VII STRASSENS MATRIX MULTIPLICATION →

- Without divide & conqueor -

$A_{n\times n}$ $B_{n\times n}$

$C = A+B$

↳ n^2 elements $\Rightarrow O(n^2)$ [1 addition for 1 element]

$C = A \times B$

↳ every element → 3 multiplications (2X3, 3X4)

nXn elements → n multiplications $O(n^3)$

- With Divide & Conqueor -

Small problem if matrix sizes are equal or less than 2X2.

Given matrix size should be powers of 2, otherwise fractions will come.

∴ No. of rows = No. of columns [Square matrices]

Ex. $A = \begin{bmatrix} 10 & 11 & 12 & 13 \\ 14 & 15 & 16 & 17 \\ 18 & 19 & 20 & 21 \\ 22 & 23 & 24 & 25 \end{bmatrix}_{4\times4}$ (A_{11}, A_{12}, A_{21}, A_{22})

$B = \begin{bmatrix} 16 & 17 & 18 & 19 \\ 20 & 21 & 22 & 23 \\ 24 & 25 & 26 & 27 \\ 28 & 29 & 30 & 31 \end{bmatrix}_{4\times4}$ (B_{11}, B_{12}, B_{21}, B_{22})

$C = A \times B$

$C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}_{4\times4}$

Divide

$C_{11} = A_{11}B_{11} + A_{12}B_{12}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

1 mul nXn ⇓ 8 mul + 4 add. $\frac{n}{2}\times\frac{n}{2}$ $\frac{n\times n}{2\,2}$

1 mul 4X4 ⇓ 8 mul + 4 additions 2X2 2X2

Let T(n) be the time complexity of multiplying two matrices of size mxn (m=n)

$T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2$

$T(4) = 8T(4/2) + 4(2)^2$

$T(32) = 8T(32/2) + 4(32/2)^2$

Recurrance Relation-

$$T(n) = \begin{cases} O(1) & \text{if } n \le 2 \\ 8T(n/2) + 4(n/2)^2 & \text{if } n > 2 \end{cases}$$

$= 8T(n/2) + n^2$ (Simplified)

$= O(n^3)$ (using substitution)

Stack space = $\log_2 n$ (divide by 2)

Matrix multiplication in both cases will take same time, so consider space, non recursive is better because no need of stack space.

* Strassen matrix multiplication -

According to strassen

$$T(n) = \begin{cases} O(1) & \text{if } n \le 2 \\ 7T(n/2) + 16(n/2)^2 & n > 2. \end{cases}$$

$T(n) = 7T(n/2) + 4n^2$ (Simplified)

$= O(n^{2.81})$ (using substitution)

Home-work

1. An array of n-elements. Choose one which is (three elements) neither min nor max. Time complexity. O(1).
2. An array of n-element given. find no. of inversions.

A | 1 | 2 | 3 | 4 | 5 | 6 |
50 30

$1 < 2$ — inversion

$A[1] > A[2]$ $O(n \log n)$ Merge Algo

∴ An array of n points in the (x,y) plane is given find the closest pair. O(nlogn)

∴ An array of n elements in which until some place x all are increasing order after all are decreasing order is given. find the x. O(logn)

Master Theorem -

If RR is of the form

$T(n) = aT(n/b) + f(n)$ where a & b are constants and $a \geq 1, b > 1$ $f(n)$ = +ve f^n

case 1 if $f(n) = O(n^{\log_b a - \epsilon})$ where ϵ is constant $\epsilon > 0$

then $T(n) = \Theta(n^{\log_b a})$

case 2 if $f(n) = \Omega(n^{\log_b a + \epsilon})$ where $\epsilon > 0$

then $T(n) = \Theta(f(n))$

case 3 if $f(n) = \Theta(n^{\log_b a})$

then $T(n) = \Theta(n^{\log_b a} \cdot \log n)$

Comparing $f(n)$ and $n^{\log_b a}$ in case I, $n^{\log_b a}$ is polynomial (n^ϵ, $\epsilon > 0$) times greater.

Comparing $f(n)$ and $n^{\log_b a}$ in case 2, ~~$f(n)$~~ $n^{\log_b a}$ is polynomial (n^ϵ, $\epsilon > 0$) times smaller.

Comparing $f(n)$ and $n^{\log_b a}$ in case 3, both are asymptotically equal.

Ex 1. $T(n) = 64T(n/2) + n^4$

$n^{\log_2 64} = n^6$.

$f(n) = n^4$.

$T(n) = O(n^6)$.

Ex2. $T(n) = T(n/2) + c$.

$n^{\log_b a}$ $\quad n^{\log_2 1} = n^0$ $\quad f(n) = c = 1$

IIIrd case $\quad T(n) = \Theta(1 \cdot \log n)$

$= \Theta(\log n)$

Ex3 $\quad T(n) = 2T(n/2) + n^3$

$n^{\log_b a} = n^1$ $\quad f(n) = n^3$ (Greater one

$T(n) = \Theta(n^3)$ $\quad$ IInd case

Ex4 $\quad T(n) = 2^n T(n/2) + n^5$

$n^{\log_2 2^n} = n^n$

$f(n) = n^5$ $\quad$ So, $T(n)$ = Not applicable

a & b must be constants, not functions

Ex5. $\quad T(n) = 1.5\, T\left(\frac{n}{3/2}\right) + n^2$

$a = 1.5 \quad b = 1.5$ $\quad$ a cannot be fraction

So, $T(n)$ = Not applicable.

Ex6. $\quad T(n) = 25\, T\left(\frac{n}{5}\right) - n^3$

$a = 25 \quad b = 5 \quad n^{\log_b a} = n^2 \quad f(n) = -n^3$

$T(n)$ = not applicable $\quad$ -ve function

Ex7 $\quad T(n) = 2T(n/2) + n \log n$

$n^{\log_b a} = n^{\log_2 2} = n$.

$f(n) = n \log n$

$T(n)$ = Not applicable $\quad n^{\log_b a}$ is logarithmic times smaller than $f(n)$, not polynomially.

If recurrance rel^n contains root operator,

$T(n) = T(\sqrt{n}) + c$

$a = 1 \quad b$

1. Assume $n = 2^K$

$T(2^K) = T(2^{K/2}) + c$.

2. Assume $T(2^K) = S(K)$

$S(K) = S(K/2) + C$

$n^{\log_2 1} = n^0 = 1 \quad f(n) = 1$

$S(K) = \Theta(\log K)$

3. $S(K) = \Theta(\log K)$

$T(2^K) = \Theta(\log K)$

1. $T(2^K) = \Theta(\log K)$

$T(2^{\log_2 n}) = \Theta(\log \log_2 n)$

$T(n) = \Theta(\log \log_2 n)$

Ex-9 $T(n) = 2T(\sqrt{n}) + \log n$

1. Assume $n = 2^K$

$T(2^K) = 2T(2^{K/2}) + \log 2^K$

2. Assume $S(K) = T(2^K)$

$S(K) = 2S(K/2) + K$

$S(K) = \Theta(K \log K)$ 1

3 $S(K) = \Theta(K \log K)$

$T(2^K) = \Theta(K \log K)$

4. $T(n) = \Theta(\log_2 n \log(\log_2 n))$

~~If recurrance relation~~ / If $n^{\log_b a}$ is logarithmic time smaller than $f(n)$, then

$f(n) = \Theta(n^{\log_b a} \cdot (\log n)^K)$

where $K \geq 0$ then

$f(n) = \Theta(n^{\log_b a} (\log n)^{K+1})$

Ex 10 $T(n) = 2T(n/2) + n \log n$

$a = 2 \quad b = 2 \quad f(n) = n \log n \quad n^{\log_b a} = n$

smaller by logarithmic

$n \log n = \Theta(n (\log n)^1)$ then $T(n) = (n (\log n)^2)$

Problem 1

Consider the following C program -

```
A(n) {
  if (n ≤ 1)  return (100);
  else
  return A(n/2) + A(n/2) + n;
}
```

→ Not a function call

Recurrance Relation -

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

$T(n) = \Theta(n)$ (Time complexity)

* functions don't make up complexity, only function calls contribute. (In addition to loops)

Output of program

$T(n) = T(n/2) + T(n/2) + n$ (what return ?)

1

Problem 2.

Consider the following C program -

```
A(n) {
if (n ≤ 1)  return (n² + n + 1)
else
return ( 5A(n/2) + 3A(n/2) + mA(n));
}
```

→ Matrix addition

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ 5T(n/2) + 3T(n/2) + n^2 & \text{if } n > 1 \end{cases}$$

Time complexity $T(n) = 8T(n/2) + n^2$

$n^{\log_2 8} = n^3$ $\quad T(n) = O(n^3)$.

$f(n) = n^2$ $\quad n^{\log_b a} = n$.

* Multiplications, divisions, additions take only constant time

return (5T(n/2) - 3T(n/2) / mA(n)).

$O(n^2)$

Only function calls matter.

Problem 3

Consider the following C program:

```
A(n) {
  if (n<=1) return 1;
  else
  return (n*A(n-1))
  }
```

```
b = A(n-1)      T(n-1)
c = n*b          1
return c
```

Time complexity

$$T(n) = \begin{cases} O(1) & \text{if } n \le 1 \\ T(n-1)+c & \text{if } n > 1 \end{cases}$$

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

Output

$$T(n) = \begin{cases} 1 & \text{if } n \le 1 \\ n * T(n-1) & \text{if } n > 1 \end{cases}$$

$T(n) = n! = O(n^n)$

Multiplications

$$T(n) = \begin{cases} 0 & \text{if } n \le 1 \\ T(n-1)+1 & \text{if } n > 1 \end{cases}$$

$T(n) = n-1$

Problem 4 Consider the following C program:

```
fibb(n)
{
```

```
if (fibb n==0) return 0;
 if (n==1) return 1;
 else
 return ( fibb(n-1)+ fibb(n-2) );
 }
```

find RR for time complexity, O/p, additions

Output -

$$T(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ T(n-1)+T(n-2) & \text{if } n>1 \end{cases}$$

Time complexity-

$$T(n) = \begin{cases} O(1) & \text{if } n=0 \text{ or } n=1 \\ T(n-1)+T(n-2)+c & \text{if } n>1 \end{cases}$$

$$= O(2^n)$$

Additions-

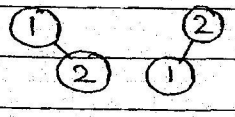
$$T(n) = \begin{cases} 0 & \text{if } n=0 \text{ or } n=1 \\ T(n-1)+T(n-2)+1 & \text{if } n>1 \end{cases}$$

HEAP SORT =

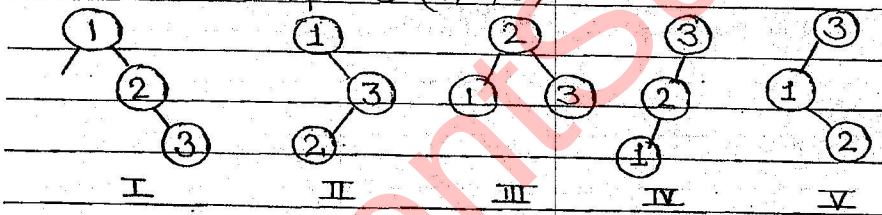
Binary tree - atmost two [0, 1 or 2]
Strict binary tree - strictly two or zero [0, 2] or full binary tree
Binary search tree - labels on nodes. Comparing root, left nodes are less and right are more.

How many binary search trees are possible with given n nodes?

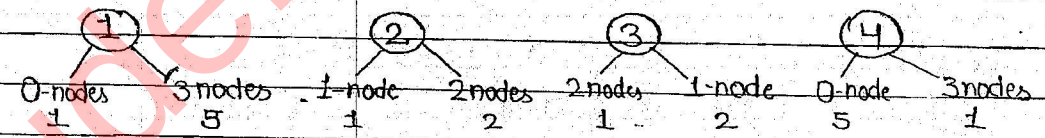
$n = 0$ — 1 empty BST
$n = 1$ — 1 BST
$n = 2$ (1,2) — 2 BST



$n = 3$ (1,2,3)

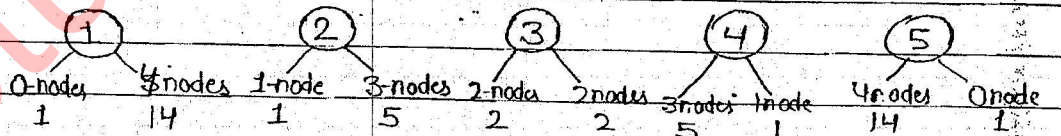


$n = 4$



$1\times5 + 1\times2 + 2\times1 + 5\times1 = 14$ BST

$n = 5$



$1\times14 + 1\times5 + 2\times2 + 5\times1 + 14\times1 = 42$ BST

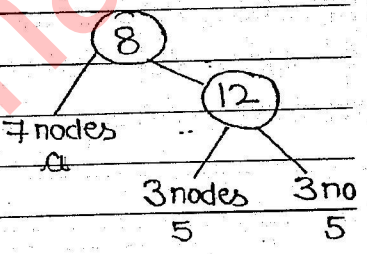
Let T(n) be no. of ~~no~~ BST possible with n number of nodes, then

$$T(n) = \sum_{i=0}^{n-1} T(i) \cdot T(n-(i+1)) \qquad T(0)=1 \quad T(1)=1$$

No. of BST possible = $\frac{2nC_n}{n+1}$

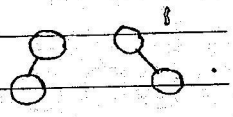
$T(n) = T(L) + T(R)$ $\quad L+R+1 = n$

1. How many BST possible with 15 nodes (1,2 .. 15) in such a way that in all those BST, 8 will be the root and in right hand side 12, will be the root



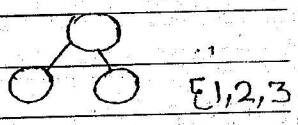
for 7-nodes,
No. of BST = a (say).
Answer is $a \times 5 \times 5$
$= 25a$.

2. How many un-labeled binary trees possible with the n nodes?



n=2 2 BST
n=3 5 BST

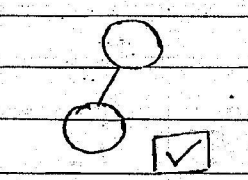
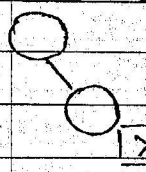
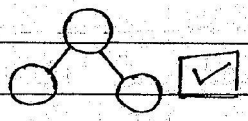
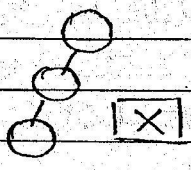
Given an unlabelled binary tree-
No. of BST - 1 way
No. of binary trees - n! ways



No. of unlabelled Binary tree = No. of BST

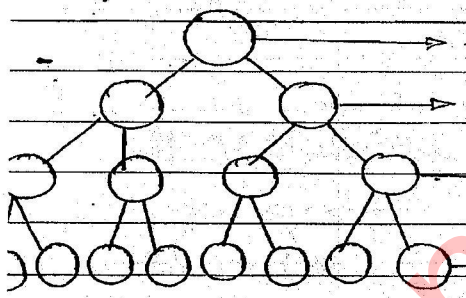
3. Given n nodes, how many BST are possible if the no. of levels are n. (n=6)

Almost complete binary tree → first complete left, then go to right. first complete current level, then go to next

Right to left few gaps may be there. No chance of intermediate gaps.

Complete binary tree → No gaps present. A special type of ACBT.



level 1 2^0
level 2 2^1 N-level
level 3 2^2 No. of nodes $2^n - 1$
level 4 2^3 $[2^0+2^1+2^2+\dots 2^n]$

1. If CBT contain K-levels, total nodes $2^K - 1$
2. In CBT, at K^{th} level, total nodes 2^{K-1}
3. In CBT, the no. of leaf nodes = $\lceil$ total nodes / 2 $\rceil$
   non-leaf / internal nodes = $\lfloor$ total nodes / 2 $\rfloor$
4. In CBT n-nodes, K-levels
   $n = 2^K - 1$
   $K = \log_2(n+1)$
5. Height → root to leaf no. of edges [longest]
   no. of levels − 1 $h = \log_2(n+1) - 1$

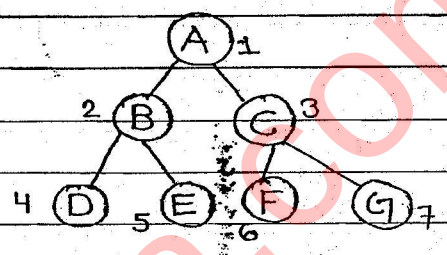
~~AVL tree is a balanced binary search tree.~~

How to represent binary tree ?

Array representation -

Ex1

A	B	C	D	E	F	G
1	2	3	4	5	6	7

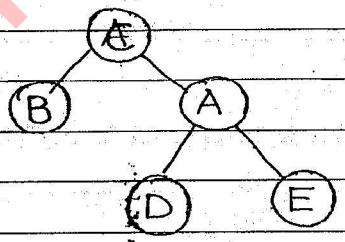


for ith position, the node is stored, then (i) parent = $\lfloor i/2 \rfloor$

(ii) Left child = $2i$ (iii) Right child = $2i+1$

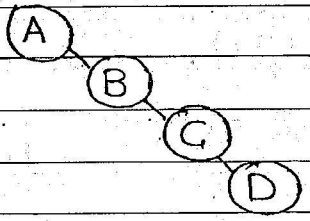
Ex2.

C	B	A	-	-	D	E
1	2	3	4	5	6	7



Ex3

A		B				G								D
1	2	3	4	5	6	7								15

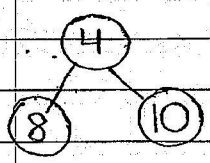


A binary tree contains n nodes

min array size n

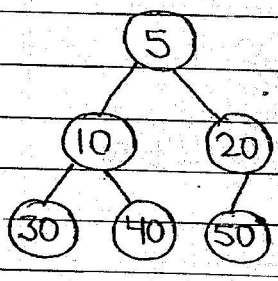
max array size $2^n - 1$.

⇒ Min-Heap tree - In a almost complete binary tree, the root is always smaller than the child nodes.

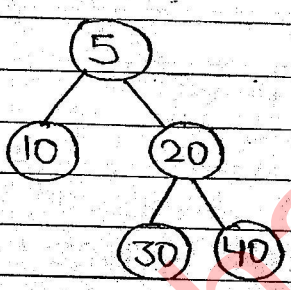


Page No. 10
Date : 05/03/16

Definition - At every node, root is minimum or equal comparing its children and the tree must be almost complete binary tree.

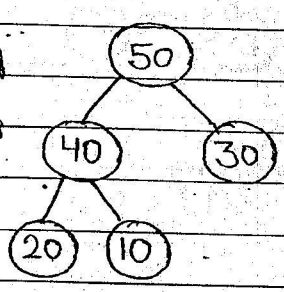


✓

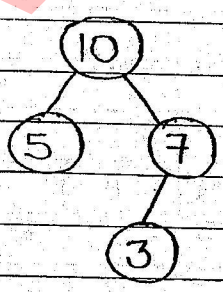


X

= Max-heap tree - In the given ACBT, root is always maximum or equal comp its children at each node.



✓



X

Not mentioned (heap tree)

Max-heap tree is meant for maximum elements.

To find Ist max

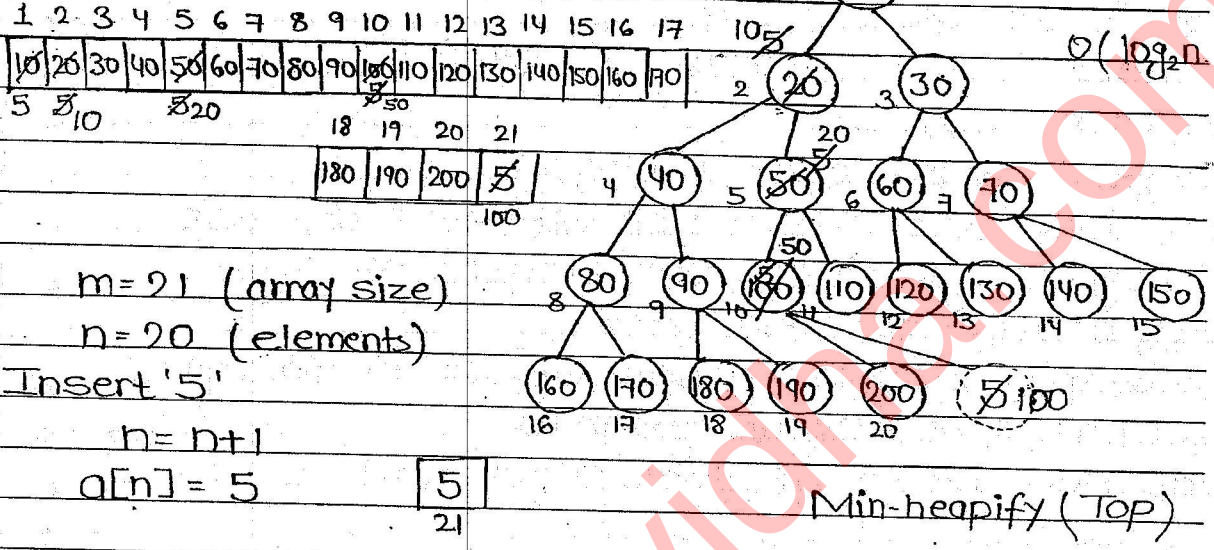
1 element - 1 comparison - O(1).

To find n^{th} max or min

Min-heap tree. Minimum element O(1)] finding.
Maximum element O(n).]

maximum finding no advantage

Insertion

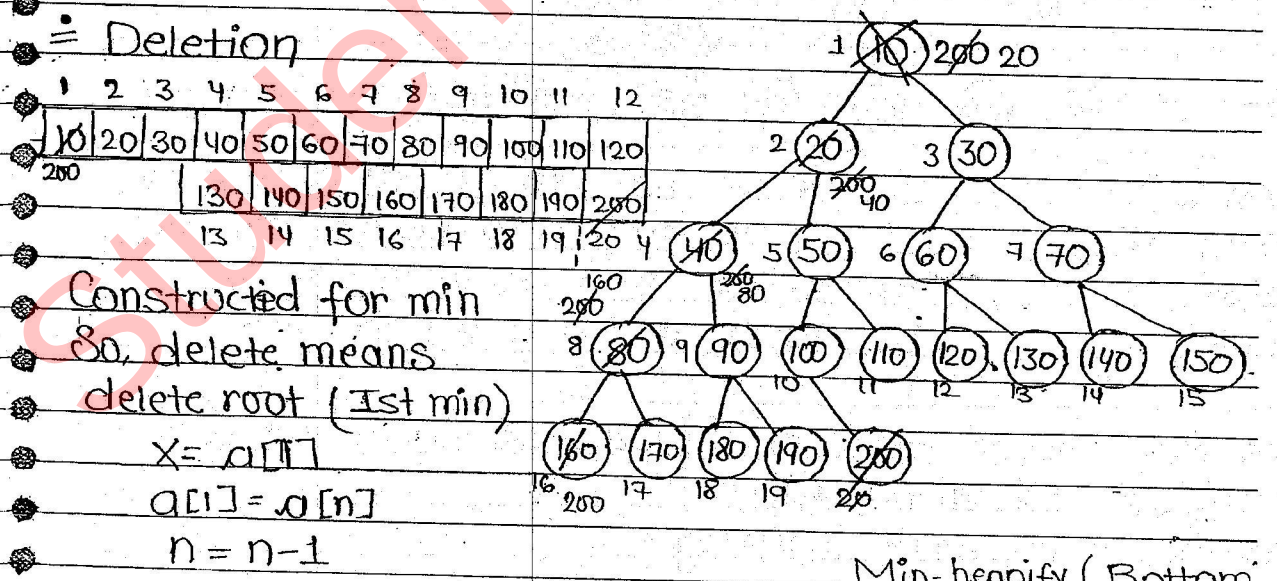


m = 21 (array size)
n = 20 (elements)
Insert '5'
n = n+1
a[n] = 5

5
21

No. of comparisons - $\log_2 n$
No. of swaps - $\log_2 n$. Total - $2\log_2 n$
Insertion time complexity - $O(\log_2 n)$
Best case $O(1)$ Average case $= \frac{1+2+3+\ldots \log n}{\log n}$
Worst case $O(\log_2 n)$ $O(\log n)$

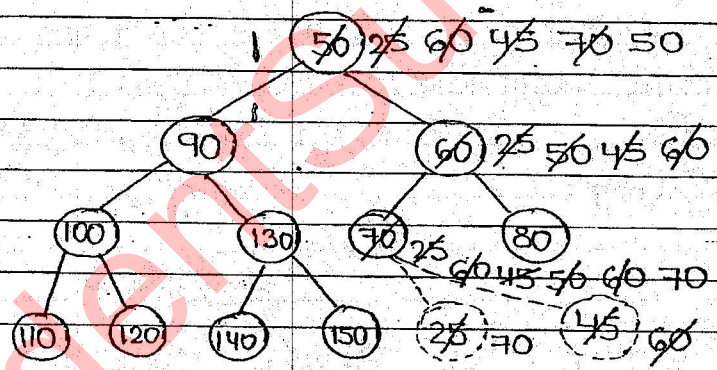
Deletion



Constructed for min
So, delete means
delete root (1st min)
X = a[1]
a[1] = a[n]
n = n-1

No. of comparisons $2.\log_2 n$ [2 comp at each level]
No. of swaps $\log_2 n$
Total $3\log n = O(\log_2 n)$
Best case $O(1)$
Average or Worst case $O(\log n)$
Insertion, deletion in min-heap or max-heap will take $O(\log n)$ as worst case, average case & $O(1)$ as best case.
Min-heap will give descending order & max-heap will give ascending order.
Heap sort is an inplace algorithm.

Problem- Consider the following min-heap

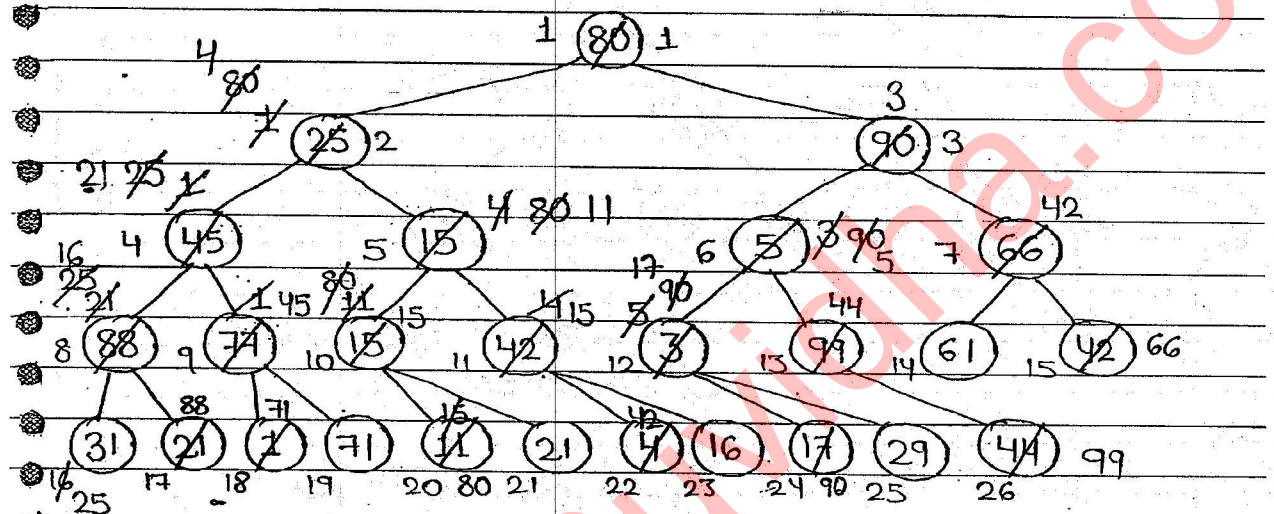


Insert 25	25 90 50 100 130 60 80 110 120 140 150 70
Insert 45	25 90 45 100 130 50 80 110 120 140 150 70 60
Delete	45 90 50 100 130 60 80 110 120 140 150 70
Delete	50 90 60 100 130 70 80 110 120 140 150

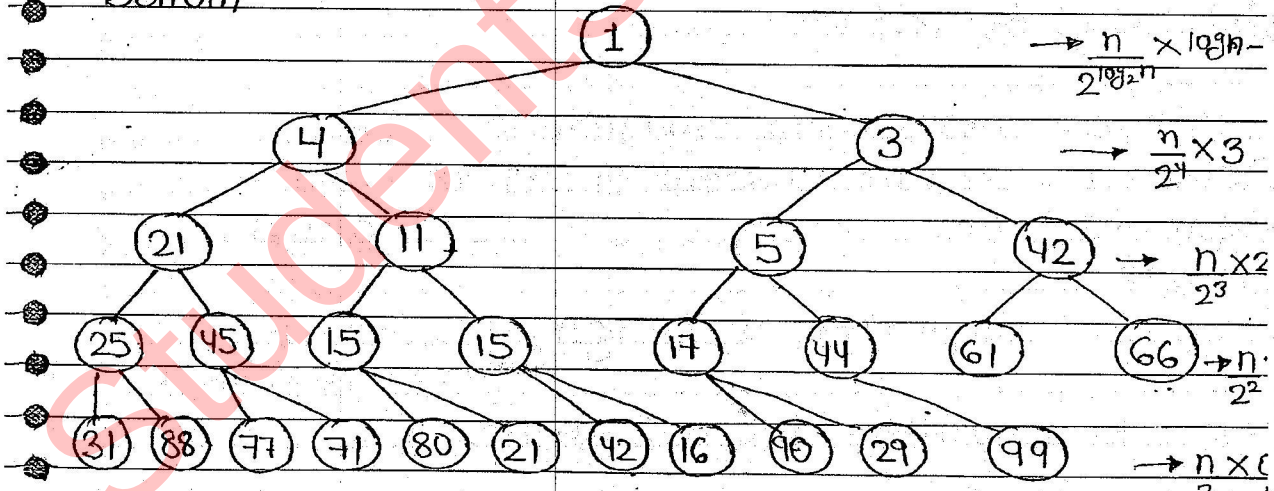
Insertion always from last | Deletion from the top.
finding 7th minimum $O(1)$
deleting 7th minimum $O(7\log n)$
Creating min-heap for n element $O(n\log n)$

≡ Build heap method - building a heap with O(n) given an array to form heap

1. i/p 80 25 90 45 15 5 66 88 77 15 42 3 99 61 42 31 21 1, 71 11 21 4 16 17 29 44



Apply 'for' loop in reverse & follow min-heapify bottom

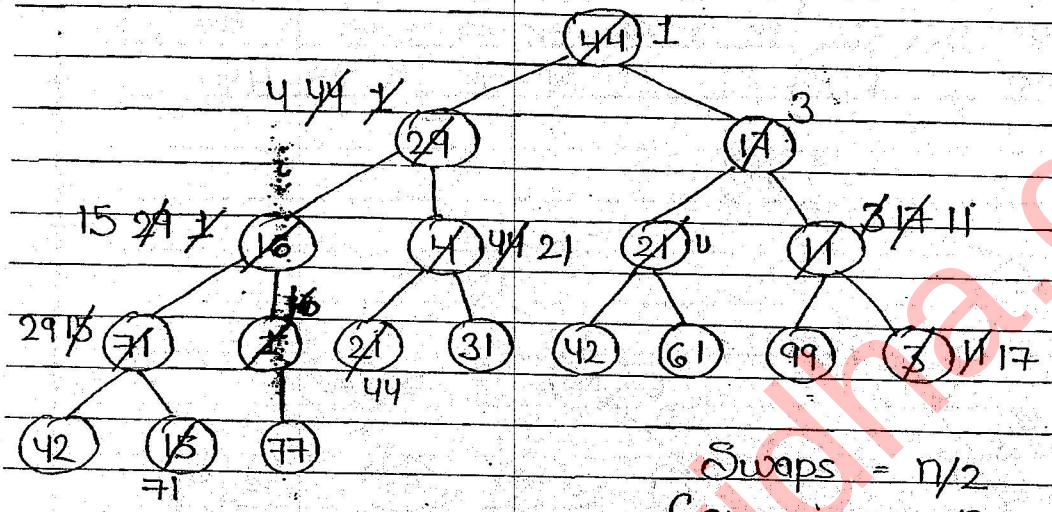


Comparisons = 2 × Swaps

Time complexity O(n)

Total Swaps = $\frac{n}{2}\times 0 + \frac{n}{2}\times 2 + \frac{n}{2}\times 2^2 + \dots \frac{n}{2^{\log_2 n}}(\log n - 1) = O(n)$

∴ i/p 44 29 17 16 4 21 11 71 1 21 31 42 61
99 3 42 15 77



Swaps = n/2
Comparisons n.

Answer
Heap is : 1 4 3 15 21 21 11 29 16 44 31 42
61 99 17 42 71 77

≡ Heap Sort Algorithm -

1. Create max-heap using build-heap O(n)
2. Delete one by one element to store from right to left O(nlogn)
3. Return

Time complexity O(nlogn) [Every case]
[Distinct elements]

[Heap sort algorithm is not stable.]

Instead of linear search, if you use binary search while insertion in min-heap, the no. of comparisons is log(logn) but swaps are logn. Then time complexity remains O(logn).

Bubble sort -

i/p : A [77 54 92 61 120 32 15 21]
1 2 3 4 5 6 7 8

1st pass
[54 77 61 92 32 15 21] 120 — 7 comp.

2nd pass
[54 61 77 32 15 21] 92 120 — 6 comp.

3rd pass
[54 61 32 75 21] 77 92 120 — 5 comp.

4th pass
[54 32 15 21] 61 77 92 120 — 4 comp.

5th pass
[32 15 21] 54 61 77 92 120 — 3 comp.

6th pass
[15 21] 32 54 61 77 92 120 — 2 comp.

7th pass
15 21 32 54 61 77 92 120 — 1 comp.

Increase-key decrease-key operations will take O(logn) time in min-heap as worst/av case. O(1) a best case.

Search operati in min-heap tree is O(n as linear sea

1. Bubble sort requires (n-1) passes.
2. Total comparisons $(n-1)+(n-2)+\dots 1$
$= n(n+1)/2 = O(n^2)$ [Every case]
3. Total swaps $(n-1)n/2 = O(n^2)$ [Worst & Average]
$0 = O(1)$ [Best case]
4. Time complexity = Swaps + comparisons
$= O(n^2)$ (Every case)
5. Inplace algorithm
6. Stable algorithm
7. No swaps in any pass, stop, array sorted. $O(n)$ as best case. $O(n^2)$ as worst & average case.

Selection Sort -

i/p - [77 54 92 61 120 32 15 21]

1st pass	15 [54 92 61 120 32 77 21]	7 comp, 1 swap
2nd pass	15 21 [92 61 120 32 77 54]	6 comp, 1 swap
3rd pass	15 21 32 [61 120 92 77 54]	5 comp, 1 swap
4th pass	15 21 32 54 [120 92 77 61]	4 comp, 1 swap
5th pass	15 21 32 54 61 [92 77 120]	3 comp, 1 swap
6th pass	15 21 32 54 61 77 [92 120]	2 comp, 1 swap
7th pass	15 21 32 54 61 77 92 [120]	1 comp, 1 swap

- Selection sort requires (n-1) passes
- Total comparisons $(n-1)+(n-2)+\cdots$ th
  $O(n^2)$ Every case
  Inplace + Stable
- Total swaps n-1 Every case
  In worst case, the no. of swaps O(n)
- Time complexity $O(n^2)$ [BC, WC, AC]
- Meant for swap operations in worst case.

Insertion - sort -

i/p [50 | 80 120 90 70 40 150 5 65]

Asking for swap

1st pass	50 80	120 90 70 40 150 5 65
2nd pass	50 80 120	90 70 40 150 5 65
3rd pass	50 80 90 120	70 40 150 5 65
4th pass	50 70 80 90 120	40 150 5 65
5th pass	40 50 70 80 90 120	150 5 65
6th pass	5 40 50 70 80 90 120 150	65
7th pass	5 40 50 65 70 80 90 120 150	

1. Best case -
   Total passes n-1
   Total comparisons n-1 [1 for each pass]
   Total sort swaps 0 ⇒ O(n)
2. Worst case -
   Total passes n-1
   Total comparisons 1+2+3+...n-1 = $O(n^2)$
   Total swaps 1+2+3+...n-1 = $O(n^2)$
   Time complexity ⇒ $O(n^2)$
3. Insertion sort is meant for best case only.
4. In the given array, if most of elements are sort then, insertion sort will give best case O(n) bu quick sort will give worst case $O(n^2)$
5. No. of inversions in given array = no. of swap operations in insertion sort.
6. In given array, if atmost n inversions are there then most of elements are sorted. At that time, insertion sort will give best case O(n).

10	20	30	40	50	5
1	2	3	4	5	6

7. Average case
   $\frac{n}{2}$ best + $\frac{n}{2}$ worst

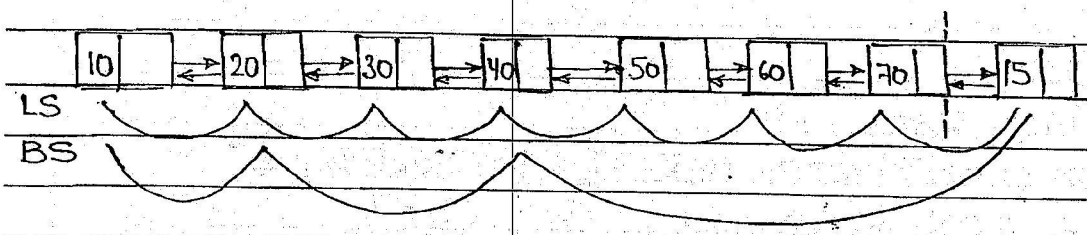
   $\frac{n}{2} \times 1 + \frac{n}{2} \times n \Rightarrow \frac{n}{2} + \frac{n^2}{2} = O(n^2)$

8.

10	20	30	40	50	60	70	5

LS
BS

Linear search - (1-element ↔ n comp ↔ n swaps) n-element $O(n^2$
Binary search - (1element ↔ logn comp ↔ n swaps) n-element $O(n^2$



Linear search
1- element
n comparisons
0 swaps (only link)
n
n- element
$n \cdot n = O(n^2)$

Binary search
1-element
log n comparisons
No swaps
log n
n- element
$n \cdot \log n = O(n \log n)$

∴ Binary search not possible for linked lists.

SORTING

Comparison based

Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	BC	AC	WC

Non-comparison based

Counting, Radix, Bucket — $O(n)$

Certain Limitations
(To be read....)

∴ In all comparison based sorting algorithms worst cases
lower bound $O(n \log n)$ upper bound (n^2)
Best case
lower bound $O(n)$ upper bound $O(n^2)$

≡ Counting Sort

- ✓ Counting sort works when a particular range of numbers is given
- ✓ It does not work for negitive numbers.
- ✓ Assumes that every element is a small integer.
- ✓ Sorting is based on some numeric keys.

Ex-1 Input 10 7 12 4 9 13

Range : 4 to 13

Index :	4	5	6	7	8	9	10	11	12	13
Count :	0	0	0	1	0	0	1	0	1	1
Sumcount :	1	1	1	2	2	3	4	4	5	6

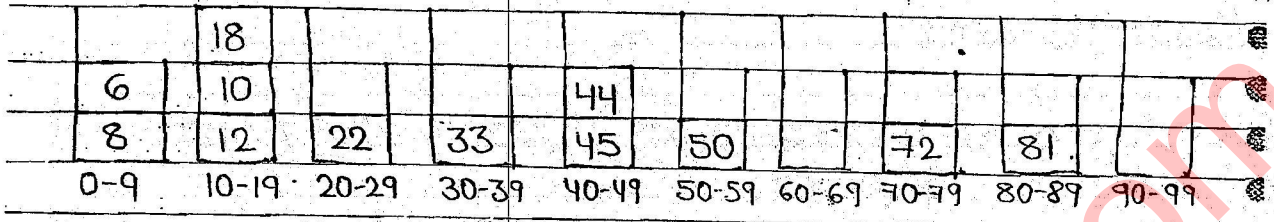
Position :	1	2	3	4	5	6
Sorted I/P :	4	7	9	10	12	13

≡ Bucket Sort

- ✓ In this algorithm, we create buckets and put elements into them.
- ✓ Then we apply some sorting algorithm (Insertio sort) to sort elments in each bucket.
- ✓ Finally we take elments out and join them to get the sorted result.

Ex-1 Input 22 45 12 8 10 6 72 81 33 18 50

To sort these elements, we will take 10 bucke



Sort each bucket using some sorting algorithm & taking the elements back.

Sorted i/p : 6 8 10 12 18 22 33 44 45 50 72 81

Radix Sort :-

Sort the numbers from last significant digit to the most significant digit.

The no. of digits in the numbers should be same.

Ex-1 Input : 10, 15, 1, 60, 5, 100, 25, 50

Padding : 010, 015, 001, 060, 005, 100, 025, 050

Pass 1 : Sort according to Ist LSB.

010	001	015	100
060		005	
100		025	
050			

010, 060, 100, 050, 001
015, 005, 025

Pass 2 : Sort according to 2nd digit

100	010	025	050	060
001	015			
005				

100, 010, 025, 050, 060
001, 015, 005

Pass 3 : Sort according to MSB.

010	⇒	001
025		005
050		010
060		015
		100

001, 005, 010, 015, 025
050, 060, 100