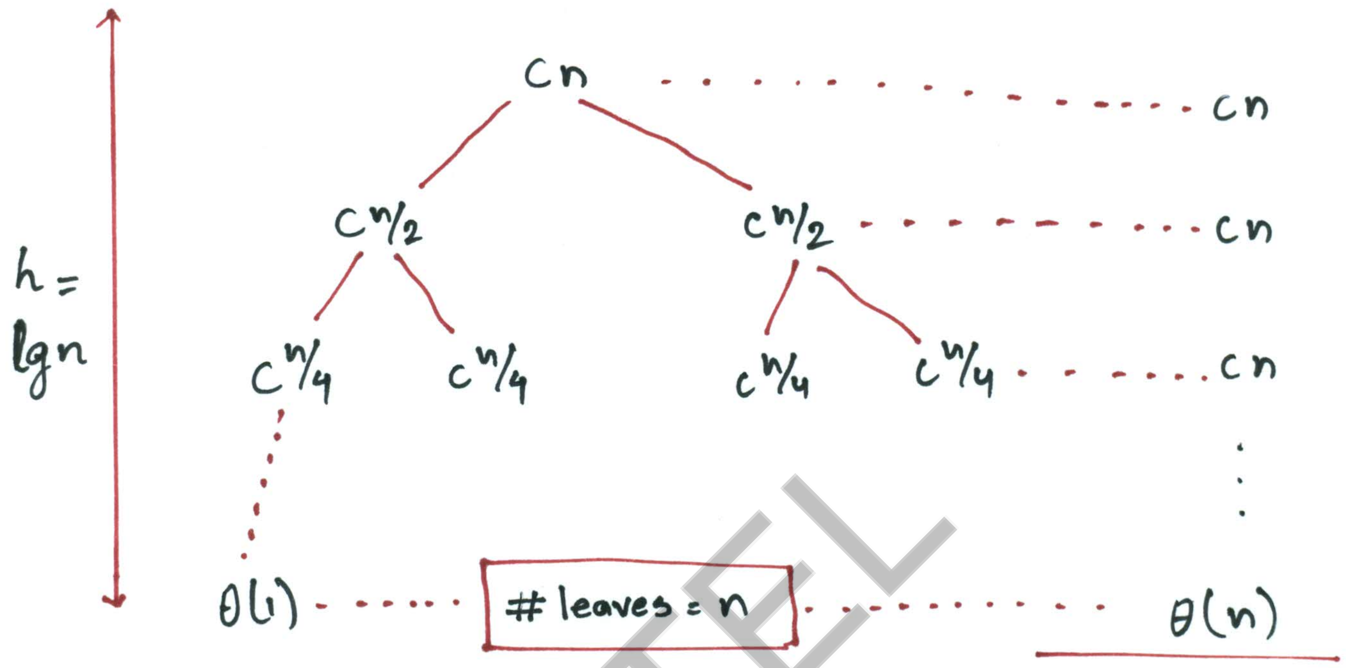
Recurrence for Merge Sort

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

- We shall usually omit the base case when $T(n) = \Theta(1)$ for sufficiently small n and when it has no effect on the solutions to the recurrence.

Recursion Tree

Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant



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

Conclusions

- $\Theta(n \lg n)$ grows more slowly than $\Theta(n^2)$
  Therefore, merge sort asymptotically beats insertion sort in the worst case
- In practice, merge sort beats insertion sort for $n > 30$ or so

Solving Recurrences : Substitution Method

It is the most general method:

1. Guess the form of solution
2. Verify by induction
3. Solve for constants

Example

$T(n) = 4T(n/2) + n$

- [Assume that $T(1) = \Theta(1)$]
- Guess $O(n^3)$
- Assume that $T(k) \le ck^3$ for $k < n$
- Prove $T(n) \le cn^3$ by induction.

Example of Substitution

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4c(n/2)^3 + n \\
&= (c/2)n^3 + n \\
&= cn^3 - ((c/2)n^3 - n) \quad \leftarrow \text{desired - residual} \\
&\le cn^3 \quad \leftarrow \text{desired}
\end{aligned}
$$

whenever $\underbrace{((c/2)n^3 - n)}_{\text{residual}} \ge 0$, for example if $c \ge 2$, $n \ge 1$

Example (Continued)

- We must also handle the initial conditions that is, ground the induction with base cases.

- **Base**: $T(n) = \Theta(1)$ for all $n < n_0$, where n_0 is a suitable constant

- For $1 \le n \le n_0$, we have "$\Theta(1)$" $\le cn^3$, if we pick c big enough

But this bound is not tight

A tighter upper bound

We shall prove that $T(n) = O(n^2)$

Assume that $T(k) \le ck^2$ for $k < n$:

$$
\begin{aligned}
T(n) &= 4T(n/2) + n \\
&\le 4cn^2 + n \\
&= \cancel{O(n)}
\end{aligned}
$$

wrong! We must prove the I.H.

$$
\begin{aligned}
&= cn^2 - (-n) \quad [\text{desired} - \text{residual}] \\
&\le cn^2
\end{aligned}
$$

for no choice of $c > 0$

We lose

A tighter upper bound

IDEA: Strengthen the inductive hypothesis

- Subtract a low-order term.

Induction hypothesis:

$$T(k) \le c_1 k^2 - c_2 k \text{ for } k < n$$

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\le 4\left(c_1 (n/2)^2 - c_2 (n/2)\right) + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= c_1 n^2 - c_2 n - (c_2 n - n) \\ &\le c_1 n^2 - c_2 n \quad \text{if } c_2 > 1 \end{aligned}$$

We pick c_1 big enough to handle this situation.