Date: __/__/20__ Lec 1 Day: ______

Analysis of Algorithms

The theoretical study of computer programs performance and resource usage

Whats more important than perf?
Correctness, User freindliness, Scalability, efficiency, etc.

Why study algs and perf?

- Enables efficient, scalable software
- Helps in choosing optimal problem solving methods.
- Ensures programs run fast & use resources wisely
- Prepare for real world demands and constraints.

Date: _/_/20__ Day: ______

Problem: Sorting

Input: sequence $\langle a_1, a_2, \dots, a_n \rangle$ of numbers.

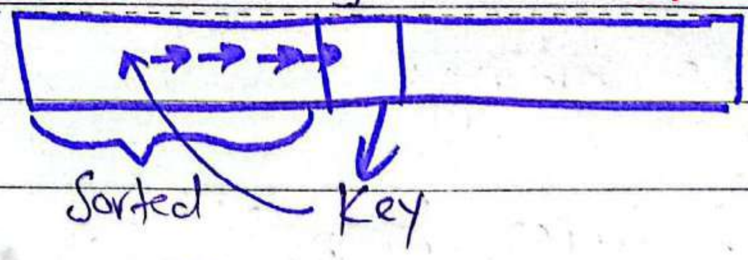
Output: permutation $\langle a_1', a_2', \dots, a_n' \rangle$ such that $a_1' \le a_2' \le \dots \le a_n'$

Insertion sort

```
Insertion-Sort (A,n) //Sorts A[1...n]
for j <- 2 to n
    do Key <- A[j]
    i <- j-1
    while i>0 and A[i]>Key
        do A[i+1] <- A[i]
           i <- i-1
    A[i+1] <- Key
```

Date: __/__/20__ Day: ______

A:



Ex: 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

- Depends on input (e.g already sorted, in reverse order)
- Depends on inputs size (6 elem vs 6×10^9)

  - parametrize in input size

Talk about time as the function of the size of the input that

Date: / /20 Day:

we are sorting.

- Want upper bounds
  We want to know that the time is no more than certain amount and the reason is because that represents a guarantee to the user.

Kinds of Analysis. Focus.

- Worst-case (usually)
  T(n) = max time on any input of size n.

- Average case (sometimes)
  T(n) = expected time over all inputs of size n
  - Expected type of input
  - To know the type of input (Need assumption of statistical distribution of inputs)

Date: __/__/20 No good Day: ______

- Best-case (bogus)
- You can cheat
- A slow algorithm that works fast on some input

What is insertion sort's worst-case time?

Depends on computer.

- relative speed (on same machine)
- absolute speed (on diff. machines)

BIG IDEA!

Asymptotic Analysis

1. Ignore machine dependent constants
2. Look at growth of $T(n)$ as $n \to \infty$

In short, to look at the efficiency of an algorithm and correctness irrelevant of the power of a machine.

Date: __/__/20__ Day: ______

Asymtotic Notation

Theta

- Θ - notation:

Drop lower-order terms
Ignore leading constants

Ex:

$$\underbrace{3}_{\text{Leading constant}}n^3 + \underbrace{90n^2 - 5n + 6046}_{\text{lower-order terms.}}$$

$$\Rightarrow \Theta(n^3)$$

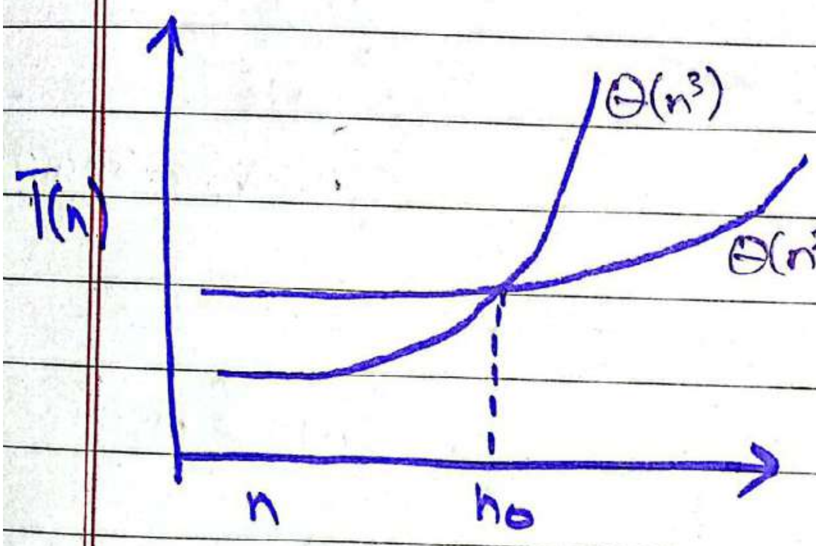
- Represents the upper and the lower bounds of the running time of an algorithm.

- Used to represent Averge-case complexity

- You add the running times for each possible input combination and take the average in the averge case

Date: __/__/20 Day: ______

- As $n \to \infty$, $\Theta(n^2)$ alg. always beats a $\Theta(n^3)$ alg.



- Even though alg. with $\Theta(n^3)$ time may be slower than algo. with $\Theta(n^2)$ time but they can be faster on reasonable size of inputs.

Date: __/__/20 Day: ______

Insertion Sort Analysis:

Worst Case: input reverse sorted.

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

(Arithmetic series)

Is insertion sort fast?

- Moderately so, for small n
- Not at all for large n.

Now, we look at a faster algorithm than insertion sort which is merge sort.

Date: / /20 Day:

Merge Sort

$T(n)$	Merge-Sort $A[1 \ldots n]$
$\Theta(1)$	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 2 sorted lists.

Key Subroutine: Merge

$\lceil n/2 \rceil$ 20   12 n

13   11

$\cancel{7}$ $\cancel{9}$

$\cancel{2}$ $\cancel{1}$

1     $\lceil n/2 \rceil + 1$

Compare each element of both list like as $1<2$, $2<9$, $7<9$, $9<13$

1 2 7 9 and so on

Time $= \Theta(n)$ on n total elements.

Date: _/_/20 Day:

Recurrence

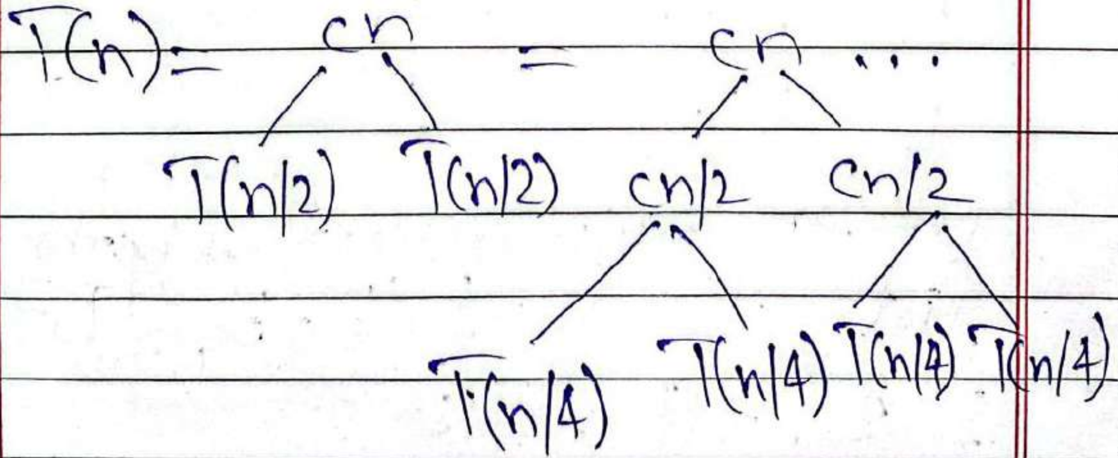
$$T(n) = \begin{cases} \Theta(1) & \text{if } n=1 \ \boxed{1} \\ 2T(n/2)+\Theta(n) & \text{if } n>1 \end{cases}$$

[1] We usually omit ~~base~~ base cases in recurrence because if you're something on constant size input it takes constant time.

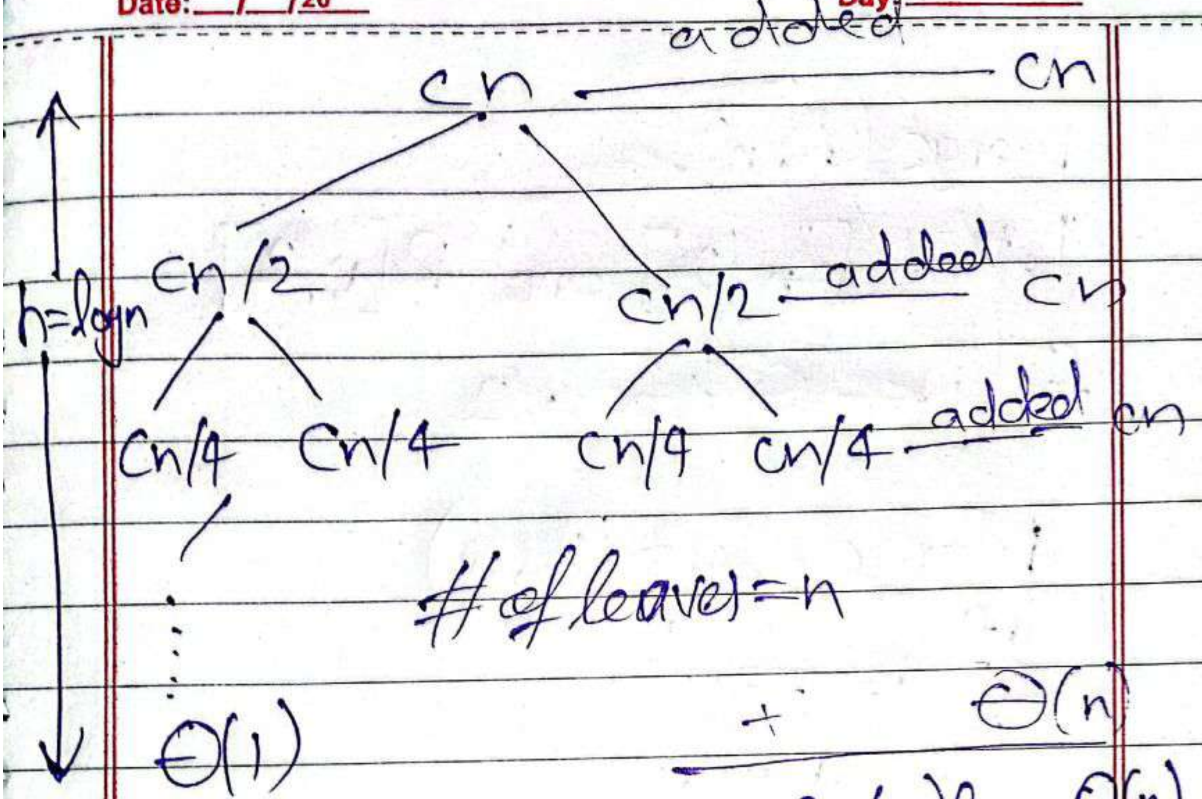
Recursion Tree:

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

$$c>0$$



Keep doing until you end up with.



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

$\Theta(n \lg n)$ is faster than $\Theta(n^2)$.
That's why Merge sort is faster than insertion sort. This goes for all algos.

So algos with time $\Theta(n \log n)$ like Merge sort will beat algs. with time $\Theta(n^2)$ like insertion sort on a large enough database.

A = array , p = 1st elem , q = midpoint , r = last elem

Date: __/__/20__ Day: ______

Merge (A, p, q, r)

$n_L \leftarrow q - p + 1$

$n_R \leftarrow r - q$

let $L[0 : n_L]$ and $R[0 : n_R]$

for $i \leftarrow 0$ to n_L

$L[i] \leftarrow A[p+i]$

for $j = 0$ to n_R

$R[j] \leftarrow A[q+j+1]$

$i = 0$

$j = 0$

$k = p$

while $i < n_L$ and $j < n_R$

if $L[i] \leq R[j]$

$A[k] \leftarrow L[i]$

$i \leftarrow i + 1$

else $A[k] \leftarrow R[j]$

Date: _/_/20 Day: ____

$j \leftarrow j+1$
$k \leftarrow k+1$

while $i < n_L$
$A[k] \leftarrow L[i]$
$i \leftarrow i+1$
$k \leftarrow k+1$

while $j < n_R$
$A[k] \leftarrow R[j]$
$j \leftarrow j+1$
$k \leftarrow k+1$

Delete Left and Right.

Date: / /20 Day:

Merge-Sort (A, p, r)

if $p \geq r$ return

$q = (p+r)/2$

Merge-Sort (A, p, q)

Merge-Sort (A, q+1, r)

Merge (A, p, q, r)