$H = \{h \mid h : u \to \{0, 1, \ldots, m-1\}\}$

Bad

H

$h \in H(bad) \longrightarrow h(x) = h(y) \quad \forall\, x \neq y.$

gives collision from store.

$$Bad = \left\{ h \in H \;\middle|\; h(x) = h(y) \quad \forall\, x \neq y \right.$$

$\therefore |Bad| = \frac{|H|}{M}$ cardinality of |Bad| (no. of elements.)

Universal Collection! → hash function from the collection of hash function

why this is good?

Theorem: Let h be a hash chosen (uniformly) at random from the universal set of H. Suppose h is used to hash n keys into m slots. Then given a key x,

$$E(\#\text{ of collision w/u } x) < \frac{n}{m}$$

Proof: Indicator random variable?

$$C_{xy} = \begin{cases} 1 & \text{if } h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

$P(C_{xy} = 1) = \frac{1}{m}$ (Fixing x & P.t colliding)
(hash fun from univ hashset

Collision happens? → when we choose our hashfunction from that portion

$$E(C_x) = E\left(\sum_{y \in T - \{x\}} C_{xy}\right)$$

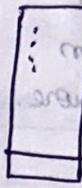
$$= \sum_{y \in T - \{x\}} E(C_{xy}) = \frac{n-1}{m} < \frac{n}{m}$$ (which is $\frac{n}{m}$) (So good)

Universal choice is good! Since we have a collection of bad hash portion: $\frac{1}{m}$ fraction of the total. (Randomly: Using bad hash will be less).

Perfect hash functions

Let $m \rightarrow$ prime number.

$K = (K_0, K_1, \ldots, K_r)$ (decompose this key (in to r bits) +1)

0  m-1

$0 \leq K_i < m$

K_0	K_1	K_2 ..	K_r

'decomposing keys'

Random Strategy

Constant $a = (a_0, \ldots, a_r)$ where

$0 \leq a_i < m$ (chosen randomly)

$$h_a(K) = \left(\sum_{i=0}^{r} a_i K_i\right) \bmod m$$

↳ Inner produce

'hash function: depending on choice of a' (Run time)

$$H = \left\{ h_a \;\middle|\; \begin{array}{l} a = (a_0, \ldots a_r) \\ a_i \in \{0, \ldots m-1\} \text{ (random)} \end{array} \right\}$$

↳ Hash function collection. a_i chosen randomly

↳ use that random $h(a)$ by using $\left(\sum_{i=0}^{r} a_i K_i\right) \bmod m$

Prove that: Size of the bad portion

$|H|$ = Cardinality of universal hash function.

a_0	...	a_r

$\sum$ m → elements

Permutation = m^r chances

$$\frac{|H|}{m} = \frac{\boxed{a_1 \cdots a_{r-1} \mid a_r}}{m \times m \times \cdots m}$$

Cardinality of bad portion $|H|$

$x = (x_0, \ldots x_r)$

$y = (y_0, \ldots y_r)$

$$BAD = \{ h_a \in H \mid h_a(x) = h_a(y) \}$$ ↳ 'Bad'

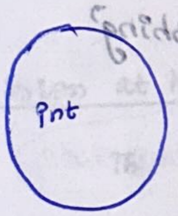
$x_0 \neq y_0$

$h_a(x) = h_a(y)$

$$\left(\sum_{i=0}^{r} a_i x_i \right) \bmod m = \left(\sum_{i=0}^{r} a_i y_i \right) \bmod m$$

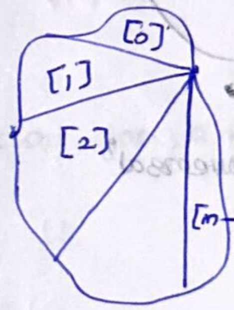
If thes two exp. equal under modulo
↓
These two are under congruence relation

'Congruence' '≡' relation
(equivalent relation)
↓



→ remainder 0 to $m-1$

div by m



'equal under modulo' – our case!

$$(a_0 x_0 - a_0 y_0) = \sum_{i=1}^{r} a_i (y_i - x_i) \bmod m$$

$$a_0 (x_0 - y_0) = \sum_{i=1}^{r} a_i (y_i - x_i) \bmod m$$

we say two integers a, b – in same class

So, $a \equiv b \pmod m$

$a \bmod m = b \bmod m$

If $x_0 \neq y_0$, then they have inverse under mod m.

$$a_0 \equiv \left(\sum_{i=1}^{r} a_i (y_i - x_i) \right) \cdot (x_0 - y_0)^{-1} \pmod m$$

a_0	...	a_r

$m \times m \times \cdots m$

m^r = Bad

$m^r \longrightarrow |Bad|$ = cardinality of bad set

$$|Bad| = m^r = \frac{m^{r+1}}{m} = \frac{|H|}{m}$$

Universal Collection

$|H|$

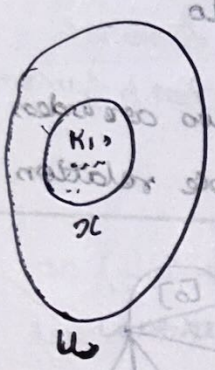
Perfect hashing → Static arrangements

Given n keys, construct static hash table, space $\propto n$ → (No Insertion/deletion)

Constant time search

Solution

Idea: 2 level scheme with universal hashing at each level.



'universal'

→ Say (2 keys Collisions 14, 27)

↓

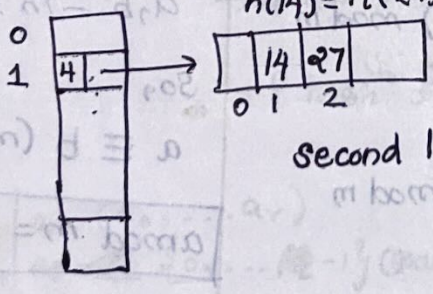
'Second level hashing'

4 slots for 2 keys

9 slots for 3 keys

n_i^2 slots for n_i keys

$h(14) = h(27) = 1$ (1st level)

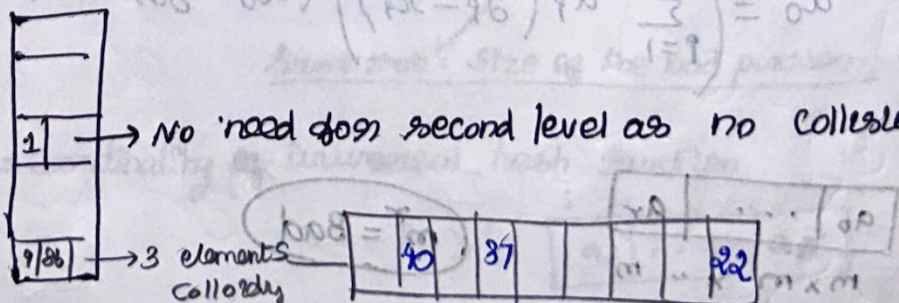


Second level hashing — n_i^2 slots.

Choose A

$h_{31}(14) = 1$ (index of sub array)

$h_{31}(27) = 2$ (index of sub array)



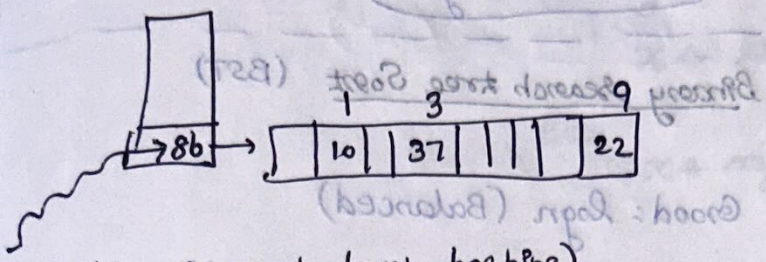
No need for second level as no collision.

$h(40) = h(37) = h(22) = ?$

$h_{86}(40) = 1$ — '2 level hashing'

'Second level hashing' – No collision (since n^2 slots)

Search (37)



1st hashing (second level hashing)

$\therefore A(g_n) : 86$

$\therefore h_{86}(37) = 3$ (index) (Two hash functions)

Analysis → why there is no collision
→ storage!

Collision at level 2

Theorem: Let H be a class of univ hash function for a table size $m = n^2$, n Keys, → table size n_i^2

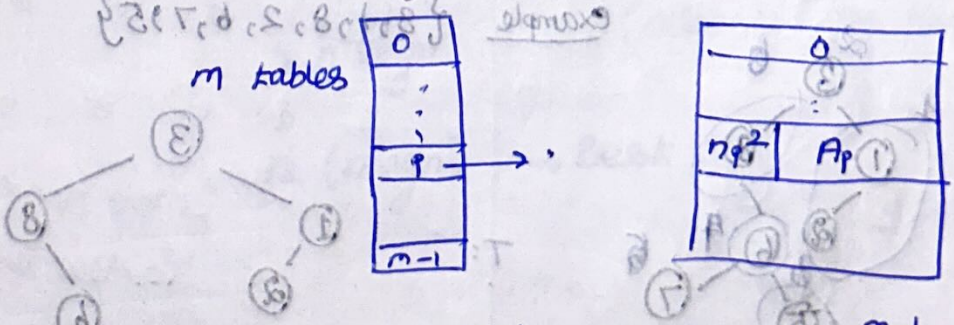
the expected no. of collision is atmost $\frac{1}{2}$ (even 1 collision)

(x, y) keys $\longrightarrow \frac{1}{m} = \frac{1}{n^2}$

$$E(\#\text{collisions}) = \frac{{}^{n}C_2}{n^2} = \frac{n(n-1)}{2} \cdot \frac{1}{n^2} < \frac{1}{2} \text{ (Atmost half)}$$

'we can't expect even 1 collision' → Good hash function.

Storage analysis

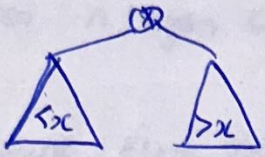


$$\text{Total Storage} = m + \sum_{i=0}^{m-1} n_i^2 \Rightarrow m + \sum_{i=0}^{m-1} E(n_i^2)$$

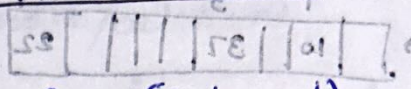
Expected storage $= n + \sum_{i=0}^{n-1} \left(E(n, i) \right)$

$= \theta(n) + \theta(n)$

$= \theta(n)$ "Total Storage"

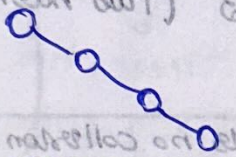


Binary search tree Sort (BST)



* Good: $\log n$ (Balanced)

Bad tree



$\Theta(n)$

Use Binary tree - to have a Sorting algo.

BST - Sort $(A[1 \ldots n])$ — Root → leaf node

1) $T \leftarrow$ empty.

2) For $i \leftarrow 1$ to n

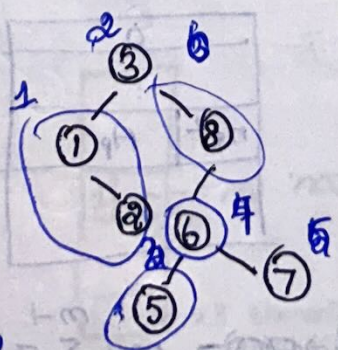
3) BST - Insert $(T, A[i])$

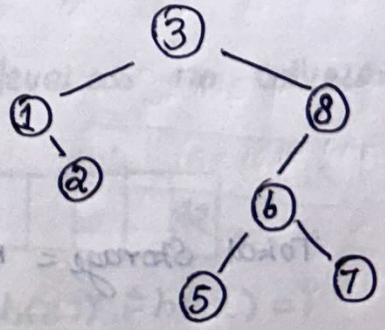
4) Inorder - Tree - Walk (T)

"Tree traversal" → Left Subtree → Root → Right Subtree.

In, pre, post order

Example $\{3, 1, 8, 2, 6, 7, 5\}$



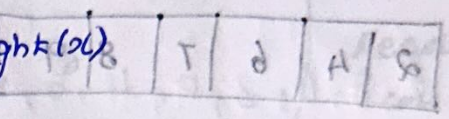


Inorder

1 2 3 5 6 7 8

BST Insert (T, A[i])

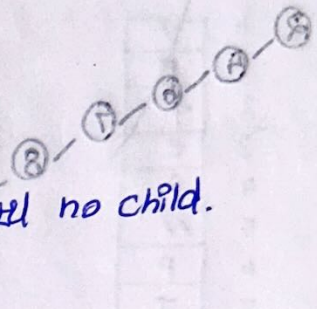
1) $x \leftarrow root(T)$
2) while ($x \neq$ Null)
   if Key $(x) < A[i]$
   $x \leftarrow left(x)$
   else
   $x \leftarrow right(x)$



In order tree:

1) $x \leftarrow$ root
2) print (left (x))
3) Print (Root)
4) print (Right(x))

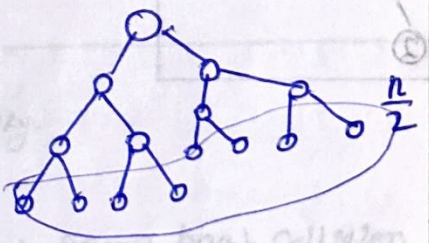
Until no child.



$\Theta(n) \rightarrow$ (Seeing node only once)
n: nodes: $\Theta(n)$.

Build a tree: (Time) $\Omega(n \log n)$

$Time = \sum depth(x)$
$x \leftarrow T$



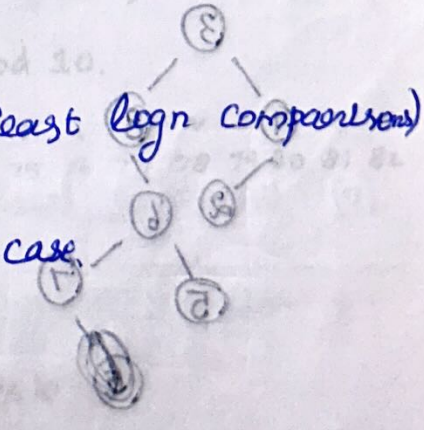
Atleast each of these node, we need to compare $\frac{n}{2}$ times.

depth = $\log n$

$Time = \sum depth(x)$
$x \leftarrow T$

$> n \log n$

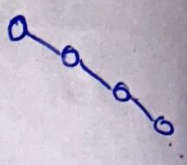
$\Omega(n \log n)$ → Best case



(Atleast $\log n$ comparisons)

Best case: $\Omega(n \log n)$

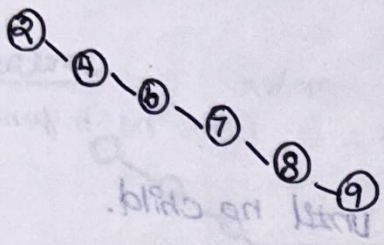
Worst case: $O(n^2)$ — bad tree



Time = $\sum_{x \in T}$ depth(x) = $1+2+\cdots+n$

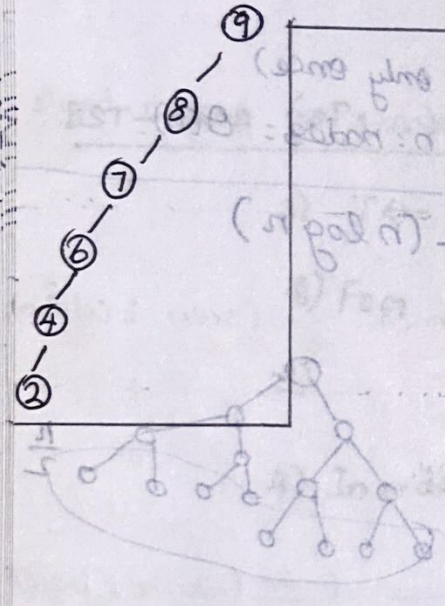
$= \frac{n(n+1)}{2} = O(n^2)$

A: | 2 | 4 | 6 | 7 | 8 | 9 | → Already Sorted



→ Everybody need to compare with all prev. element

(or) Reverse Sorted



worst case → $O(n^2)$

Best case → $\Omega(n \log n)$

Comparison b

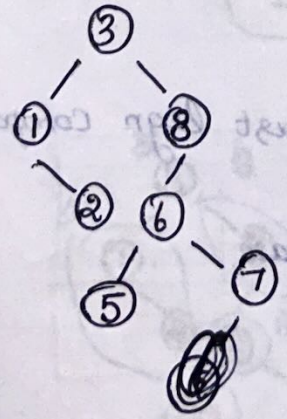
what's the relation

Quicksort

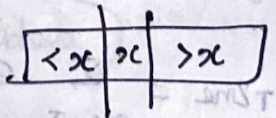
Worst Case → $O(n^2)$

Best → $\Omega(n \log n)$

A = 3, 1, 8, 2, 6, 7, 5



Quick Sort ← pivot



3 1 8 2 6 7 5

1 2 3 8 6 7 5

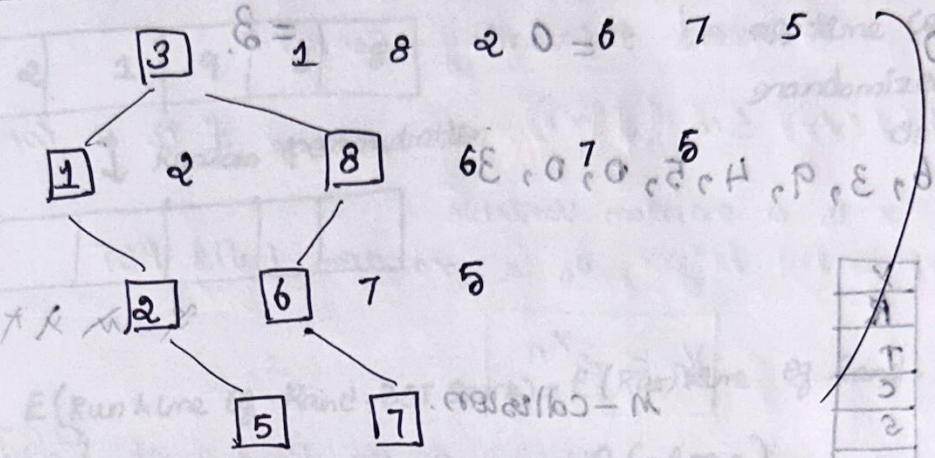
'Stable partition'

(order won't change)

① ② ③ 5 ⑥ 8 7 8

① ② ③ ⑤ ⑥ 7 ⑧ → 'No of comparisons' same as BST

① ② ③ ⑤ ⑥ ⑦ ⑧



'Same tree'

In partition – no. of comparisons same as BST

'quicksort' – 6 will compared with 8 & 3 same as BST

BST sort – performs same no. of comparisons as quicksort!

↓

· Same time complexity.

---

1) * 4679, 1989, 4199 } x mod 10 → same hash collision (In groups)
   * 1471, 6171

2) Hash table – maps keys to values (structure)

3) h(x) = (ord(x) – ord(A) + 1) mod 10.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82

S	T	U	V	W	X	Y	Z
83	84	85	86	87	88	89	90

K → 10% 10 = 0 | R → 18% 10 → 8 | P → 16 % 10 → 6

C
68-65
= 3

S
84-65
= 9
J
75-65

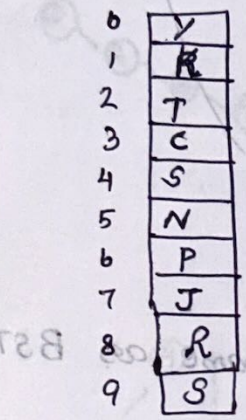
N
79-65
= 4
M
78-65

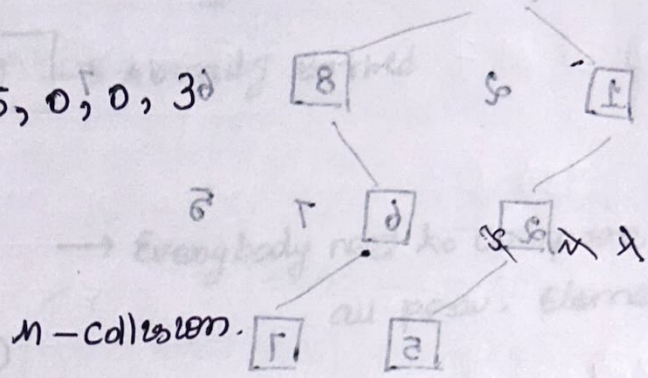
Y
89-65
= 5

T
85-65

= 0 = 0 = 2

1, 8, 6, 3, 9, 4, 5, 0, 0, 3





---

100 slots, 3 slots occupied after the 3 insertions?

4) $\frac{97}{100} \times \frac{96}{99} \times \frac{95}{98} = \frac{(97)^3}{(100)^3}$

5) Division method $h(k) = k \bmod m$

6) Distribute among uniformly — $9^3 \bmod 10$ (more space)

7) open addressing: $\frac{1}{1-\alpha}$ → True

8) Collision → Competing for same bucket

9) No. of ways 1, 2, 3, 4, 5, 6, 7 in a BST

$2^{7-1} = 2^3 \times 2^3 = 64$

10) universal hash → atmost $\frac{1}{2}$ collisions.

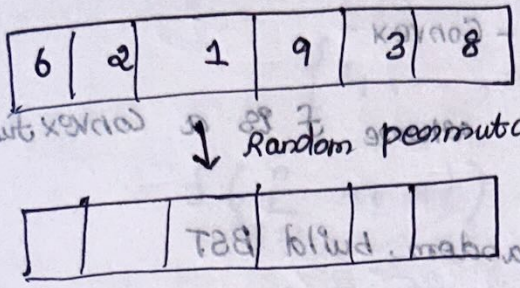
Randomly build BST

Rand-BST_Sort (A[1,...n])

1) Random permutation (A)

2) BST_Sort (A)

↳ Time complexity same as time complexity of randomized quick sort.



↓ Random permutation

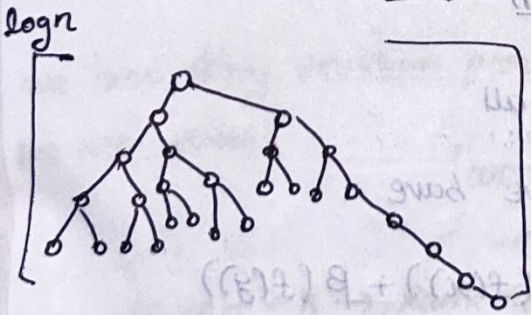
E(Runtime of Rand. BST Sort) = E(Runtime of Rand. quickSort)

$= \Theta(n \log n)$

$$E\left(\sum_{x \leftarrow T} \text{depth}(x)\right) = \Theta(n \log n)$$

$$E\left(\frac{1}{n} \sum \text{depth}(x)\right) = \Theta(\log n)$$ ∴ proved.

Avg. depth of Randomly built BST = $O(\log n)$



logn → Balanced

only branch-more

Avg. depth $\leq \frac{1}{n} \times \left((n \log n) + \sqrt{n} \cdot \frac{\sqrt{n}}{2}\right)$

$\sqrt{n}$ elements with height $\frac{\sqrt{n}}{2}$

$\leq O(\log n)$

'Height may not be logn — even if Avg. depth is $O(\log n)$'

Prove E(height) = log n

Height of a randomly build BST

Outline of the Analysis:

1) Jensens inequality - Convex

$f(E(X)) \leq E(f(X))$ where f is a convex function

X is a random variable

2) Exponential height of a random build BST

$$Y_n = 2^{X_n}$$

$X_n \rightarrow$ Height of the Rand. built BST

3) Prove that $2^{E(X_n)} \longrightarrow$ Convex function $\leq E(2^{X_n}) =$

$E(Y_n) = O(n^3)$

$\Rightarrow E(X_n) = O(\log n)$

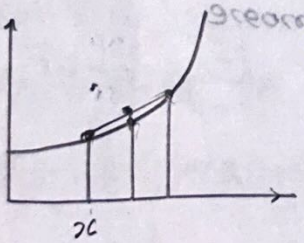
Convex function

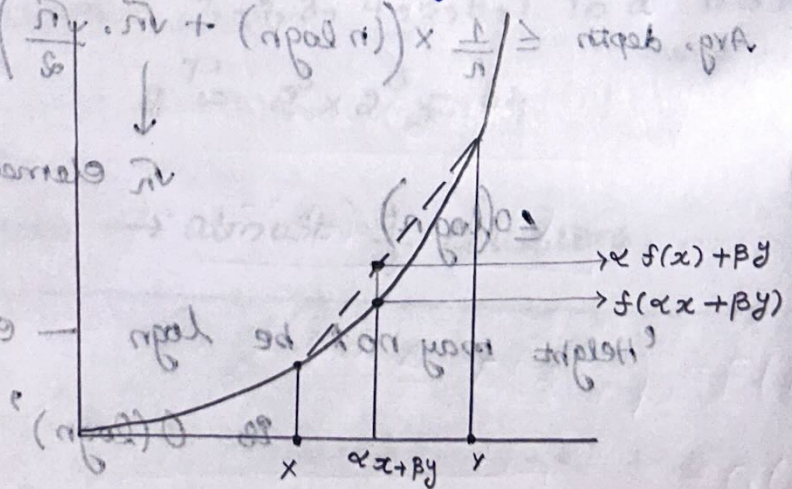
$f : R \rightarrow R$ is Convex if for all

$\alpha, \beta \geq 0$, $\alpha + \beta = 1$, we have

$$f(\alpha x + \beta y) \leq \alpha(f(x)) + \beta(f(y))$$

$\forall\ x, y \in R$





Jensens equality telling,

$$f(E(x)) \leq E(f(x))$$

Lemma

Let $f: R \to R$ be a convex function

$$\{\alpha_1, \alpha_2, \ldots \alpha_n\} \qquad \sum_{q=1}^{n} \alpha_q = 1.$$

$$\{x_1, \ldots, x_n\}$$

$$f\left(\sum_{q=1}^{n} \alpha_q x_q\right) \leq \sum_{q=1}^{n} \alpha_q f(x_q)$$

Expectation of $f(x)$

$$f(E(x)) \leq E(f(x))$$ Proved.

Height analysis : BST (Height – Randomly build)

* X_n = height of a Randomly Chosen BST with n elements.

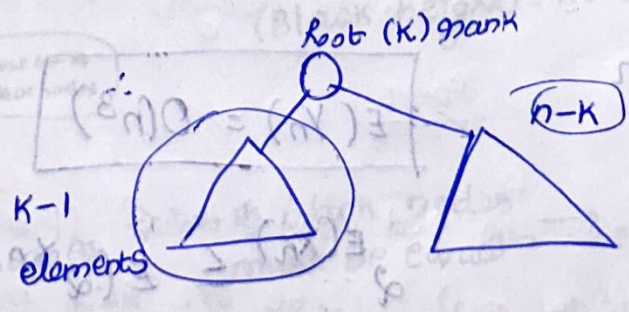
$Y_n = 2^{X_n}$ Show: $E(Y_n) = O(n^3)$

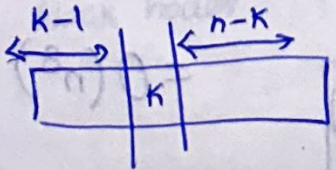
we are doing random permutation: we don't know – which will be the root!

'we don't know rank of the root'

∴ Let K → Rank of the root (we don't know root & also its rank)

$$X_n = 1 + \max\{X_{K-1}, X_{n-K}\}$$





$$Y_n = 2^{\max\{Y_{k-1},\, Y_{n-k}\}}$$

$$Z_{nk} = \begin{cases} 1 & \text{if rank of root} = k \\ 0 & \text{otherwise} \end{cases}$$

$$P(Z_{nk} = 1) = \frac{1}{n} \text{ (equally likely)}$$

$$E(Z_{nk}) = \frac{1}{n}$$

$$Y_n = \sum Z_{nk} \left(2^{\max\{Y_{k-1},\, Y_{n-k}\}}\right)$$

$$E(Y_n) = \sum E(Z_{nk})\, 2^{\max\{Y_{k-1},\, Y_{n-k}\}}$$

$$= 2 \sum_{k=1}^{n} E(Z_{nk})\, E\left(\max\{Y_{k-1},\, Y_{n-k}\}\right)$$

$$E(Y_n) = \frac{2}{n} \sum_{k=1}^{n} E\left(Y_{k-1} + Y_{n-k}\right)$$

$$= \frac{4}{n} \left(\sum_{k=0}^{n-1} E(Y_k)\right)$$

Proof $E(Y_n) = O(n^3)$

$$\leq \frac{4}{n} \sum_{k=0}^{n-1} c k^3$$

$$\leq \frac{4c}{n} \sum_{k=0}^{n-1} k^3$$

$$\leq \frac{4c}{n} \int_0^n x^3 \, dx$$

$$\leq \frac{4c}{n} \left(\frac{x^4}{4}\right)_0^n$$

$$\leq \frac{4c}{n} \left(\frac{n^4}{4}\right)$$

$$\leq cn^3$$

$$E(Y_n) \leq cn^3$$

$$E(Y_k) \leq ck^3$$

$\forall\, k < n$

Induction hypothesis

$$E(Y_n) = O(n^3)$$

$$2^{E(X_n)} \leq E(2^{X_n}) = E(Y_n) \leq cn^3 = O(n^3)$$

$$\Rightarrow E(X_n) = \log O(n^3)$$

$E(x_n) = 3\ O(\log_2 n) = O(\log_2 n)$

Red black tree

Before: we saw Avg case: Balanced : No guarantee

worst case: It may be bad.

Balanced binary tree — Red black tree.

* Not randomly build
* Guaranteed.

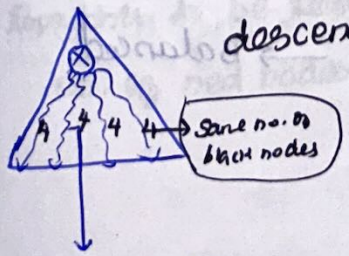
(query: In log time: In balanced time).

Guaranteed BST

* AVL tree
* B-tree
* 2-3 tree
* 2-3-4 tree
* Red black tree

Red black tree

1) It is a binary search tree
2) Each node is coloured — either red/black
3) Roots are black.
4) Nil (Leaves) are blacks
5) If a node is red - then the parent has to be black.
6) All simple path from any node x to DL descendant leaf - have same no. of black nodes.
(Black height) = no. of " "



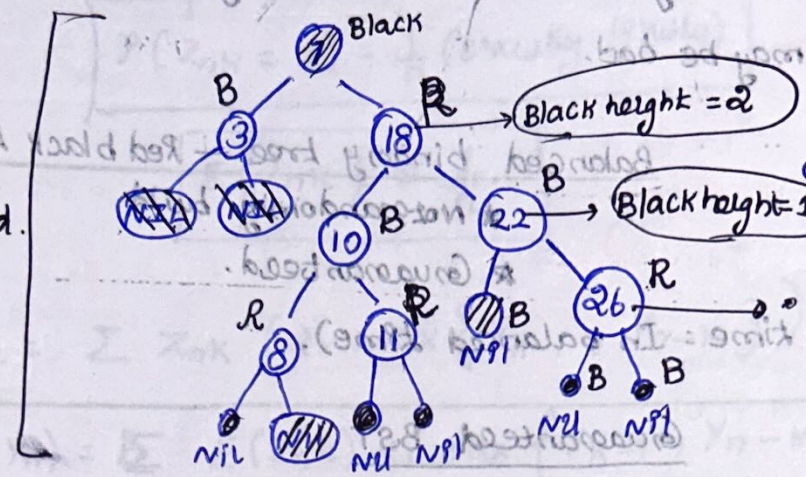
Each path crosses equal no of black nodes ⇒ no. of black nodes must be equal → each path has same no. of black node. 'Black height'

★ If Property 6 – Satisfied – we call the tree – Red black tree

'Because of this property' – Guarantees a balanced tree

→ 'No random permutations – Guaranteed'

eg:



'New Nil leaves' at all leaves' – descendants

Nil – black

Root – black.

BH (18) = 2

BH (22) = 1

BH (10) = 1

BH (7) = 2.

Theorem

Height of a Red-black tree

A red black tree with n nodes n keys has height h which is less than $2\log(n+1)$

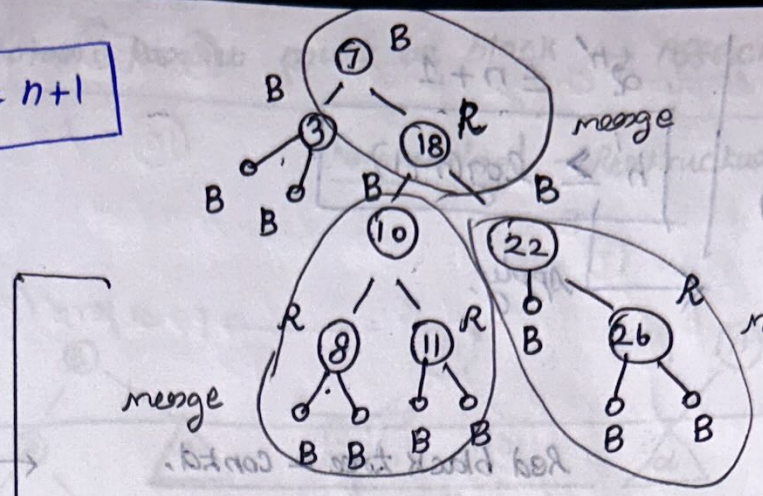
$$h \leq 2\log(n+1)$$

↳ $h = O(\log n)$ → Balanced

Prove

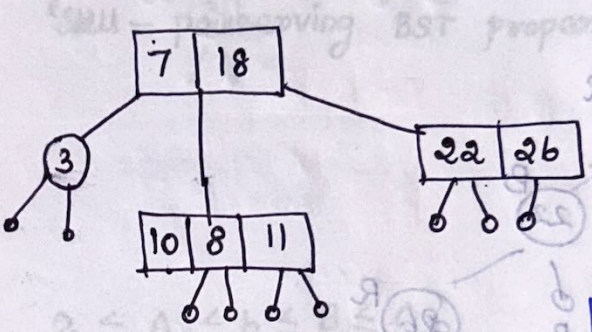
Intuition: Remove the red node: How? (Parent is black when the node is red): Merge the red node to its parent!

leaves = n+1



merge red node with black parent!

converted 2-3-4 tree.



height = h'

leaves = n+1

Leaves = either 2/3/4

2-3-4 tree (Balanced)

$h' \geq \frac{h}{2}$ (we are merging a red node)

Consider any path: parent must Black- If nodes are Red

∴ No of black nodes > # of Red nodes

(Always)

For this to be ~~less~~ than no. of red nodes need to be greater → Not possible.

* $h' \geq \frac{h}{2}$ always

$n+1 \geq 2^{h'}$

If every node has 2 children then $2^{h'}$ → minimum no. of child it must have

$2h' \geq h$

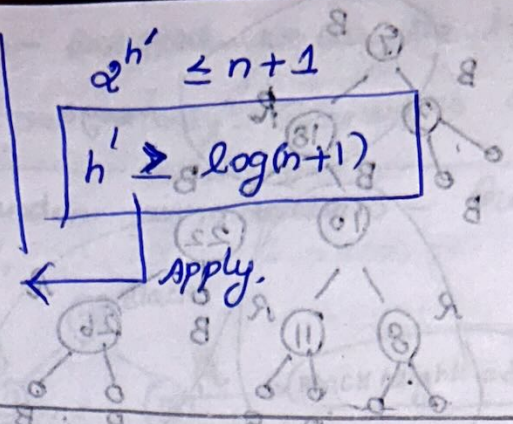
$h \leq 2\log(n+1)$

$h = O(\log n)$

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

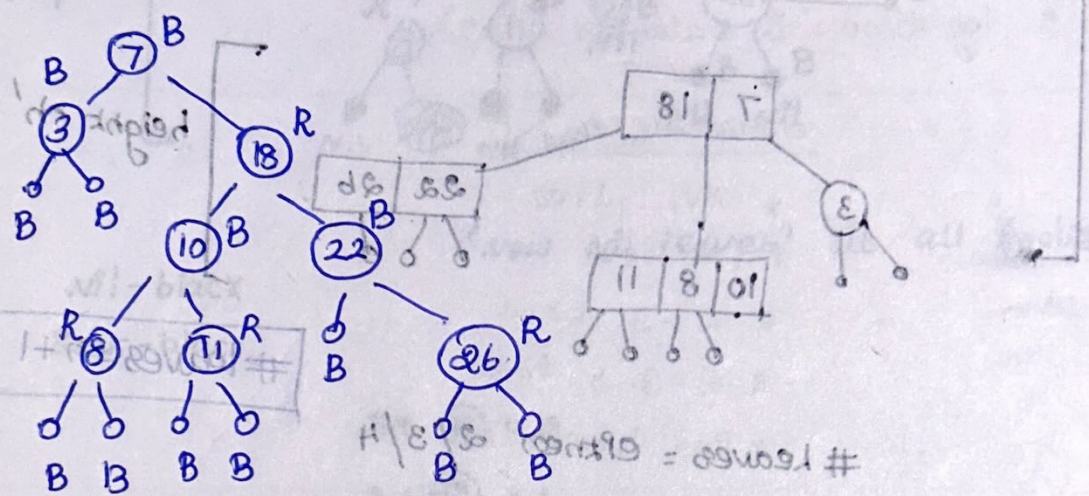
$h' \geq \log(n+1)$

Apply.



Red black tree – Contd.

* modification operations need to insert a node in



Modifying operations (Anybody can form, Insert, Delete

leave)

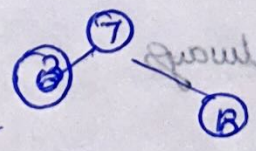
$\frac{h}{2} \leq h'$

* Advantage: Search a node : $\log_2(n)$: Comparison upto height of the tree.

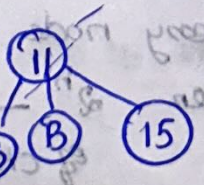
* Searching
  Finding min/max
  Succession/predecession

Insertion – Node (Like in BST)

Insert (T, 15)



$\frac{h}{2} \leq h'$



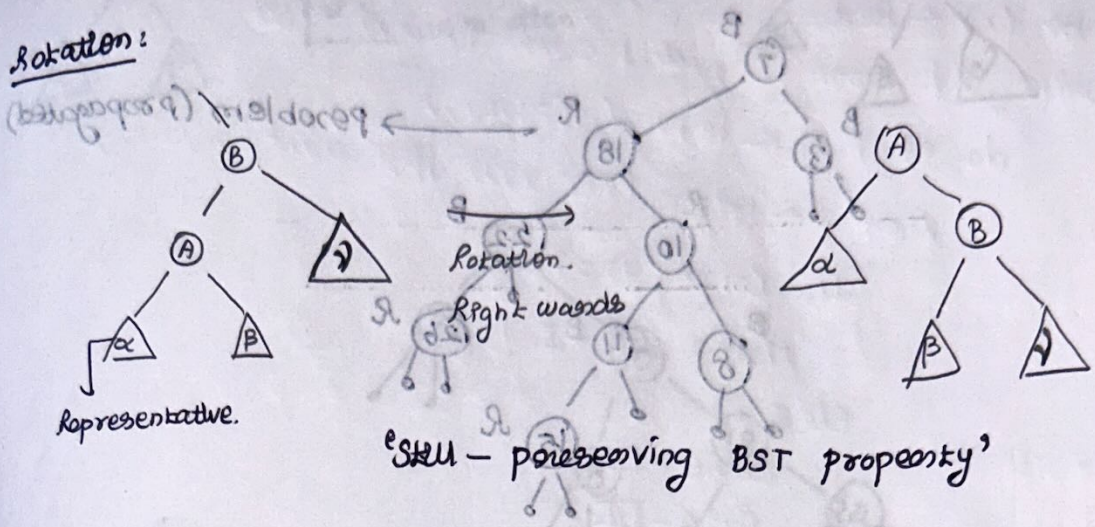
$h' \leq 1+h$

→ which Colour: Black danger (Black height got affected)!

Red Colour: Parent must be black → Affects → Black heights

Recolouring! – Restructure (Rotation)

Rotation:



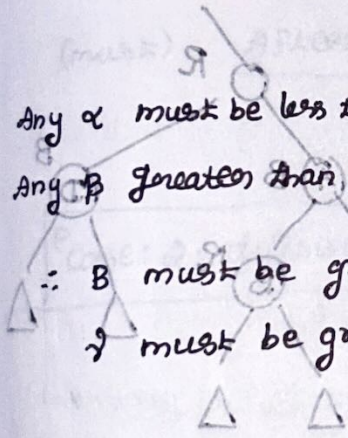
'Still – preserving BST property'

$a \in \alpha$

$b \in \beta$

$c \in \gamma$

$a \leq A \leq b \leq B \leq c$



Any α must be less than A

Any β greater than A

∴ B must be greater than A

γ must be greater than B

A greater than α

Smaller than B, β, γ

B Smaller than γ

'Same'

'we can left rotate too'!

How much time!

Just changing pointers (3 to 4) → O(1)

case: 1

* (8) (11) → same colour as , and (10)

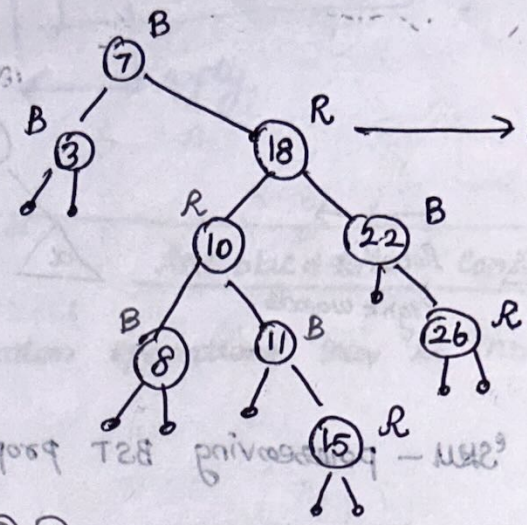
their parent is black.

Parent – Red (8), (11)

Grandparent – Black (10)

Insertion: Change Colors of (8) & (11) → to black

(10) to Red.

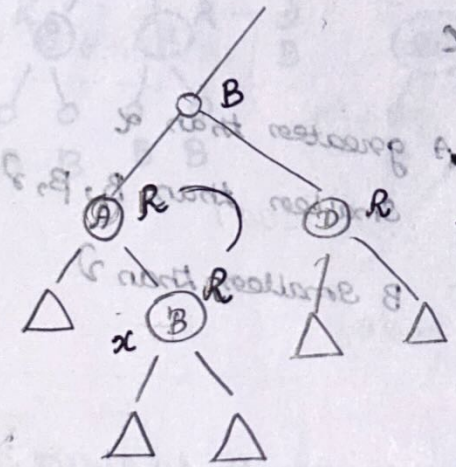


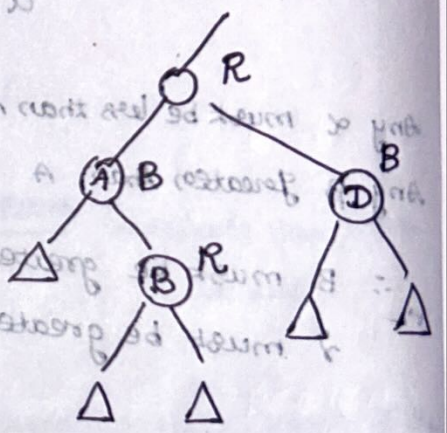
B - changed (8), (11)

R - changed (10)

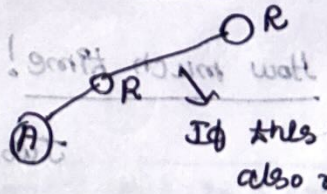
case : 1

Solution



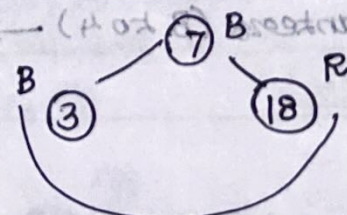


Problem



If this is also red problem.

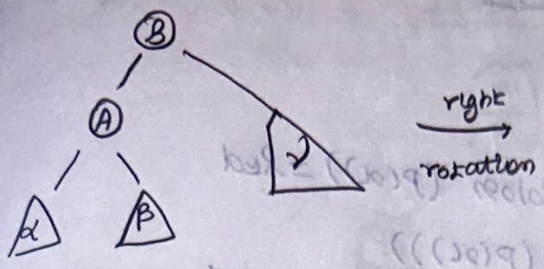
Is the problem Solved : own case

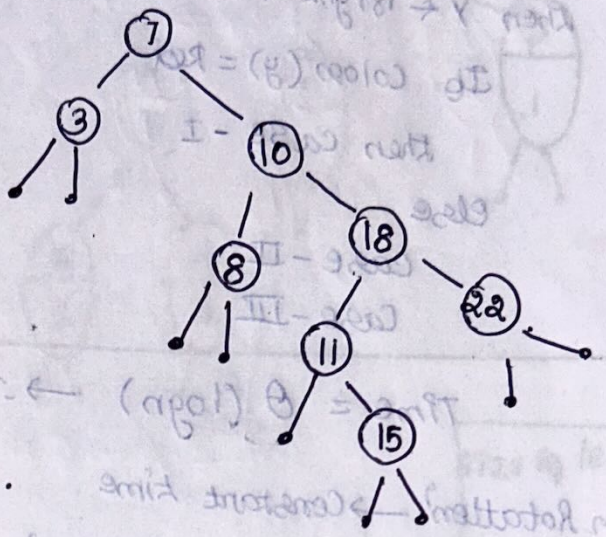


Not same colors

Case - II - perform rotation

Right rotate

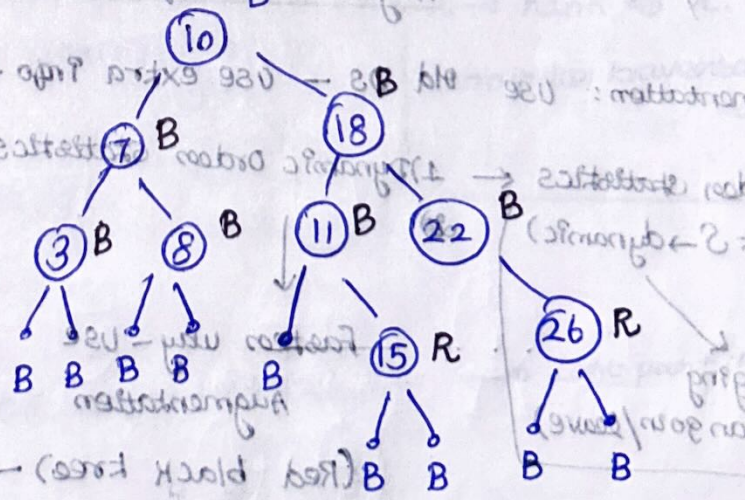




(must) After doing case-II perform case-III

~~Left~~ rotate?

case: 2: followed by case: 3



P(x) → parent(x)

Red-black tree - Insert (T, x)

1) BST - Insert (T, x)
2) Color (x) ← Red
3) while x ≠ root (T), and Color (P(x)) = Red
   do if P(x) ← left(x)
   then y ← right (P(P(x))) || y = aunt

RB - Insert (T, x)

1) BST - Insert (T, x)

2) Color (x) ← Red

3) while x ≠ root (T), and color (P(x)) = Red

do if P(x) ← left (P(P(x)))

then y ← right (P(P(x))) || y = aunt/uncle (x)

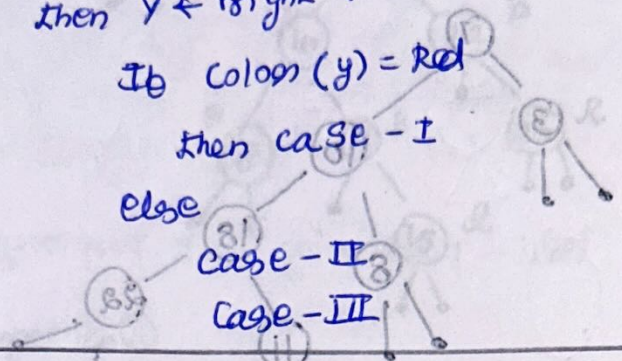
If color (y) = Red

then case - I

else

case - II

case - III



Time = $\Theta(\log n)$ → ∵ Just inserting a node

Recoloring & Rotation → Constant time

Traversing - height of the tree = $O(\log n)$

III by Deletion = needs Success array (Tedious)

$O \log(n)$ too.

Augmentation of data structure

Augmentation: Use Old DS — use extra info — Solve problem

order statistics ← 1) Dynamic order statistics (array n numbers Find ith smallest number) (Rank = ?)

(Set S → dynamic)

↙

changing

(Any can go in/leave)

↓

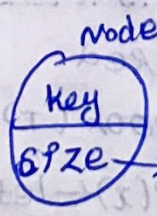
Fastest way - use Augmentation

(Red black tree) — use augmentation

---

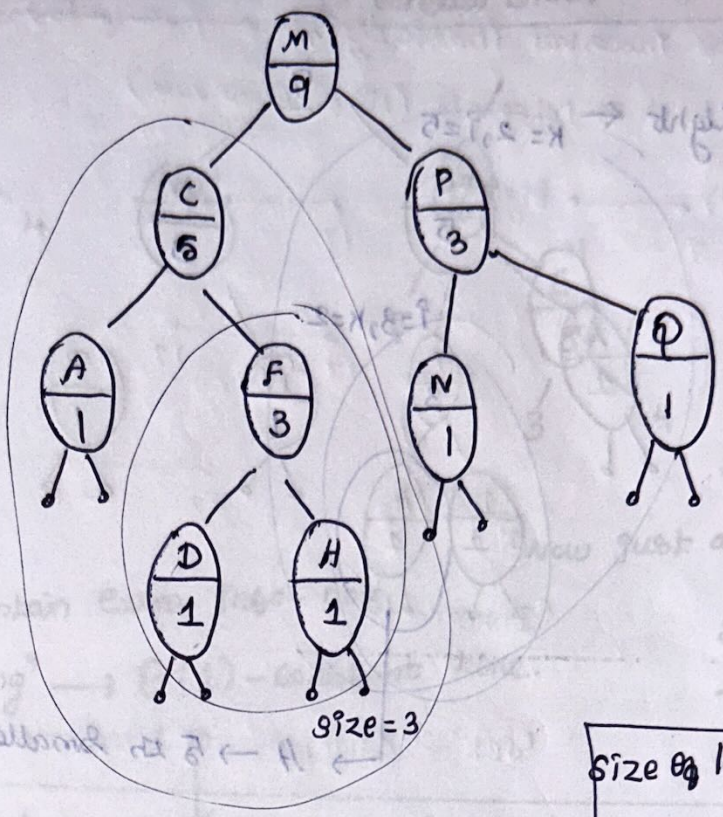
* use black - red tree for the set S

* Keep the subtree size in the nodes → Augmentation (extra info)



Augmentation.

Order Statistics – tree (Red-black tree)



size of left + size of right + size of right

size of node = 1 + left + right

How to find ith smallest element:

OS – select (T, i)

↓

Here on dynamic set!

1) x ← root (T)
2) k ← size (left(x)) + 1
3) if (i = k)
   return x
4) if (i < k)
   OS – select (left(x), i) → Left part of tree
6) if (i > k)
   OS – select (right (x), i – k)

k → Rank of x.

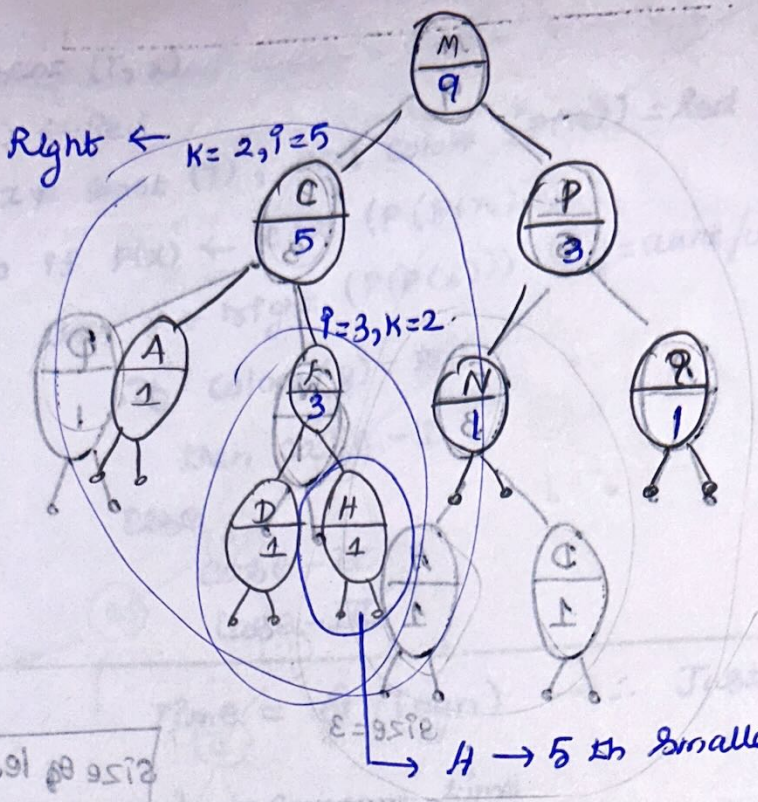
Inorder traversal – Sorted array.

5th smallest element

k = left(x) + 1

k = 5 + 1 = 6

K = 6, i = 5 (i < K) → left



Right ← K = 2, i = 5

i = 3, K = 2

∴ Right side

i = i − K

= 5 − 2

i = 3

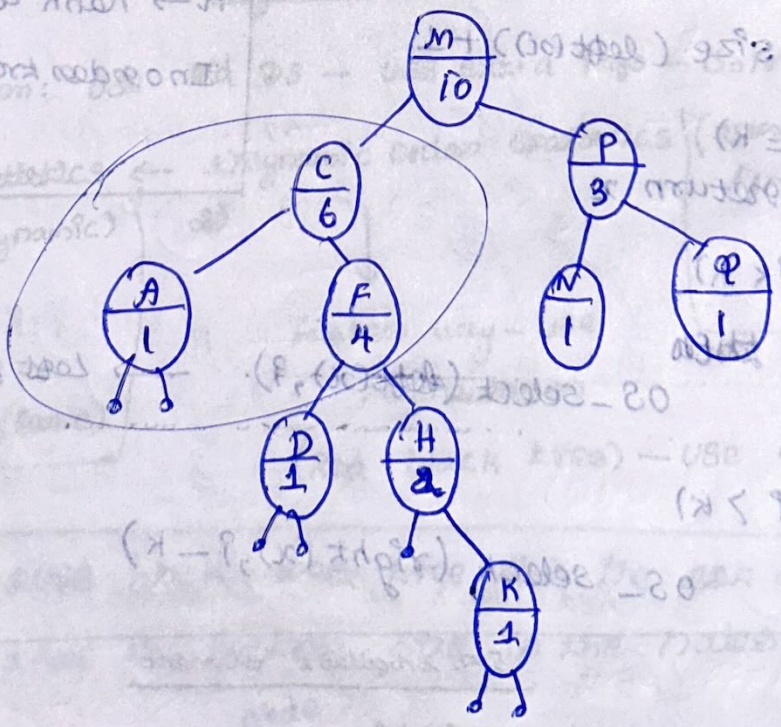
→ H → 5th Smallest element.

Running Time: O(h): Just height (going to)

↓

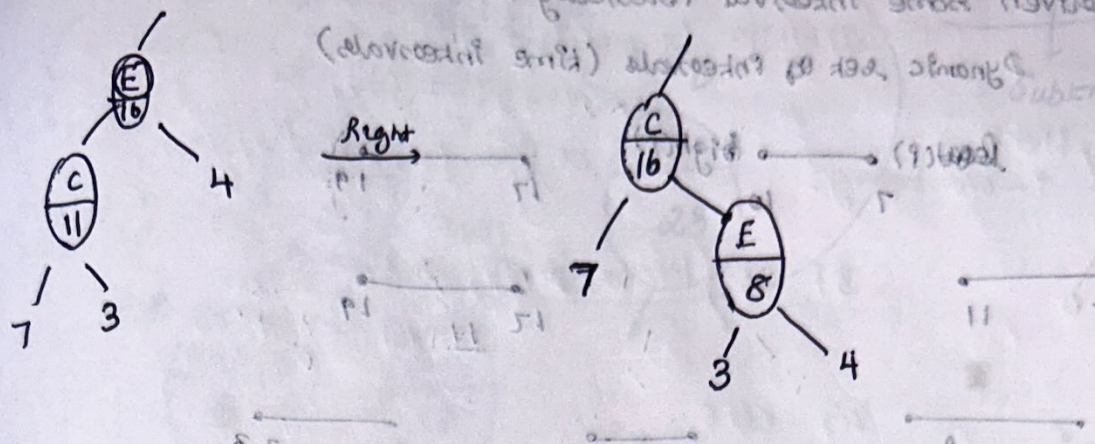
How to perform Insertion

Insert K



'Insert' – change size, make again as Red-black tree

See: In rotation operation: Is it easy to handle extra info! In constant time. — No issue



• To maintain extra info - not doing anything? → $\Theta(1)$ - constant time.

• Now just add again

$3+4+1=8$

$7+8+1=16$

If Shifting Constant time

* Insertion / Deletion → done in $(\log_2 n)$ time

$\Theta(\log_2 n)$

Instead of storing: subset size: If we just store the rank itself - then what's the issue!

* Not easy to update Rank - after inserting a node (∵ Change all element's rank) → Take $\Theta(n)$

∴ Not a good idea!

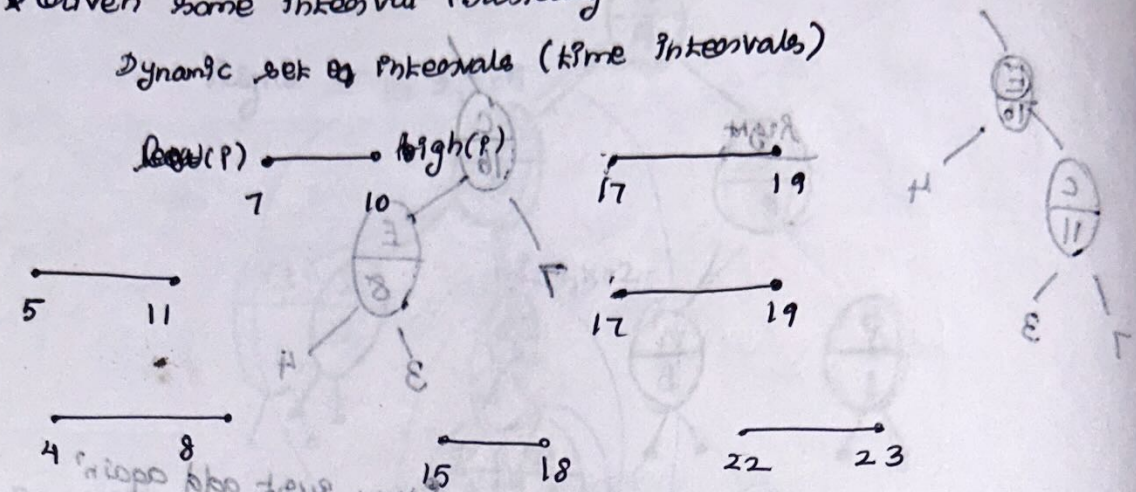
Data Structure augmentation

1) Choose underlying data structure (Red black tree)
2) Determine additional info to be stored!
3) Verify - that this additional info → Can be maintained for modifying (Insert / delete)

Interval tree – problem

★ Given some interval (basically time intervals)

Dynamic set of intervals (time intervals)



"Intervals"

"set" – Collection of intervals.

---

query: Given a given interval i, Find an interval from S which overlaps with i

Say: $i = [9, 14]$

Find interval overlapping with 9 & 14.

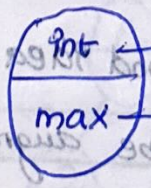
Maintain a data structure; Data structure Augmentation:

1) Red – Black tree (Underlying DS)

2) Add info → (Solve the query) →

Problem: Find query interval overlapping with query interval.

Add: info:



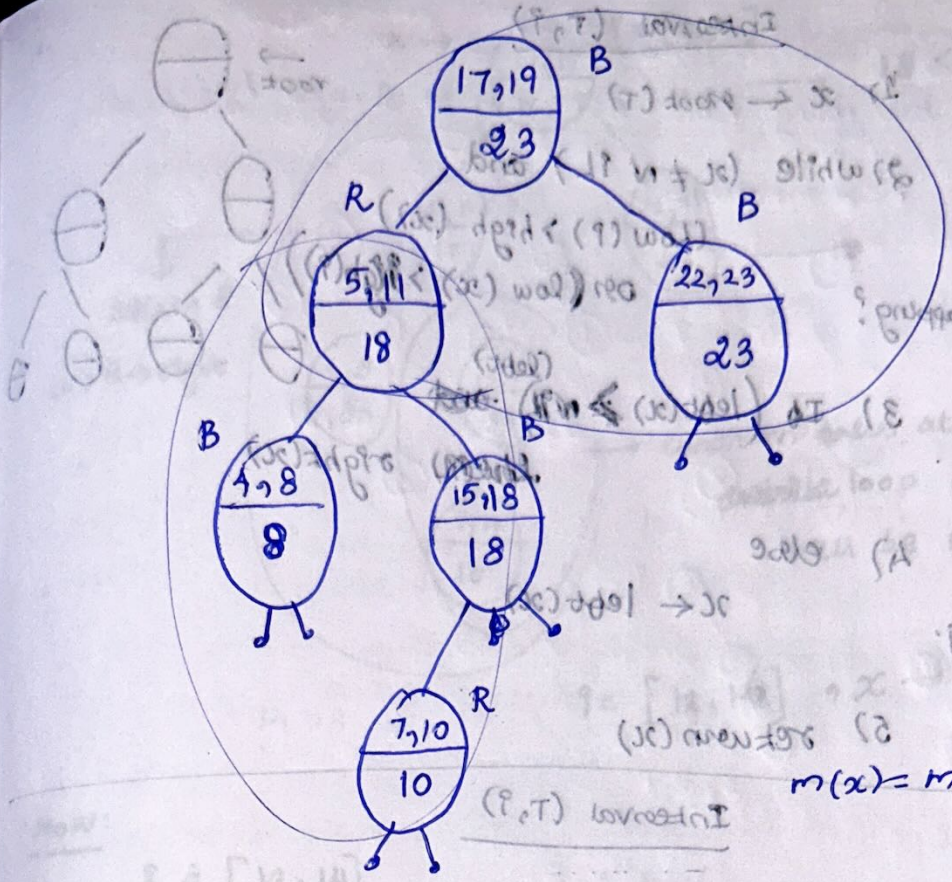
★ Keyed on low (left)

★ end points.

$M(x)$ → Largest value of the subtree rooted at x

(interval)

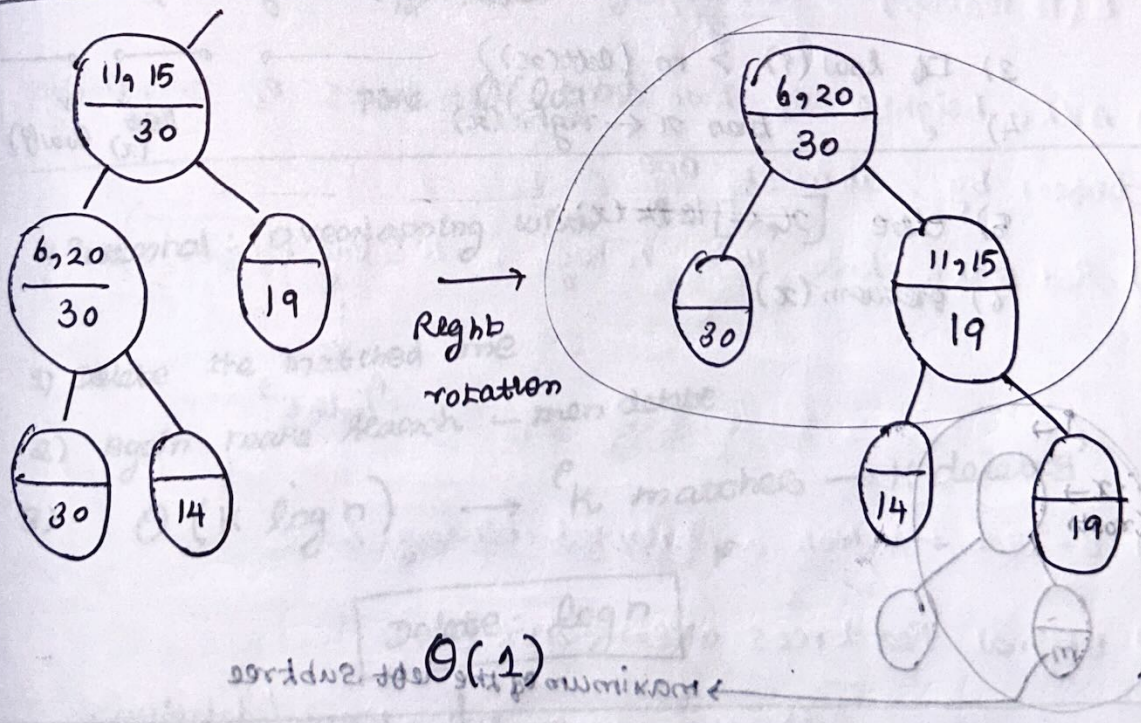
$int(x)$



[17,19]
max(x) = 19
↓
Subtree

$$m(x) = \max \begin{cases} high(x) \\ m(left(x)) \\ m(right(x)) \end{cases}$$

manage - additional Prop



$\Theta(1)$

Insertion : O (logn)

Interval (T, i)

1) $x \leftarrow root(T)$

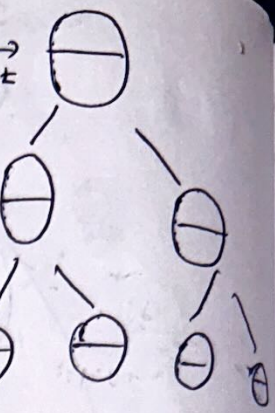
2) while ($x \neq NIL$) and (low(i) > high(x)) or (low(x) > high(i))

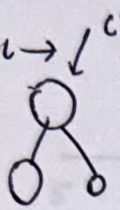
3) If (left(x) $\neq$ NIL)

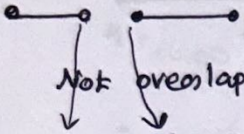
then right(x)

4) else $x \leftarrow left(x)$

5) return (x)







Interval (T, i)

1) $x \leftarrow root(T)$

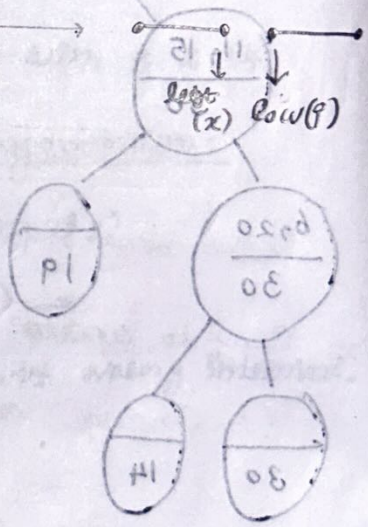
2) while ((x ≠ NIL) and ((low(i) > high(x)) or (low(x) > high(i))

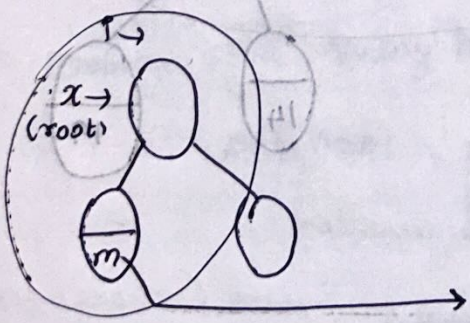
3) If low(i) > m (left(x)) ⟶

4) then $x \leftarrow right(x)$

5) else $x \leftarrow left(x)$

6) return (x)



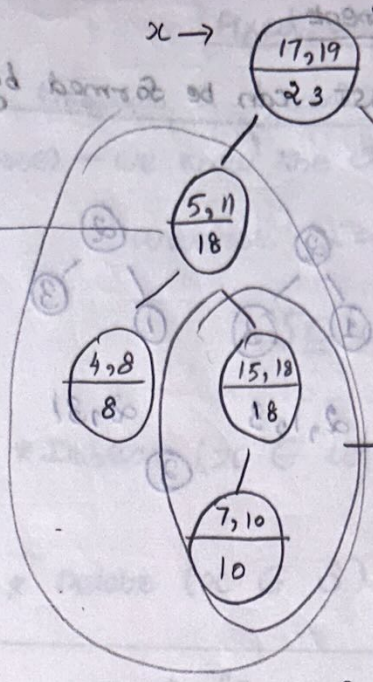


If low (i) > left (x) → maximum

∴ m(left(x)) → max of the left subtree.

∴ No use to see questions in left subtree.

(See right one).



$q = [14, 16]$

$14 < 18 \rightarrow$ left (left(x))

(we are searching something)

Root node is larger

∴ Search left

$left(x) = 8$

$14 > 8 \rightarrow$ right

condition fails at double loop (This x will be returned)

$14 > 8 \rightarrow$

$q = [14, 16]$, $x \rightarrow [15, 18]$

Now:

$q = [12, 14]$

$12 < 18 \rightarrow$ left

$12 > 8 \rightarrow$ right

$12 > 10 \rightarrow$ right (Nil) → No interval intersects

Time: $O(\log n)$

---

*Interval: Overlapping with $q = [\ ,\ ]$

1) Delete the matched one
2) Again make search → then delete.
3) $O(K \log n)$ → K matches → K deletes

Delete: $\log n$

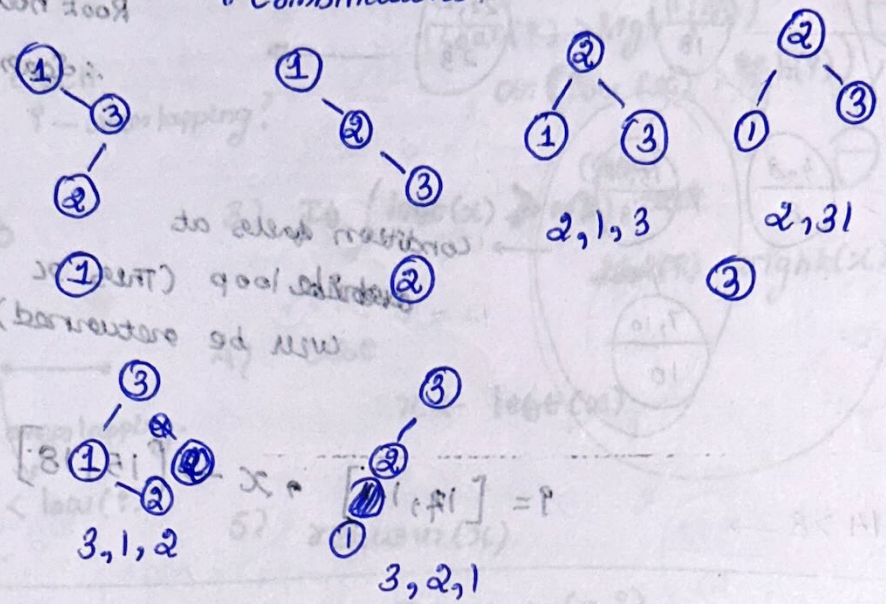
↓

Output sensitive Algorithm

Assignment

1) How many Rand BST can be formed by {1, 3, 2}

6 Combinations.



Totally: 5

2) which process forms Rand. binary tree = Stochastic pro

3) $E(\#\text{ leaves}) = \frac{(n+1)}{3}$

4) AVL, heights of two child-sub trees of any node differ by: atmost one

5) Red black tree with n keys: $n \leq \log(\sqrt{n+1})$

↓

"false"

6) B-Tree → Not a balanced binary tree

7) Optimal Red trees over AVL?

1) when more insertions / deletions.

8) BST- balanced, when the diff b/w left & right subtree of every node is not more than 1.

"heights"

9) Red-black tree: used in process schedulers, maps etc. (faster insert/deletion)