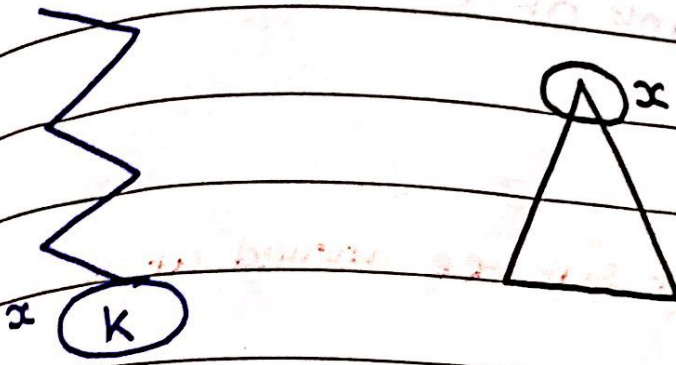
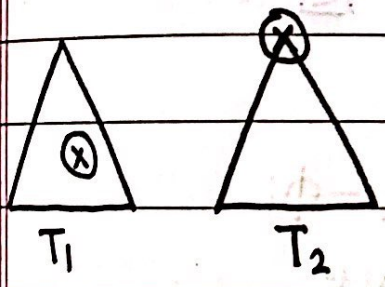
★ Splay Tree (BST)

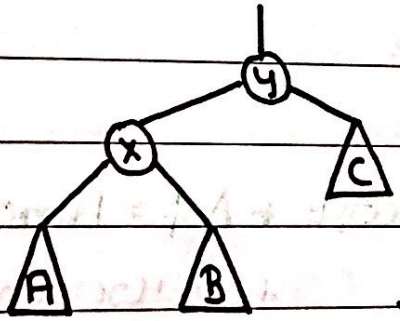


Search node with key k
make that node as root
using splaying (rotations)

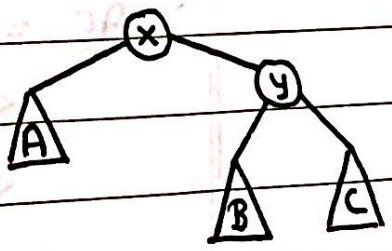
If key with k not found
make the last node in
search as the root.

★ Self-Adjusting BST: (Doesnt maintain additional info)



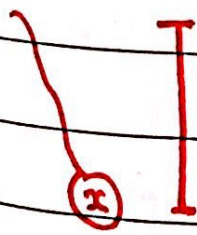


RR

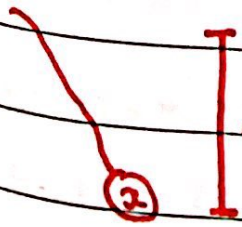


→ One level rotation : L, R

→ Two level rotation : LL, LR, RL, RR



k' : two level rot



k' : two level rot

1 : one level rot

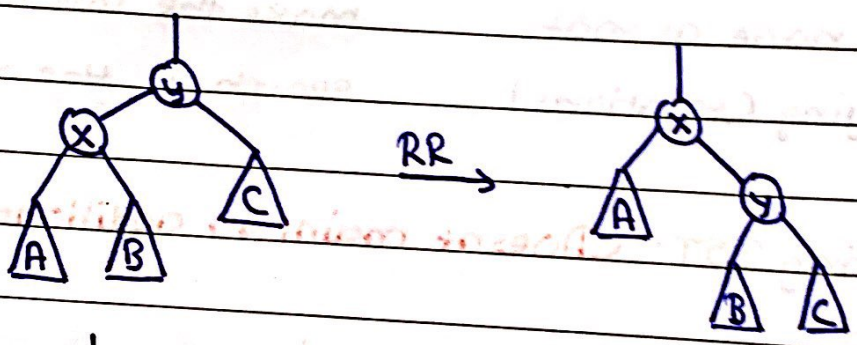
* Tree : T

$\phi(T) = \sum_{x \in T} r(x)$: rank of x

$r(x) = \log(s(x))$

↳ size of subtree rooted at x

* Amortized cost of R (One level)



ϕ_i ϕ_f

$\Delta\phi = \phi_f - \phi_i$

* Actual cost + $\Delta\phi$ = Amortized cost

1 + $(r'(x) + r'(y) - r(x) - r(y)$

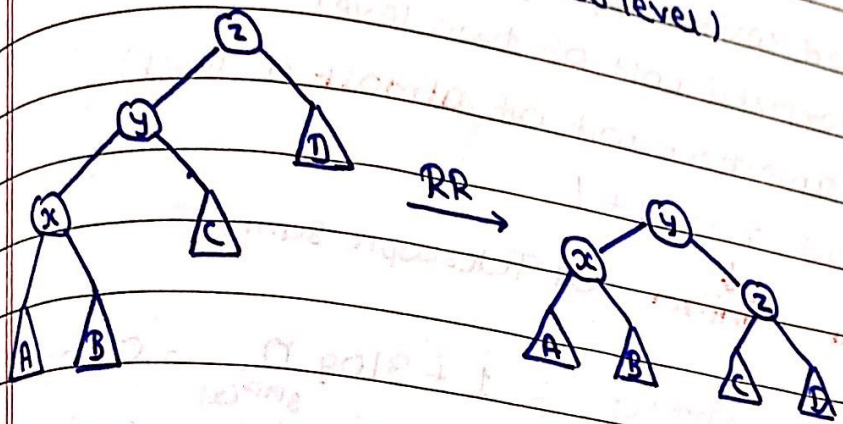
($\because r'(x) = r(y)$)

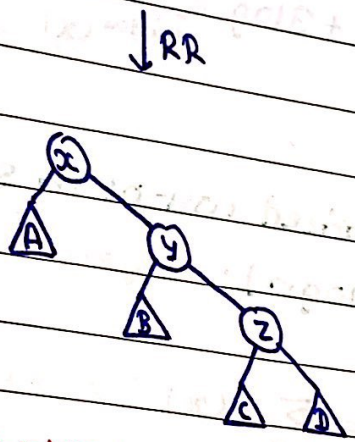
$\therefore AC = 1 + (r'(y) - r(x))$

$\leq 1 + (r'(x) - r(x))$

$\leq 1 + 3(r'(x) - r(x))$

* Amortized cost of RR (Two level)





$s(x) + s'(z) = s(z) - 1$

$\Rightarrow s(x) + s'(z) = s'(x) - 1$

$\Rightarrow \frac{s(x) + s'(z)}{s'(x)} \leq 1$

$AC = 2 + (r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z))$

$(\because r'(x) = r(z))$

$\therefore AC = 2 + (r'(y) + r'(z) - r(x) - r(y)) \quad \ldots (1)$

$\frac{s(x) + s'(z)}{2} \leq s'(x)/2$

$\Rightarrow \frac{s'(x)}{2} \geq \sqrt{s(x)s'(z)}$ (# AM-GM)

$\Rightarrow \lg \frac{s'(z)}{s'(x)} + \lg \frac{s(x)}{s'(x)} \leq -2$

$\Rightarrow r'(z) + r(x) - 2r'(x) \leq -2 \quad \ldots (\#)$

$\Rightarrow AC = 2 + (r'(y) + r'(z) - r(x) - r(y)) \leq 2 + (r'(x) + r'(z) - 2r(x))$

Using # $AC \leq 3(r'(x) - r(x))$

* Amortized cost of splay
= (Σ amortized cost of two level
+ amortized cost of atmost 1 level)
$= 3(r_f(x) - r_i(x)) + 1$

final initial (# Telescopic sum)

$\therefore 1 + 3\log \frac{S^{final}(x)}{S^{initial}(x)} = 1 + 3\log \frac{n}{s^{init}(x)} = O(\log n)$

→ Amortized cost of m splays
$O(m \log n)$

* $\phi(T) = \sum_{x \in T} r(x)$

$r(x) = \log(s(x))$

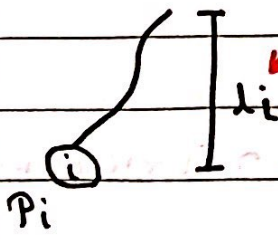
$\phi(T_0) = 0$ (Empty tree)

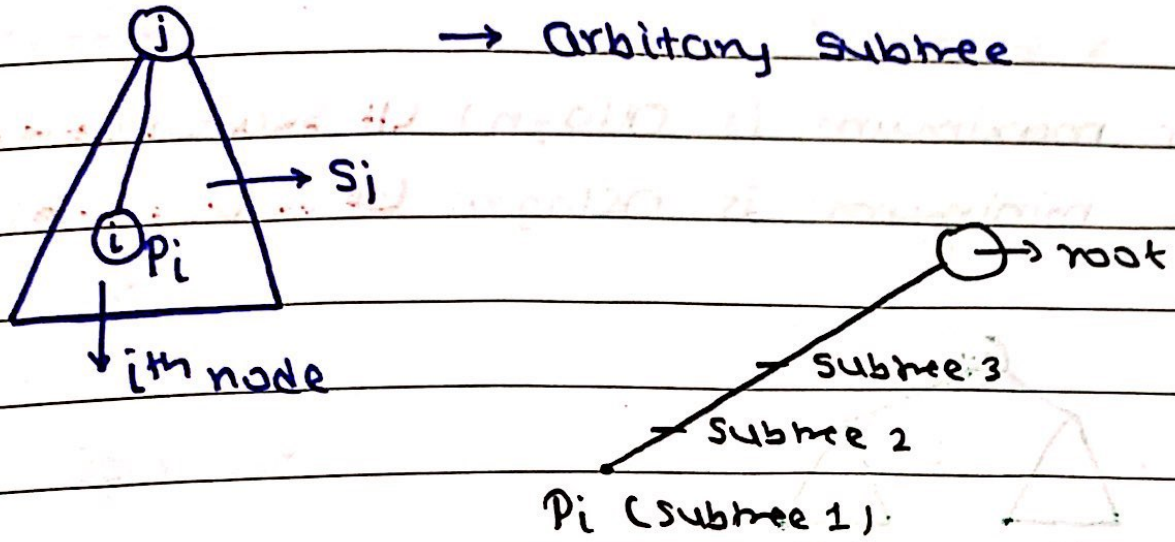
→ Insert, Search, Del operations performed with empty tree as base.

T.P.T. $\therefore$ after any op. $\phi(T) \geq \phi(T_0)$

* Expected cost of search:

→ $P_i(i)$: Prob to search ith node





$\therefore$ p_i is counted $(l_i + 1)$ times

$\therefore \sum (1 + l_i) p_i = \sum_j s(j)$

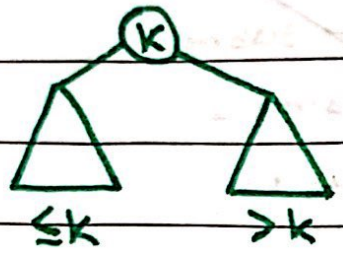
Amortised analysis : Pseudo avg worst case

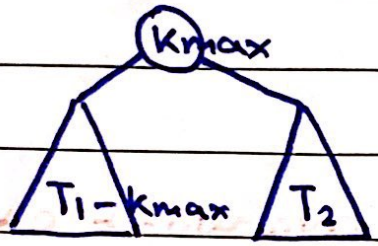
$\therefore \sum_j \log s(j) = \sum \log[(1 + l_i) p_i] \geq 0$

✱ Join

→ AC for maximum is O(log n) (# search (+∞))

" minimum is O(log n) (# search (−∞))



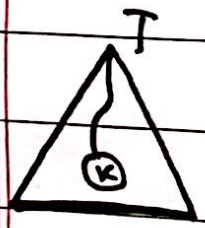


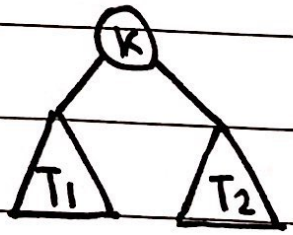
→ AC: O(log n)

→ Actual cost:

$1 + 3(\ldots) + \square + \Delta\phi = O(\log n)$

✱ Split(k)



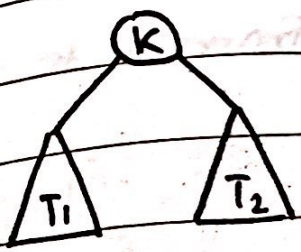


Search for k & then

Splay it

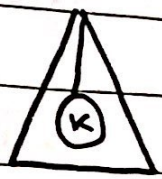
AC: O(log n)

* Insert (k)

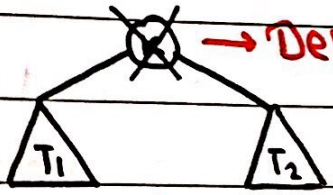


Split wrt k

* Delete k



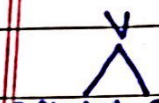
↓ Split wrt (k)



↓ Join (T_1, T_2)

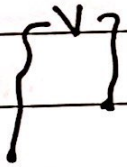
★ Leftist Tree (Follows HOT property)

s(v): shortest dis from v to leaf



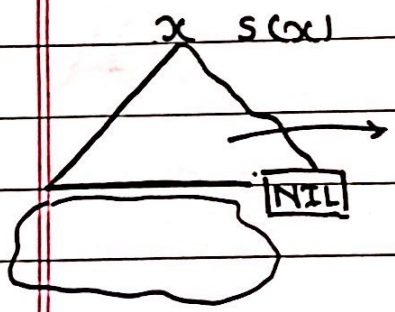
$s(v) = 1 + \min(s(left), s(right))$

★ Height Biased Leftist tree



h(left) ≥ h(right)
for every subtree

→ Among all the paths right spine is the shortest



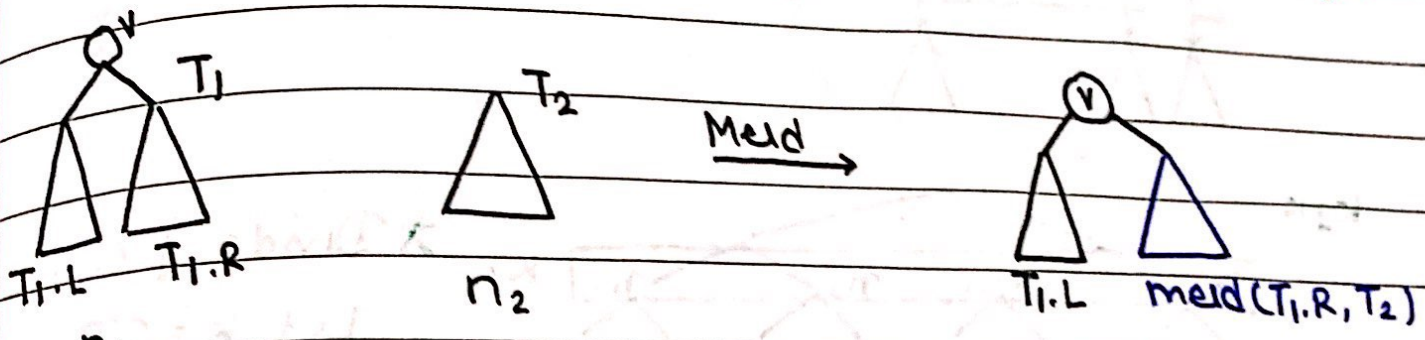
$2^0 + 2^1 + .. + 2^{s(x)-1} = 2^{s(x)} - 1$

$\therefore 2^{s(x)} - 1 \leq \#(T_x)$ → subtree

∴ s(root) is O(log n)

* Meld (T_1, T_2)

WLOG let $T_1(root).key > T_2(root).key$ (else swap)



(It is a leftist tree if not toggle left & right subtree)

* Update s(x) values of right spine

TC: $O(\log n_1) + O(\log n_2)$ (# right spine length)

$\leq O(\log n_1 n_2) \leq O(\log n)$

(# $n = n_1 + n_2$ & AM-GM)

* Insert (K)

Key node a heap of size 1

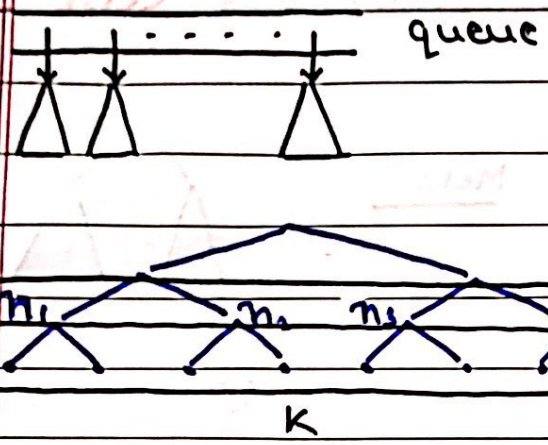
Meld 2 trees : $O(\log n)$

* Extract max

Remove the root & meld left & right subtrees

$O(\log n)$

* K-max HBLHS



* Divide & Conquer

Let $k = 2^m$

* $\sum_{i=1}^{k/2} O(\log n_i)$, $\sum_{i=1}^{} n_i = n$

$$\sum_{i=1}^{k/2} \log n_i = \log n_1 \ldots n_{k/2} \leq \log \left(\frac{\sum n_i}{k/2} \right)^{k/2} \quad \text{(AM-GM)}$$

$$= O(k \log n/k)$$

When $k = n$ algo is $O(n)$

∴ For convinence we can say the algo is

$O(k + k \log n/k)$

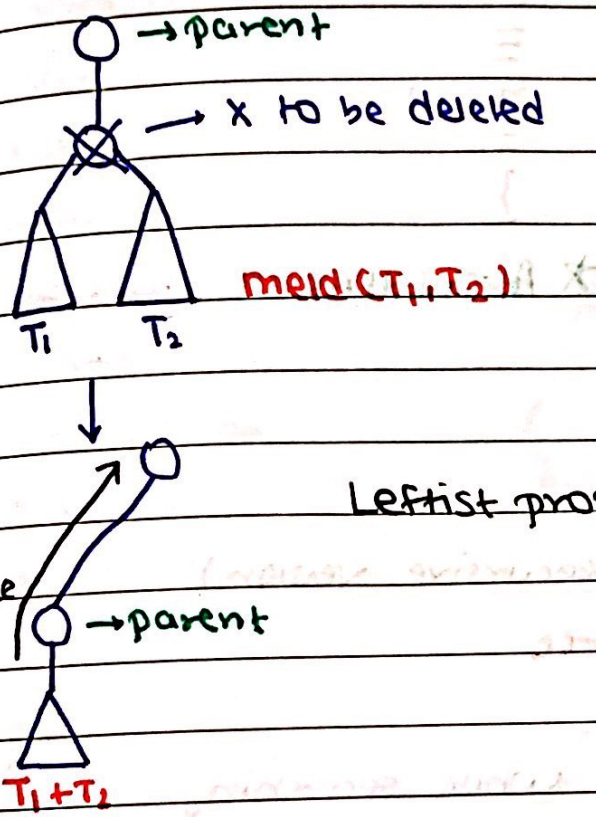
* $$\sum_{i=0}^{\log k} O\left(\frac{k}{2^i} \log \frac{n}{k} 2^i\right) = \sum_{i=0}^{\log k} O\left(\frac{k}{2^i} \log \frac{n}{k} + \frac{k}{2^i} \log 2^i\right)$$

$$= O\left(k \log\left(\frac{n}{k}\right)\left(1 + \frac{1}{2} + \ldots + \frac{1}{2^{\log k}}\right) + \sum_{i=0}^{\log k} k \log 2 \times \left(\frac{i}{2^i}\right)\right)$$

$$= O(k + k \log n/k)$$

∴ Bulid Heap in $O(n)$ # $k = n$

* Delete: Pointer to the node to be deleted given



↳ But path may have n nodes

But s(x) = O(log n)

& all values of s(x) are distinct on the path ← the modified s(x) values

∴ By PHP we have to change finitely many s(x) values & then stop

∴ Delete : log(n)

* Increase Key:

Delete k prev

Insert k new

* This DS is not self adjusting

* rec() (rec in One direction)

```
{
  ==
  rec()
}
```

Tail Recursion

Compilers can optimise better

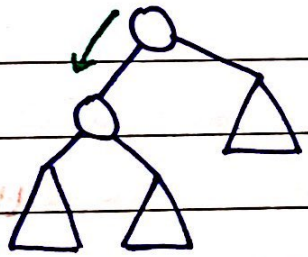
* rec()

```
{
  ==
  rec()
  ==
}
```

* Avoid this

→ Splay Tree (Tail Recursive version)

Top down Splay Tree



while searching

if go left RR

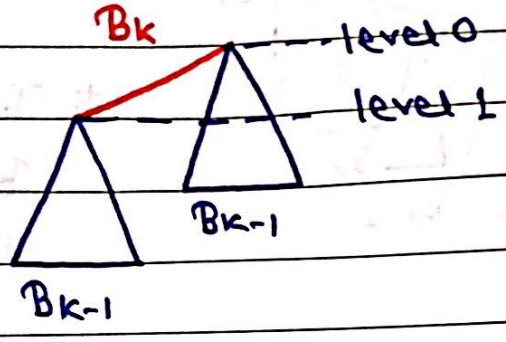
else LR

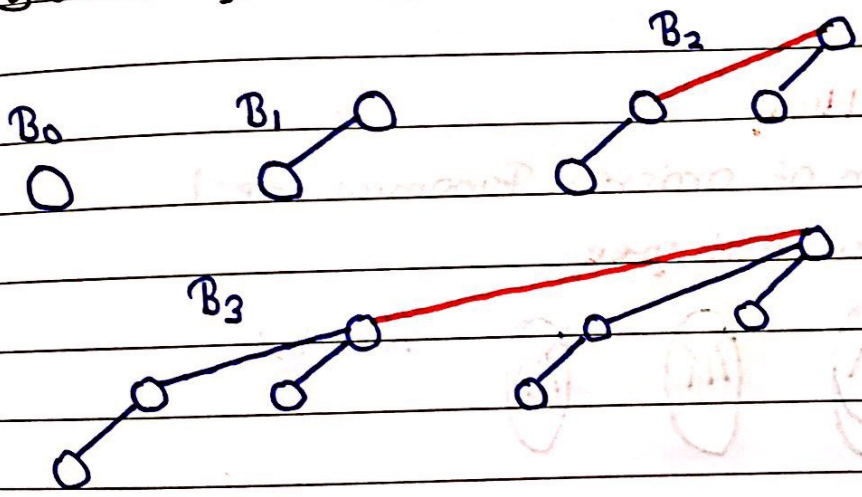
maintain pointer to next subtree to be traversed

* We cant make HBLH Tail recursive.

★ Binomial Heap
(unordered set of binomial trees)

① max Heap
② get max
③ meld
④ insert
⑤ extract max
⑥ increase key
⑦ delete key
⑧ build key





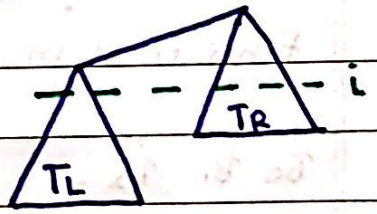
→ B_k

no. of nodes $n = 2^k$

Height = k

$\therefore h = O(\log n)$

Nodes at depth i : $\binom{k}{i}$

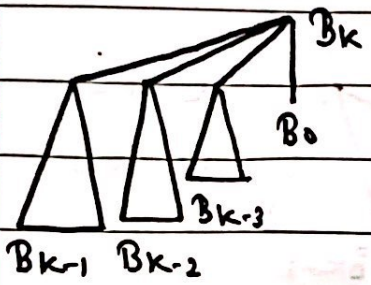


#(depth i) = #(depth i in T_L) + #(depth i in T_R)

$\therefore \binom{k-1}{i-1} + \binom{k-1}{i} = \binom{k}{i}$ (# Pascal)

Deg(root) = k (# $^kC_1 = k$ or use induction)

→ max deg is for root in B_k



(# Telescope

$B_k = B_{k-1} + B_{k-1}$

$B_{k-1} = B_{k-2} + B_{k-2}$

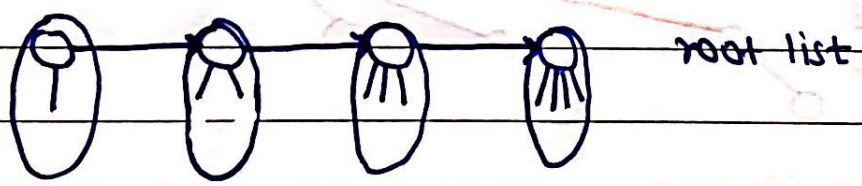
⋮

$B_1 = B_0 + B_0$)

★ **Binomial Heap**

(Collection of ordered Binomial trees)

→ **Ordered wit degree**



★ each is a max heap

→ walk along the root list

Find max in O(root list . si) (Naive)

B_0 B_1 B_2 . . . .

If B_i present ith bit set to 1 else 0 ($b_i = 0$ or 1)

$\therefore n = \sum_{i=0}^{\lfloor \log n \rfloor} b_i 2^i$

* $\lfloor \log n \rfloor + 1$ bits required to store n keys

max deg $\lfloor \log n \rfloor$ # $B_{\lfloor \log n \rfloor}$ exists

$\therefore$ Length of root list $\lfloor \log n \rfloor + 1 = O(\log n)$