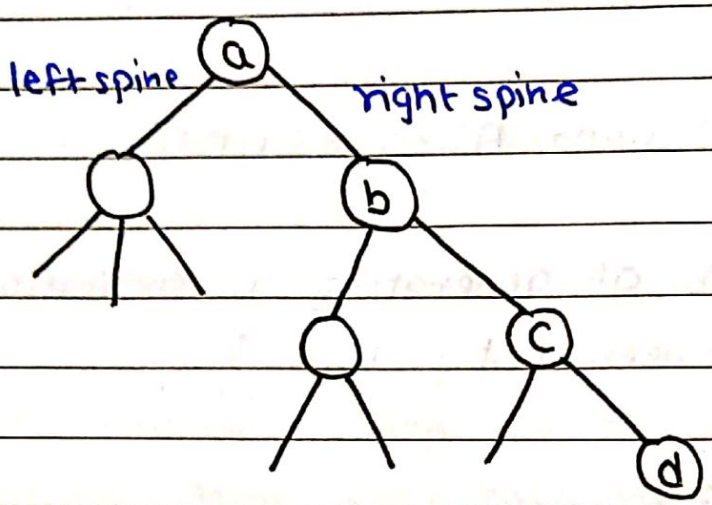
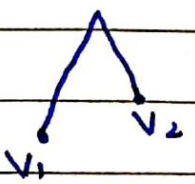
* Tree



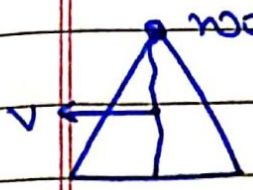
a, b, c, d : ancestor of d
a, b, c : proper ancestor of d

* Simple path: No vertex repeated
* root is at level 0
* Node with no child: leaf
* max depth of leaf node: height of tree
* no. of edges connecting root & node v is depth of node v

* K-ary tree: every internal node ≤ k child
* full k-ary tree: exactly k child
* Complete k-ary tree: K-ary tree + every leaf at same level
* Nearly k-ary tree:

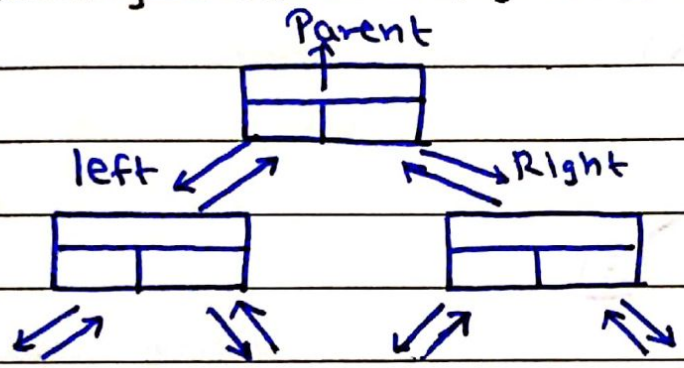


$|depth(v_1) - depth(v_2)| \leq 1$

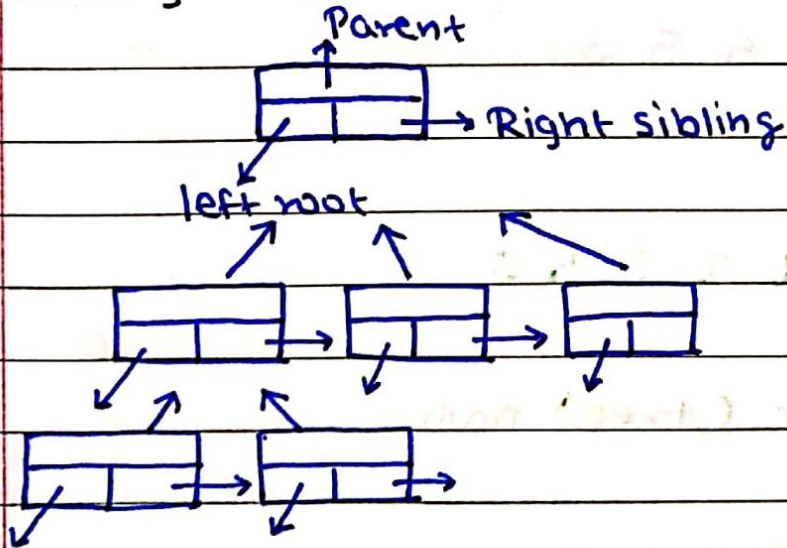


representation of a tree

★ Binary Tree (2-any tree)

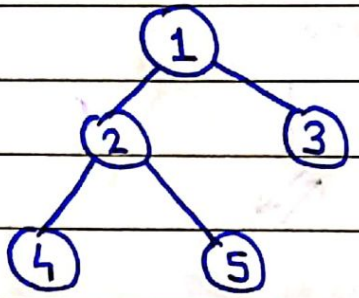


★ K-any Tree (variable K)



* Inorder : left, root, right
* Postorder: left, right, root
* Preorder: root, left, right

* Level order (BFS)



- Inorder: 4 2 5 1 3
- Preorder: 1 2 4 5 3
- Postorder: 4 5 2 3 1

- Level order 1 2 3 4 5

* void postorder (tree* node)
{
    if (node == NULL) return;
    postorder (node -> left);
    postorder (node -> right);
    Func (node -> data);
}

* void inorder (tree* node) (Using system stack)
# recursion

```
{
   if (node == NULL) return;
   inorder (node -> left);
    func (node -> data);
   inorder (node -> right);
}
```

* Using stack S

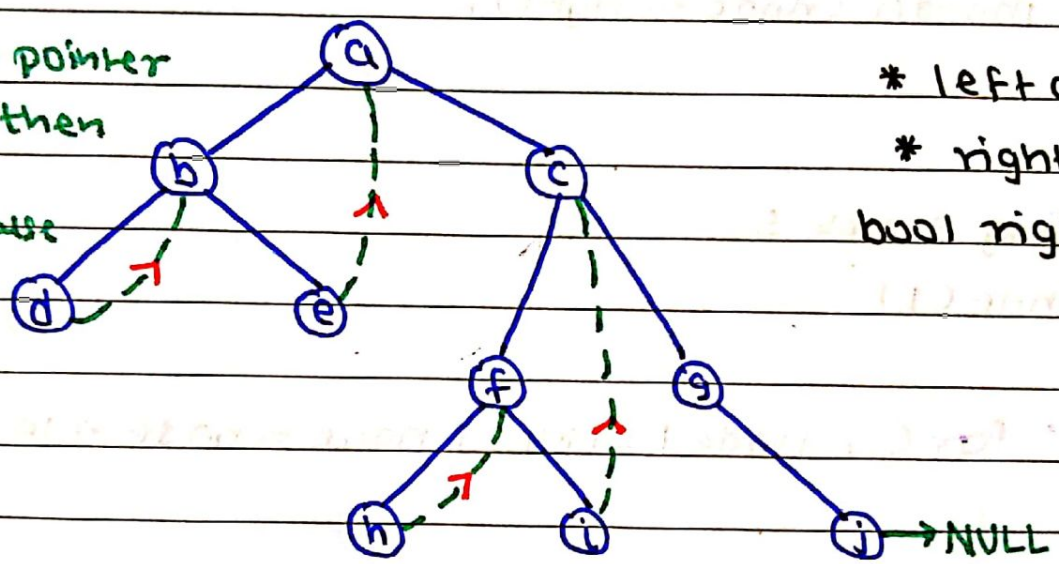
```
while (1)
{
    for ( ; node != NULL; node = node -> left)
    {
       push (s, node);
    }
  node = pop (s);
  if (node == NULL) break;
   func (node -> data);
   node = node -> right;
}
```

* Inorder Traversal using no stack:

Sys stack
user stack
} O(n)
no stack

↑ Constant used is also very less

* If right pointer is thread then bool true else bool false

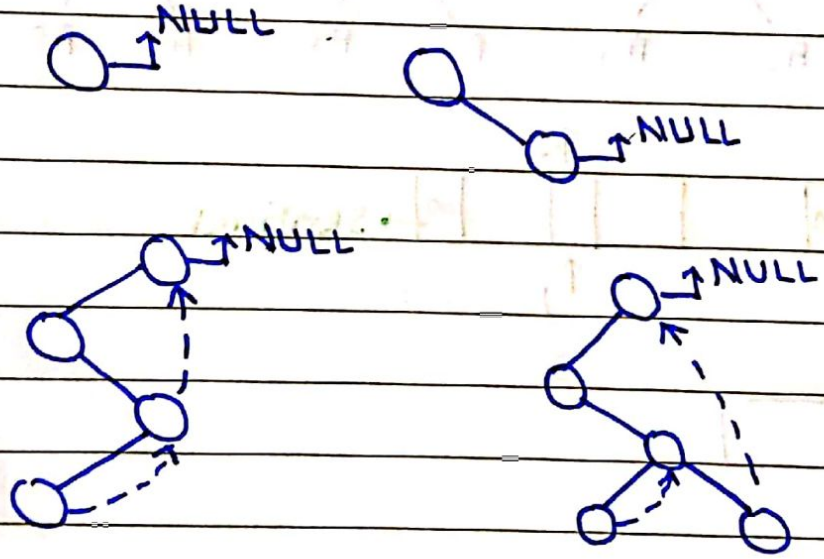


* left child
* rightchild
bool right thread

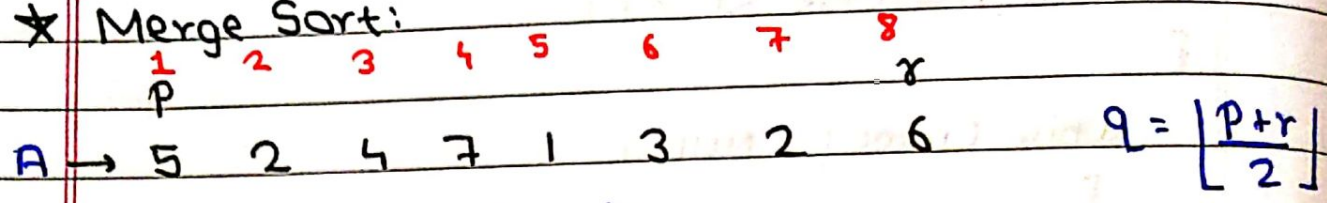
* Threaded Binary Tree

```
inorder successor (node)
{
    temp = node → rightchild;   ( If right child not present it acts as a right thread )
    if (node → rightthread == FALSE)
    {
        while (temp → leftchild != NULL)
            temp = temp → left child; }
    return temp;
}
```

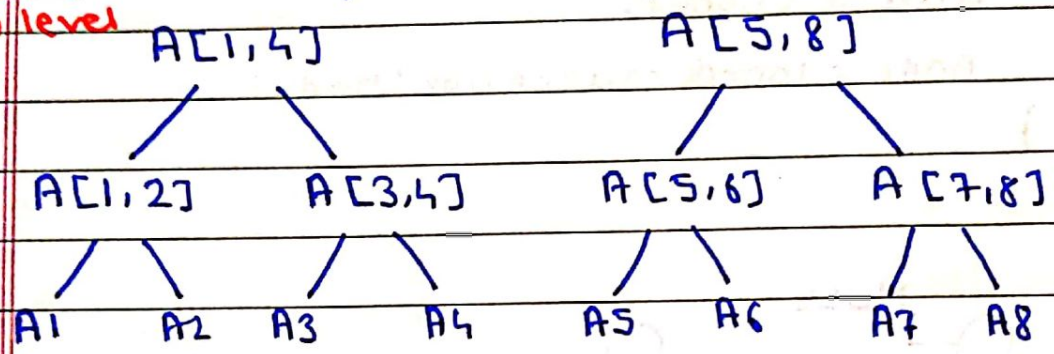
```
inorder traverse (node)        ↱ leftmost child
{
    while (node != NULL)
    {
        traverse (node);
        node = inordersuccessor(node);
    }
}
```

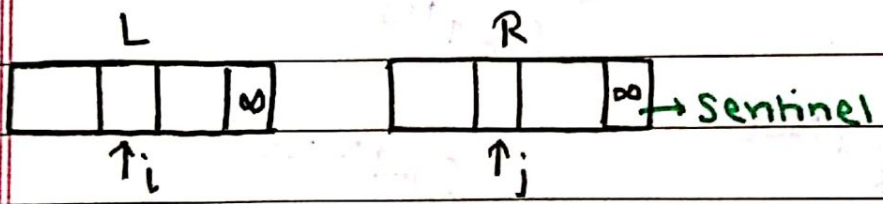


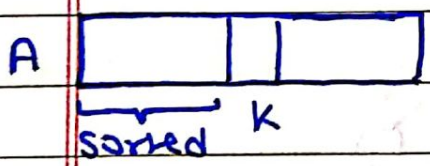
★ Merge Sort:



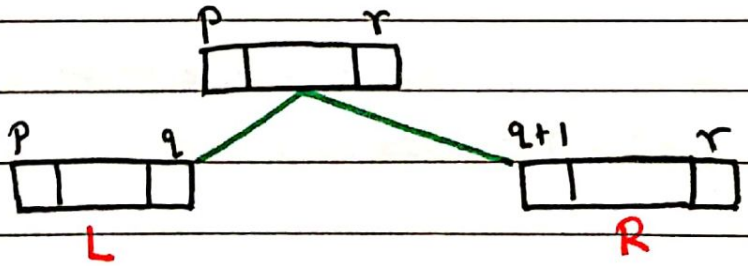
★ Work done at each level O(n)







L[1, i-1] ∪ R[1, j-1]



r - p + 1 = n

$$|L| = q-p+1 = \left\lfloor \frac{p+r}{2} - p + 1 \right\rfloor = \left\lfloor \frac{r-p+2}{2} \right\rfloor$$

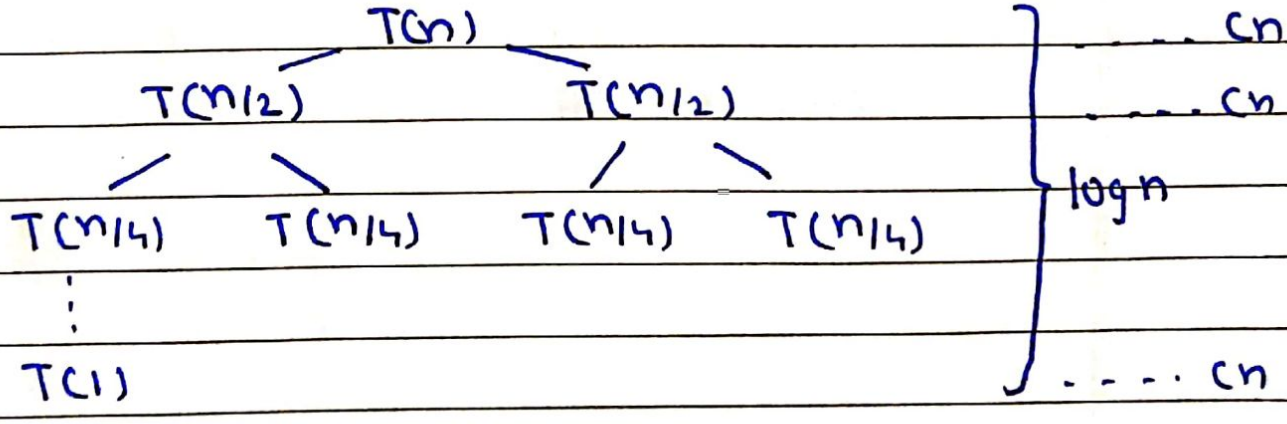
$$= \left\lfloor \frac{n}{2} + \frac{1}{2} \right\rfloor$$

$$= \left\lceil \frac{n}{2} \right\rceil$$

$$\therefore |R| = n - \left\lceil \frac{n}{2} \right\rceil = \left\lfloor \frac{n}{2} \right\rfloor$$

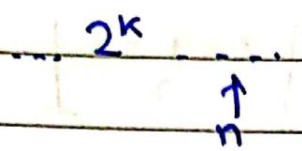
★ $T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(n)$ for $n > 1$

Let $n = 2^k$



Work done $cn(\log n + 1) \in \Theta(n \log n)$

★ $T(n) = \Theta(1)$ for $n = 1$



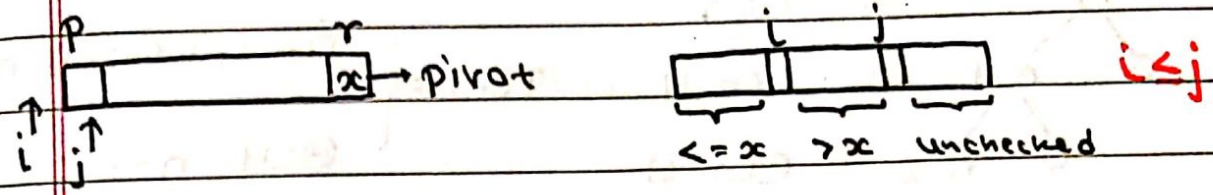
Work done $n = 2^k = c2^k(k+1)$

$n = 2^{k+1} = c2^{k+1}(k+2)$

Ratio $= 2\left(1 + \frac{1}{k+1}\right) \approx 2$ as $n \to \infty$

1) Time complexity $O(n \log n)$
2) Inplace X Workspace $\Theta(n)$
3) Stable
4) Adaptive X
5) If input in linked list then inplace.

★ Quick Sort:



p r
2 8 7 1 3 5 6 (4) → Pivot
i j

2 < 4 , 8 > 4
A[j] ↔ A[++i] j++
j++

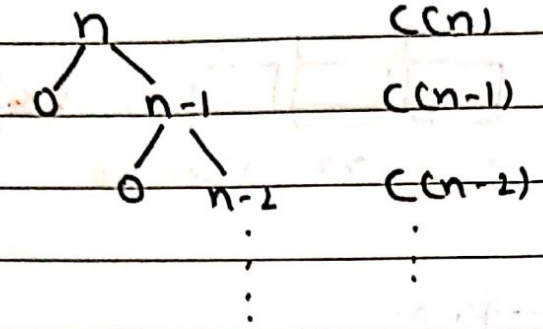
2 8 7 1 3 5 6 , 7 > 4 j++
i j

2 8 7 1 3 5 6 , 1 < 4
i j
A[j] ↔ A[++i]
j++

2 (8) 7 (1) 3 5 6
2 1 7 8 3 5 6
i j

★ On divison pivot position is fixed
Correctness can be proved by induction

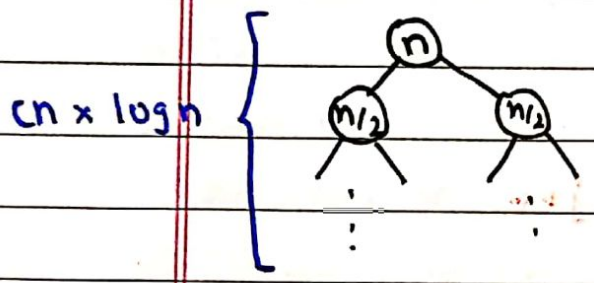
★ Worst case



$$T(n) = \begin{cases} T(n-1) + \Theta(n) & n>1 \\ \Theta(1) & n=1 \end{cases}$$

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

★ If tree is like

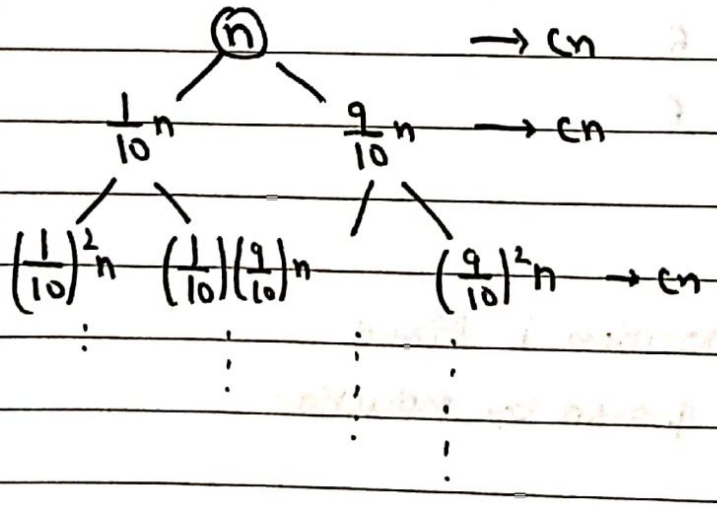


$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil - 1) + \Theta(n) \\ \Theta(1) \quad n=1 \end{cases}$$

$\therefore T(n) = cn(\log n + 1) = O(\log n \cdot n)$

$T(n) = O(n \log n)$

★ In free state



$1 = \left(\frac{9}{10}\right)^h n$

$\therefore \left(\frac{9}{10}\right)^h \leq \frac{1}{n}$

$\therefore h = \log_{10/9} n$

$\therefore T(n) \leq cn \log_{10/9} n$

$\leq O(n \log n)$

classmate
Date
Page

★ Calculating worst case time complexity

Let T(n) be worst case running time.

$$T(n) = \max_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$$

★ Substitution method

Let $T(n) \le cn^2$

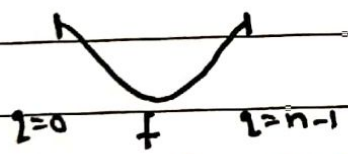
$$\therefore T(n) \le \left(\max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) \right) + \Theta(n)$$

$$f(q) = q^2 + (n-q-1)^2$$

$$\frac{df}{dq} = 2q + 2(n-q-1)(-1) = 4q - 2n + 2$$

$\therefore f' > 0 \quad q \in \left(\frac{n-1}{2}, n-1\right]$

$f' \le 0 \quad q \in \left[0, \frac{n-1}{2}\right]$



$\therefore$ maxima at $q = 0$ or $n-1$

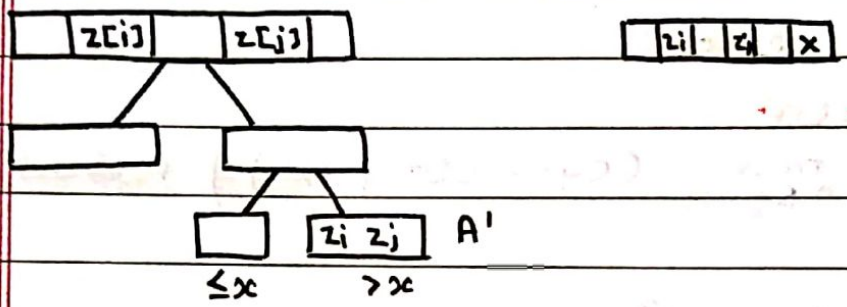
$\therefore T(n) = c(n-1)^2 + \Theta(n)$

$T(n) \le cn^2$

★ Randomised Quicksort

→ Randomly pick the pivot

→ Expected Time



Claim 1: All $[z_i \ldots z_j] \in A'$

Claim 2: z_i & z_j ⋯ 0

: 1 → when one of them is pivot

→ ∴ no. of comparisons of i^{th} & j^{th} element is 0 or 1

No of comp.

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij} \rightarrow i^{th} \text{ ele \& } j^{th} \text{ ele compared}$$

$$\therefore E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}]$$

$$E[X_{ij}] = Pr(X_{ij})$$

$$= Pr(z_i \text{ or } z_j \text{ is pivot in } [z_i \ldots z_j])$$

$$= \frac{2}{j-i+1}$$

$$\star\ E[X] = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} 2/k+1 \qquad (j-i=k)$$

$$< \sum_{i=1}^{n} \sum_{k=1}^{n} 2/k$$

$$< O(n \lg n)$$

$\therefore O(n+E[X])$

$O(n+n\lg n)$

$O(n\lg n)$

↳ very low const

1) Stable ✗

Hence not stable

2) Adaptive ✗

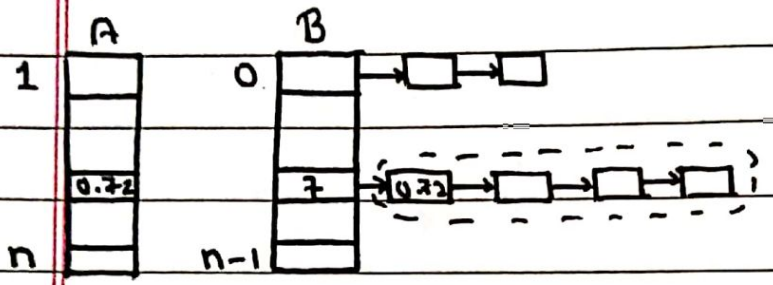
3) Inplace ✓

★ Bucket Sort:

→ All entries $\in [0, 1)$

→ A[i] is inserted to B[LnA[i]]

→ Elements of a specific bucket sorted using insertion sort.



★ Given an algo O(...) and we have an example of input for which algo takes O(...) then the algo is Θ(...)

1) Time complexity : $\Theta(n^2)$ → # insertion sort
2) Inplace ✗ (# Workspace : O(n))
3) Stable ✓
4) Adaptive
5) Online ✓

★ $T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i)^2$

↓ distribute elements

★ $E(T(n)) = \Theta(n) + \sum_{i=0}^{n-1} O(E(n_i^2))$

★ $E[n_i^2] = E[(\sum_{j=1}^{n} X_{ij})^2]$

X_{ij} : A[j] falls in bucket i

$X_{ij} \leq 0_1$

$$E[n_i^2] = \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j=1}^{n} \sum_{k=1}^{n} E[X_{ij} \cdot X_{ik}]$$

$$E[X_{ij}^2] = (1)^2 \frac{1}{n} + 0 = 1/n$$

$E[X_{ij} \cdot X_{ik}] = Pr(X_{ij} = r_1 \text{ \& } X_{ik} = r_2)$

$= Pr(X_{ij} = r_1) \times Pr(X_{ik} = r_2)$ (# independent)

$= E(X_{ij}) E(X_{ik})$

$= 1/n^2$

$\therefore E[n_i^2] = n(1/n) + n(n-1)/n^2$

$= 2 - 1/n$

$\therefore E(T(n)) = \Theta(n) + \sum_{i=0}^{n-1} O(2 - 1/n)$

$= \Theta(n)$

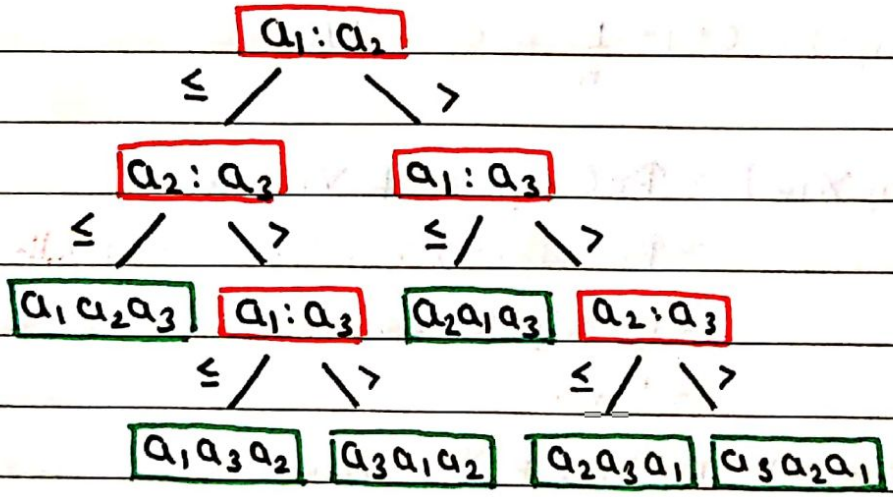
6) Avg case Time complexity : $\Theta(n)$

★ Comparison Sort

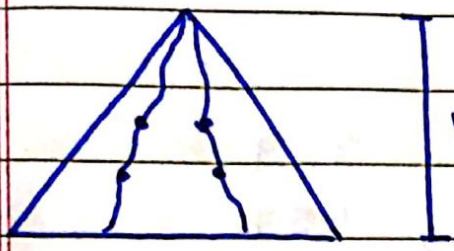
①	Bubble	$O(n^2)$	★ No. of comparisons
②	Insertion	$O(n^2)$	
③	Selection	$O(n^2)$	
④	Merge	$O(n \lg n)$	
⑤	Quick	$O(n^2)$	
⑥	Bucket	$O(n^2)$	

★ Decision Tree

→ Insertion Sort : a_1, a_2, a_3



classmate
Date
Page



$2^h \geq$ no. of leaves $\geq n!$

↓

full binary tree

$\therefore n! \leq 2^h$

$\Rightarrow n \lg n \leq h$

$\therefore h = \Omega(n \lg n)$

Counting Sort:

A: 2 5 0 3 2 3 0 3 ←

Size: A[max] - A[min] + 1 = K

Here C. Size = 6

C: 2 0 2 3 0 1 : Freq array C[i]: Freq(i)

0 1 2 3 4 5

C: 2 2 4 7 7 8 : Cumm Freq C[i]: Freq(≤i)

0 1 2 3 4 5

B: 0 3 C[3] --

1 2 3 4 5 6 7 8 C[0] --

B: used to make algo stable

1) Time complexity: $\Theta(n+k)$ *psuedopolynomial time algorithm

2) Inplace X (# B & C)

3) Stable

★ Radix Sort

329	457	329	329
457	657	839	457
657	329	457	657
839	839	657	839

★ max length of A[i] : k

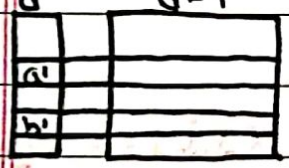
↕ Induction on no. of digits

Basis d=1

Induction Hypo d-1

Induction step

★ Sort each column using counting sort in O(n)



a' < b'

a' > b'

a' = b'

1) Time complexity O(n*k) = O(nk)

2) Stable ✓

3) Inplace X

★ To minimise Time complexity

→ given n b-bit no.s to reduce/optimise time we r bit no.s $1 \le r < b$

r bit no.s' range 0 to $2^r - 1$ ∴ counting sort $O(n+2^r)$

↳ no. of digits in rep. $= \lceil b/r \rceil$

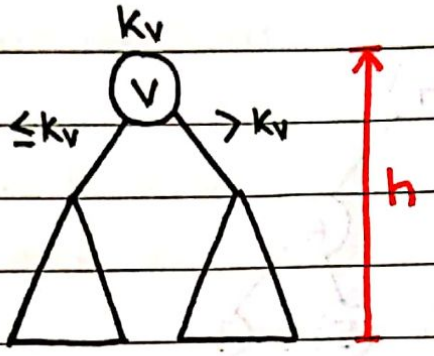
∴ $O\left(\lceil \frac{b}{r} \rceil (n+2^r)\right)$

↳ b bit no.

★ BST

V: Vertex

K_V: Key of V



★ ADT

Search (T, K)

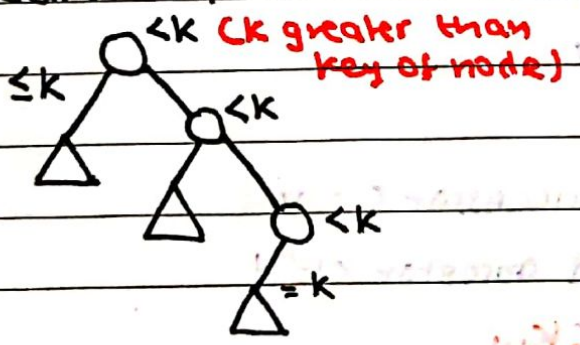
insert (T, V)

delete (T, V)

minimum (T)

maximum (T)

① Search (T, K)



At every iteration level ↑ by 1

Hence O(h)

Insert (T, V)

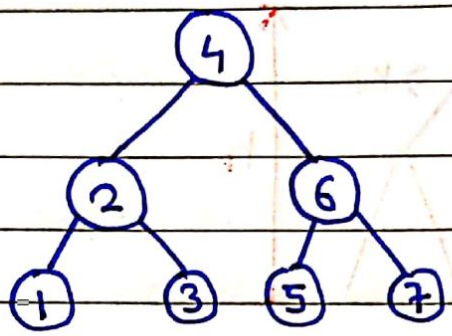
Explore the search path (K_V)

and insert appropriately

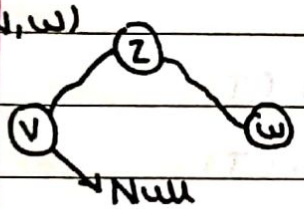
② Minimum (T) leftmost key O(h_left)

③ Maximum (T) rightmost key O(h_right)

④ Inordersuccessor (T, v) gives a sorted seq



LCA (v, w)



not a ancestor of v

$K_v < K_z < K_w$

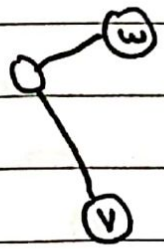
Left of w

ancestor of v

w inorder successor of v

w' left child ancestor of v

$K_v < K_{w'} < K_w$



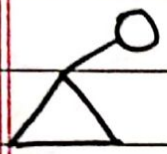
* Traverse parent → parent → ...
whenever node is left child at same node x

* Then x inordersuccessor of v

* Inorder successor O(h)

either go up or either go down

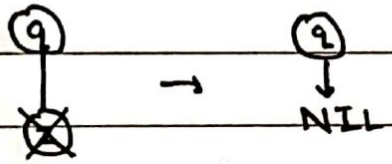
* Inorderpredecessor: O(h)



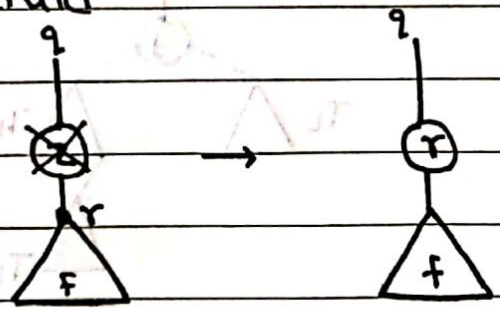
* go to left child
then to right most

(5) Delete (T, z)

(1) z : no children



(2) z : one child



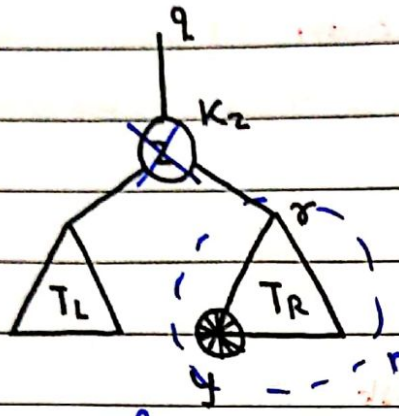
* If $z \in T_{left}(q)$

$r \leq z \leq q$

* If $z \in T_{right}(q)$

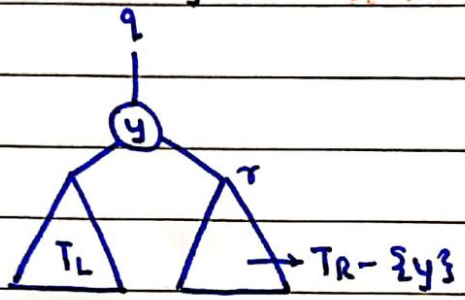
$q < z < r$

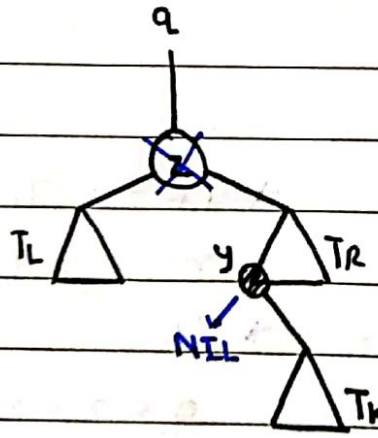
③ 2 children:

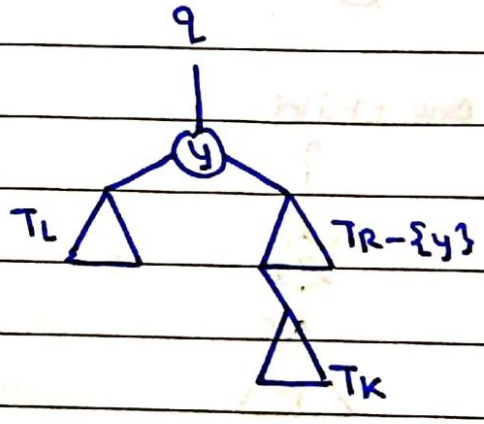


$T_R > K_2 > T_L$

$\therefore K_y > T_L$







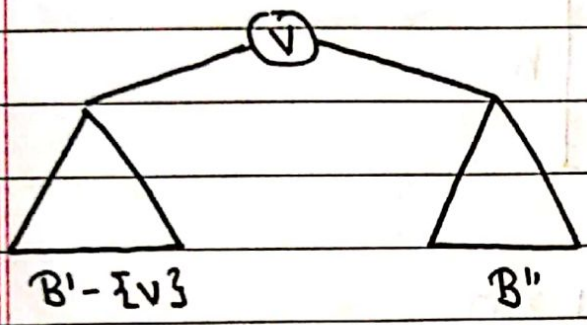
*Here to delete z we have Deleted y from the bottom like y with 1 child & inserted it at pos of z.

⑥ Join(B', B'')

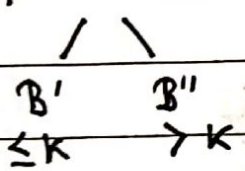
$\forall v' \in B' \& v'' \in B''$

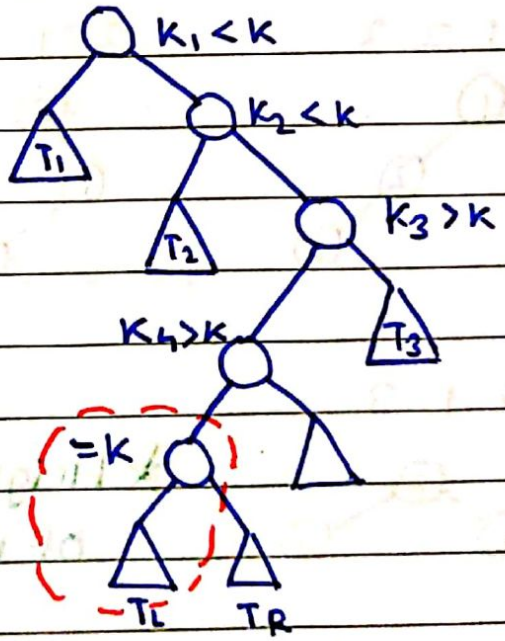
$K_{v'} < K_{v''}$ (Imp)

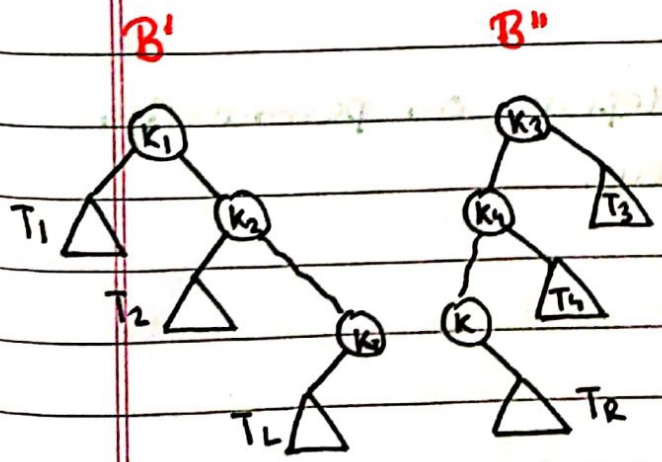
B' max $K_v : v$ (Delete)



⑦ Split(T, K)







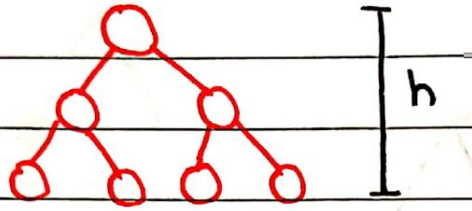
Split $O(h) * O(1) = O(h)$

↑ insert

★ Worst Case:

height O(n) # reverse or normal sorted

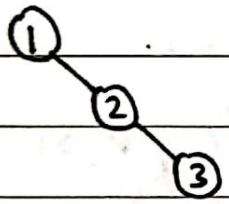
$h \geq \lfloor \lg n \rfloor$ (Best case)



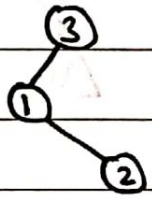
$h \geq \lfloor \lg n \rfloor$

$2^0 + 2^1 + \ldots + 2^h = 2^{h+1} - 1 \geq n$

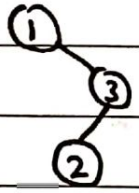
★ 1 2 3



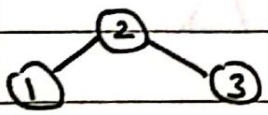
3 1 2



1 3 2



2 1 3

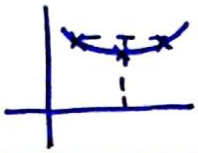


★ Height depends on permutation of n keys.

★ Jensen's Inequality:

$E(f(x)) \geq f(E(x))$

provided $f(x)$ is convex



classmate

Date

Page

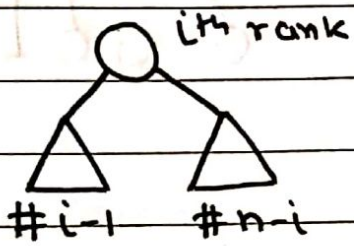
★ Randomly built BST: (RBST)

X_n: height when n keys are involved

Y_n: 2^{X_n}

$X_{n,i}$: root's rank is i

sort all keys root comes at i^{th} position



$X_{n,i} = 1 + \max(X_{n-i}, X_{i-1})$ (max height to be taken)

$Y_{n,i} = 2^{X_{n,i}} = 2 \cdot 2^{\max(X_{i-1}, X_{n-i})}$

$\therefore Y_{n,i} = 2 \max(Y_{i-1}, Y_{n-i})$

$$\Rightarrow E[Y_n] = \frac{2}{n} \sum_{i=1}^{n} E[\max(Y_{i-1}, Y_{n-i})]$$

$$\leq \frac{2}{n} \sum_{i=1}^{n} E[Y_{i-1} + Y_{n-i}]$$

$$\leq \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i]$$

★ Using substitution method we get

$E[Y_n] \leq 1/4 \binom{n+3}{3} = O(n^3)$

$\therefore E[2^{X_n}] \leq 1/4 \binom{n+3}{3}$

$\Rightarrow E[2^{X_n}] \leq cn^3$

$\therefore E[X_n] \leq k \lg n \quad \therefore E(X_n) = O(\lg n)$

★ Priority Queue

→ ADT:

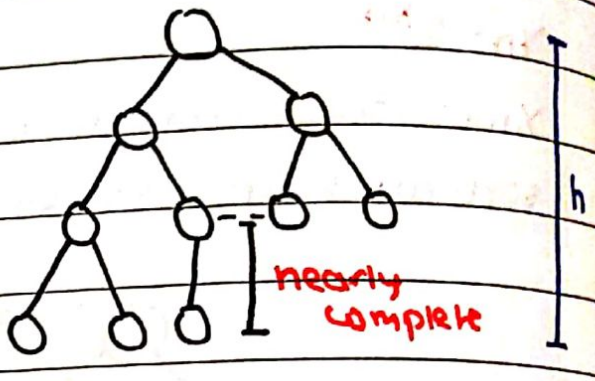
maximum

extractmax

increasekey

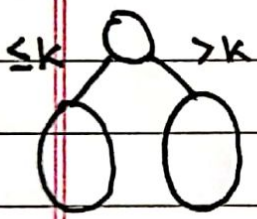
insert

delete

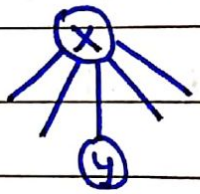


★ Binary Heap:

★ HOT property: Heapordered tree property



Binary Tree



y.key ≤ x.key

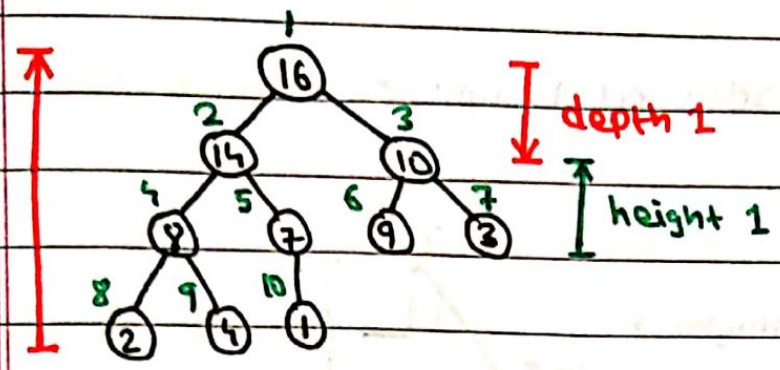
★ Heapordered binary tree property

"max binary heap"

→ binary tree

→ HOT property

→ nearly complete ≤ 1 (depth diff)



Index	1	2	3	4	5	6	7	8	9	10
	16	14	10	8	7	9	3	2	4	1

★ Parent: $\lfloor \frac{i}{2} \rfloor$

★ $2i$, $2i+1$: children

→ If it has n nodes the $h = \lfloor \log n \rfloor$

$2^0 + 2^1 + \ldots + 2^{h-1} < n \leq 2^0 + 2^1 + \ldots + 2^h$

$\therefore 2^h - 1 < n \leq 2^{h+1} - 1$

$\therefore 2^h \leq n \leq 2^{h+1}$

$\therefore h \leq \log n \leq h+1$

$\therefore h = \lfloor \log n \rfloor$

★ no. of leaves is $\lceil \frac{n}{2} \rceil$

$n = 2^h - 1 + x$

level at height $h-1$ → 1 parent 2 child

x is even ; No. of leaves $= x + (2^{h-1} - x/2)$

$= \left\lceil \frac{2^h + x}{2} - \frac{1}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil$

x is odd ; No of leaves $= x + (2^{h-1} - (x+1)/2)$

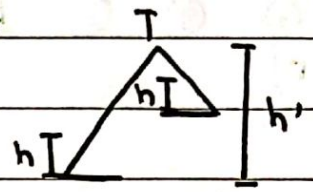
$= \lceil \frac{n}{2} \rceil$

★ At most $\lceil \frac{n}{2^{h+1}} \rceil$ nodes at height h

Basis $h=0 : \lceil \frac{n}{2} \rceil$

Hypo $\lceil \frac{n}{2^h} \rceil$

Step: No. of nodes in height h



no. of nodes at height h in T

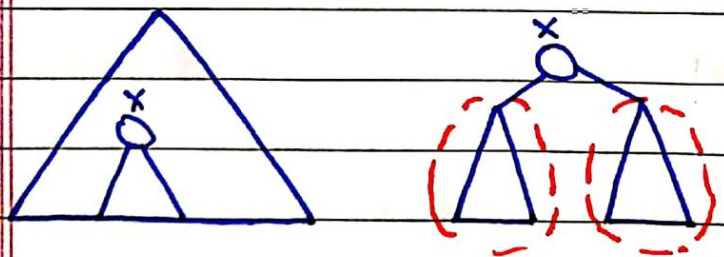
= no. of nodes at height h-1 in T'

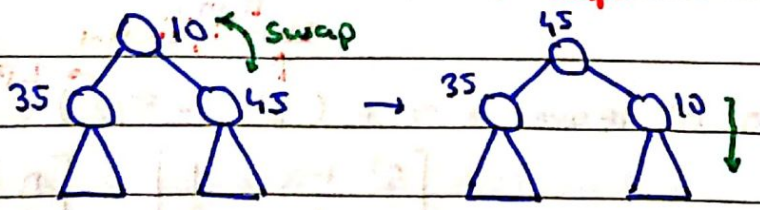
Nodes in T' is n'

T' = T - leaves

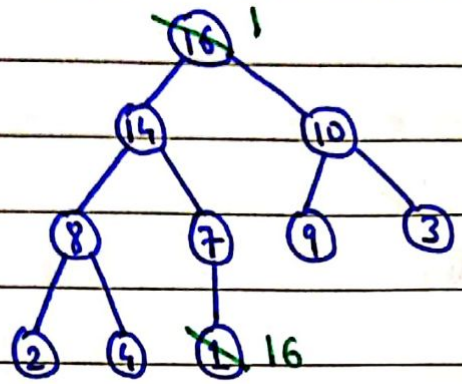
$= \lceil \frac{n'}{2^h} \rceil = \lceil \frac{n - \lceil \frac{n}{2} \rceil}{2^h} \rceil = \lceil \frac{\lfloor \frac{n}{2} \rfloor}{2^h} \rceil \leq \lceil \frac{n}{2^{h+1}} \rceil$

★ Maxheapify (x) ⓐ (hx)





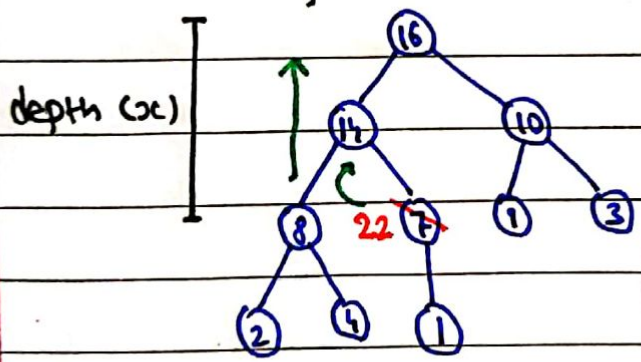
★ Extract max : O(h)



★ Swap root with last leaf

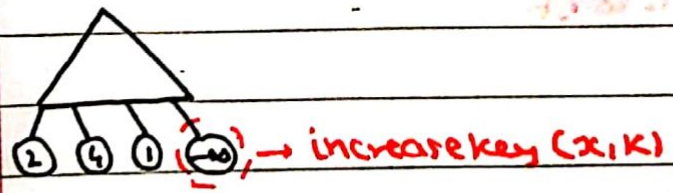
Then maxheapify (1)

★ Increase key : O(dx)



★ Here 7 is ↑ to 22

★ Insert (k) : O(h)



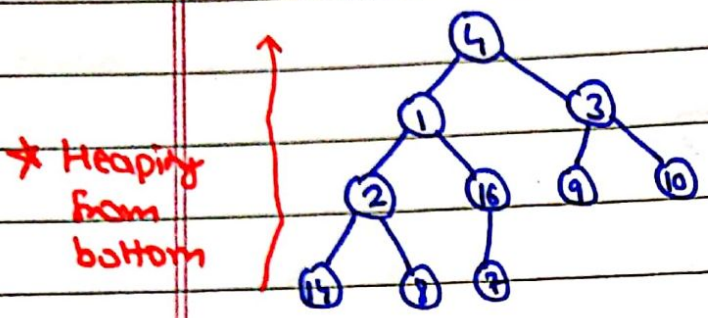
★ Delete : O(lgn)

Increasekey (x, ∞)

Extract max

Date
Page

★ Build a Heap



★ All leaves are max heaps

No leaves heapified

$n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$

$$O\left(\sum_{h=1}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil (h)\right)$$

$$\leq O\left(\sum_{h=0}^{\lg n} \left\lceil \frac{n}{2^{h+1}} \right\rceil h\right)$$

$$\leq O\left(\sum_{h=0}^{\lg n} \frac{n}{2^{h+1}} (h) + \sqcup\right)$$

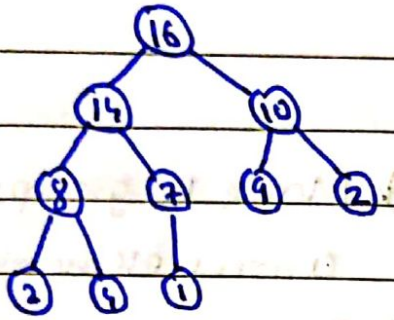
$$\leq O\left(n \sum_{h=0}^{\lg n} \frac{h}{2^{h+1}} + \sqcup\right)$$

↓

AGP which is const.

$\leq O(n)$

★ Heap Sort



★ O(n) to create a heap

★ extractmax & repeat for all

O(nlgn)

↳ hidden const very bad

1) Inplace ✓
2) Stable ✗
3) Parallelizable ✗

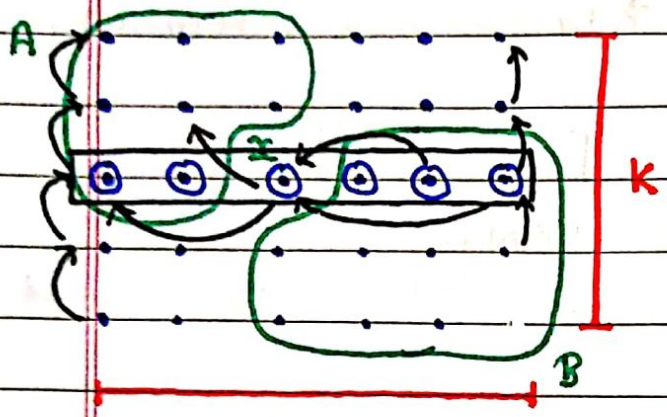
↓ Divide & conquer naturally of this type (split the work)

4) Adaptive

* Blum Selection Algo

→ To find i^{th} smallest element

→ Let k be small < 20



$\lceil \frac{n}{k} \rceil$

* Make k groups of array elements
* Sort each column
* Let x be the medians of the medians

* greater lower

* r: rank(x) found in O(n) # <x or >x

if r > i cut B & set i = i

r < i cut A & set i = i - r

* Note x > All ele of A

x < All ele of B

* WLG Let B be cut / removed

$\#B \geq \lceil \frac{k}{2} \rceil \left(\left\lceil \frac{1}{2} \left\lceil \frac{n}{k} \right\rceil \right\rceil - 2 \right)$

↳ # last col & col of x

$= \lceil \frac{k}{2} \rceil \frac{n}{2k} - 2 \lceil \frac{k}{2} \rceil$

classmate
Date
Page

* $T(n) \le T\left(n - \lceil \frac{k}{2} \rceil \frac{n}{2k} + 2\lceil \frac{k}{2} \rceil\right) + T\left(\lceil \frac{n}{k} \rceil\right) + O(n)$

remaining ele. — Median of medians — rank(x)

* Guess & Substitute

Let $T(n) \le cn$

$T(n) = O(1)$ for $n < R$

& $T(n) \le T\left(n - \lceil \frac{k}{2} \rceil \frac{n}{2k} + 2\lceil \frac{k}{2} \rceil\right) + T\left(\lceil \frac{n}{k} \rceil\right) + O(n)$

$T(n) \le cn - c\lceil \frac{k}{2} \rceil \frac{n}{2k} + 2c\lceil \frac{k}{2} \rceil + c\lceil \frac{n}{k} \rceil + an$

$= cn + \underbrace{\left(\frac{cn}{k} + c - c\lceil \frac{k}{2} \rceil \frac{n}{2k} + 2c\lceil \frac{k}{2} \rceil + an\right)}_{\le 0}$

then $T(n) \le cn$

* $\frac{cn}{5} + c - 3c\frac{n}{10} + 6c + an \le 0$ for $(k=5)$

$\Rightarrow 70c - cn + 10an \le 0$

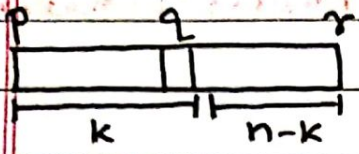
$\therefore c \ge \frac{10a}{1 - \frac{70}{n}}$

$\therefore$ for $n < 71$, $T(n) = O(1)$

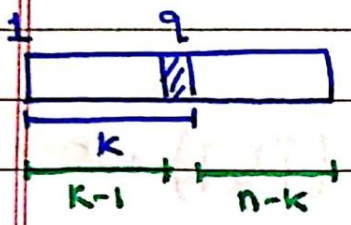
* Note if n is very small directly use insertion sort.

★ Randomised Selection (LAS VEGAS)

→ Randomly pick up the pivot



$q > i$ or $i > q$



X_k: subarray $[1, q]$ has exactly k elements

Then $X_k = 1$ else 0

$$\therefore T(n) \leq \sum_{k=1}^{n} X_k T(\max(k-1, n-k)) + O(n)$$

Recursion ↓ rank(q) ↓

$$\therefore E[T(n)] \leq \sum_{k=1}^{n} E[X_k T(\max(k-1, n-k))] + O(n)$$

$E[AB] = E[A]\,E[B]$ (if A and B independent)

$$= \sum_{k=1}^{n} E[X_k]\, E[T(\max(k-1, n-k))] + O(n)$$

$$= \frac{1}{n} \sum_{k=1}^{n} E[T(\max(n-k, k-1))] + O(n)$$

$$\leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[T(k)] + O(n)$$

★ $E(T(n)) \leq cn$ for $n \geq R$

$= O(1)$ for $n < R$

$$E(T(n)) \leq \frac{2}{n}\left[E(T(\tfrac{n}{2})) + E(T(\tfrac{n}{2}+1)) + \dots + E(T(n-1))\right] + O(n)$$

$$\leq \frac{2c}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} k + O(n)$$

$$= \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(\lfloor \frac{n}{2} \rfloor)(\lfloor \frac{n}{2} \rfloor - 1)}{2}\right) + an$$

$$= \frac{2c}{n}\left(\frac{n(n-1)}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an$$

$$\leq \frac{3cn}{4} + \frac{c}{2} + an$$

$$= cn - \underbrace{\left(\frac{cn}{4} - \frac{c}{2} - an\right)}_{\geq 0}$$

for $E(T(n)) \leq cn$

$\therefore \frac{cn}{4} - \frac{c}{2} - an \geq 0$

$\therefore c \geq \frac{4a}{n-2}$ & $n \geq \frac{2c}{c-4a}$

$\therefore R = 2c/c-4a$

Date
Page

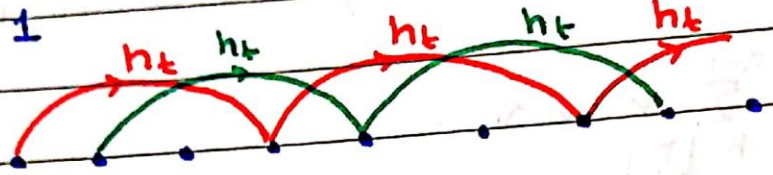
★ Shell Sort (Dummy increment Sort)

Increment seq.

$h_1 < h_2 \ldots < h_t = \lfloor \frac{n}{2} \rfloor$ By D Shell

$\|$

1



★ Sort all using h_t jumps (Insertion Sort)

$$T(n) \leq O\left(\sum_{i=1}^{t} \left(\frac{n}{h_i}\right)^2 h_i\right)$$

terms for ~~insertion sort~~ done h_i times

★ Note $h_1 = 1$ (# full proof)

$$T(n) \leq O\left(n^2 \sum_{i=1}^{t} \frac{1}{h_i}\right)$$

$$\leq O(n^2) \quad \left(\because \sum_{i=1}^{t} 1/h_i \text{ is const}\right)$$

★ large $n/2$ ele at even pos

• 0 • 0 • 0 • 0 • 0 •

$\therefore$ i^{th} smallest ele at $2i-1$

$\therefore$ we move $[(2i-1)-i] = i-1$ for each i

$\therefore \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} (i-1) \in O(n^2)$

$\therefore \Theta(n^2)$

① Stable X

② When $n < 20$: Insertion sort

$20 < n < 5000$: Shell Sort

★ Best possible $O(n^{4/3})$

T(n) Changes with choice of <h>

<h>	T(n)	
$\lfloor n/2^k \rfloor$	$O(n^2)$	
$2^k \pm 1$	$O(n^{3/2})$	
$2^p 3^q$	$O(n \log^2 n)$	1, 2, 3, 4, 6, 8, 9 ...

→ h_k, h_{k+1}, h_{k+2}

Note all linear combinations of h_{k+1}, h_{k+2} are already sorted

★ Hence to make shell sort efficient

the consecutive ele of <h> should be co-prime

eg: 1, 3, 7, 15, ... $(2^k \pm 1)$