DSAP

Advance problem solving

~~stacks / queues / lists~~

⇒ Introduction to Algo – Thomas H. cormen

⇒ Data structures & Algo analysis in c – mark Allen weiss

1 Lab sessions every week

- about 4-5 problems to be solved in the session (TA's to assist)
- credits are given for lab performance

several Hw assign.

- About 5, one every 2 weeks
- each set to have about 6-7 problems

⇒ strictly, no plagiarism

→ instructor available via office hours
- tentatively, tuesday 4-5 pm

→ seek an appointment for meeting outside of office hours.

→ email communication is also OK
- vineet gandhi, vgandhi@iit.ac.in

→ very important: seek help early enough

Assessment

- 2 Quiz (5% each) + 1 midsem exam (10%) + 1 Final exam (2
- Assignment (25%) + weekly lab (10%)
- Lab exams (25%)

- If copying is detected, you will get 0 marks for the assignm
- This policy might slightly vary as course evolve over the semester

Lec 1) Introduction & GCD

$$x = p_1^{a_1} \cdot p_2^{a_2} \cdot p_3^{a_3} \cdots p_n^{a_n}$$

$$y = p_1^{b_1} \cdot p_2^{b_2} \cdot p_3^{b_3} \cdots p_n^{b_n}$$

$$GCD(x,y) = p_1^{\min(a_1,b_1)} \cdot p_2^{\min(a_2,b_2)} \cdots p_n^{\min(a_n,b_n)}$$

int isprime[N];

-1 -1 2 -1 2 -1 2 3 2 -1 2 -1

(2) (3) ~~4~~ (5) ~~6~~ (7) ~~8~~ ~~9~~ ~~10~~ (11) ~~12~~ 13

2 3 2 -1 2 -1 2

~~14~~ ~~15~~ ~~16~~ (17) ~~18~~ (19) ~~20~~

$$\left(\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \cdots\right)$$

$$= n\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \cdots\right) \rightarrow \text{(sum diverges)}$$

$= (n \log\log n) \Rightarrow$ sieve of eratosenis

Method 2 (Feumat theorem)

$\sqrt{255+1^2}$

$\sqrt{255+a^2} = b$

$255 + a^2 = b^2$

$255 = (b-a)(b+a)$

↙

odd

35

$\sqrt{35+1} = 6$

$(6+1)(6-1)$

$\sqrt{21+1}$

$\sqrt{21+2^2} = \sqrt{25} = 5$

$(5+2)(5-2)$

$a^2 - b^2 = (a+b)(a-b)$

(Applied on 2 odd multiples)

Euclid Algo

$a = bq + r$

$GCD(a, b) = GCD(b, r)$

```
GCD(a,b)
  if(a % b == 0)
     return b;
  else
     return GCD(b, a%b);
```

GCD(32, 24)

GCD(24, 8)

$a = bq + r$

d is divisor of a & b $(a|d \;\&\; b|d)$ then $r|d$

$r = a - bq$

$= (t_1 \cdot d - t_2 \cdot dq) = (t_1 - t_2 q)d$

d is divisor of b & r $(b|d, r|d)$

$a = bq + r$ then $(a|d)$

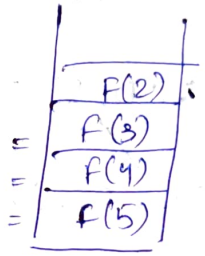
LCM = product of 2 nos / & GCD

Lec-2) Recursion

```
Fact(n)
{
  f = 1;
  for (i = 1 to n)
  {
    f = f * i;
  }
  return f;
}
```

```
Fact(n)
{
  if (n == 0)
    return 1;
  else
    return n * Fact(n-1);
}
```

Fact(5)



Recursion
- base case
- call to itself
- decrement by moving towards the base case

Imp Exam: Prove by induction

$1+2+3+\dots+n = \frac{n(n+1)}{2}$

(1) Base case: $1 = \frac{1(1+1)}{2}$

(true)

(2) Assume its true for k

$1+2+3+\dots+k = \frac{k(k+1)}{2}$

(3) For (k+1),

$LHS = 1+2+\dots+k+(k+1)$

$= \frac{k(k+1)}{2} + (k+1)$

$= (k+1)\left(\frac{k}{2}+1\right) = \frac{(k+1)(k+2)}{2} = RHS$

(so true for (k+1))

$$T(n) = T(n-1) + 3$$
$$= T(n-2) + 6$$
$$= T(n-3) + 9$$
$$T(n) = T(n-k) + 3k$$
$$n - k = 1$$
$$k = n-1$$
$$T(n) = 3(n-1) + 1$$
$$= 3n - 3 + 1$$
$$\boxed{T(n) = 3n - 2} = O(n)$$

Fibonacci

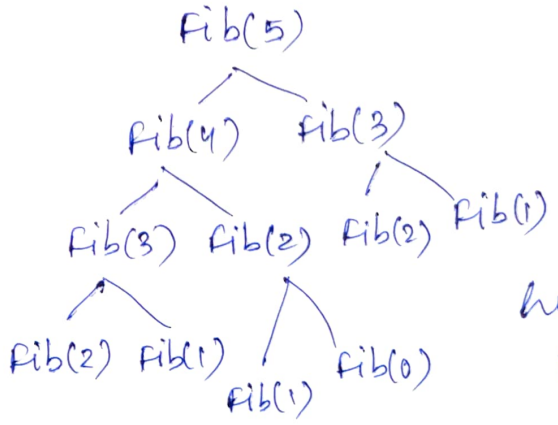
Fib(n)

0 1 1 2 3 5 8 . . .

```
{
  int prev = 0;
  int next = 1;
  int curr;
  for(i = 2 to n)
  {
    curr = prev + next;
    prev = next;
    next = curr;
  }
  return curr;
}
```

```
if(n == 0 || n == 1)
    return n;
```

```
Fib(n)
{
  if(n == 0 || n == 1)
    return n;
  else
    return Fib(n-1) + Fib(n-2);
}
```



how many function calls? (write code)

$$T(n) = T(n-1) + T(n-2) + c$$

$$T(n) = 2T(n-2) + c \quad [T(n-1) \approx T(n-2)]$$

$$T(n) = 2T(n-2) + c$$

$$T(n) = 2[2T(n-4) + c] + c$$

$$T(n) = 4T(n-4) + 3c$$

$$= 4[2T(n-6) + c] + 3c$$

$$T(n) = 8T(n-6) + 7c$$

$$\boxed{T(n) = 2^k T(n-2k) + (2^k - 1)c}$$

$$n - 2k = 0$$

$$n = 2k$$

$$k = n/2$$

$$T(n) = 2^{n/2} + (2^{n/2} - 1) \cdot c$$

$$= 2^{n/2}(1 + c) - c$$

$$\boxed{T(n) = \Theta(2^{n/2})}$$

(OR)

$$T(n) = 2T(n-1) + c$$

x^n

$$x^n = \begin{cases} x^{n/2} \times x^{n/2}, & \text{if } n \text{ is even} \\ x \times x^{n-1}, & \text{if } n \text{ is odd} \\ 1, & \text{if } n \text{ is } 0 \end{cases}$$

```
pow(x, n)
{
  if (n == 0)
    return 1;
  if (n % 2 == 0)
  { return pow(x, n/2) * pow(x, n/2);
  }
  else { return x * pow(x, n-1);
  }
}
```

```
int p = pow(x, n/2);
return p * p;
```

Modular exponentiation

$x^n \bmod m$

$(a \times b) \% m = \{(a \% m) \times (b \% m)\} \% m$

ex $\Rightarrow$ $63 \% 5 = 3$

$\{(7 \% 5) \times (9 \% 5)\} \% 5$

$= (2 \times 4) \% 5$

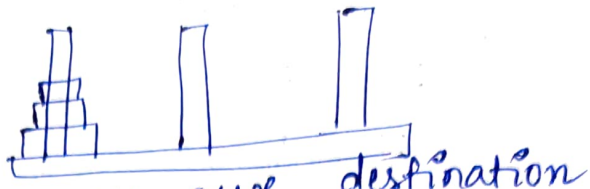
$= (8 \% 5)$

$= 3$

$$x^n \% m = \begin{cases} \{(x^{n/2} \% m) \times (x^{n/2} \% m)\} \% m, & \text{if } n \text{ is even} \\ \{(x \% m) \times (x^{n-1} \% m)\} \% m, & \text{if } n \text{ is odd} \\ 1 & ; \text{ if } n = 0 \end{cases}$$

(write code)

Towers of hanoi

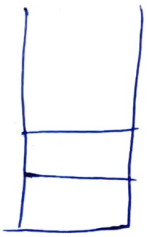


source aux destination

Towers (n, s, d, aux)
using

```
Towers (n-1, s, aux, d)
Towers ( 1, s, d, aux)
Towers (n-1, aux, d, s)
```

Stacks :-

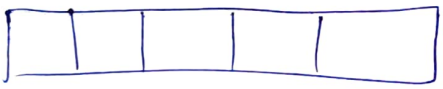


s.push (x)
s.pop ()
s.top ()

s.isempty ()

implementation

→ stack using array



top = -1

```
isempty (s)
{  if (top == -1)
       return true;
   else
       return false;
}
```

```
push (s, x)
{  top = top + 1;
   s[top] = x;
}
```

Imp**

top = -1

4	7	8	14	12	9	200	24

WC = O(n)
AC/Best case = O(1)

4	7	8	4	12	9	200	24	x

double the size

→ Balanced parenthesis

```
if (p == '{' or p == '(' or p == '[')
    s.push(p);
else if (p == '}' or p == ')' or p == ']')
{ if { s.isempty() || !ispair(p, top())
        return false;
    }
  s.pop();
}
s.empty ? true : false;
```

Algo.

→ Infix, prefix & postfix :-

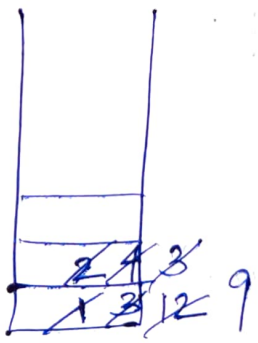
~~(x+y)*z-y~~

infix : (x+y)*z-y

prefix : ~~+*xyzy~~ -*+xyzy

postfix : xy+z*y-

postfix evaluation



left to right →

xy+z*y-

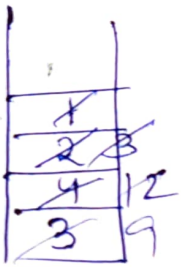
1 2 + 4 * 3 -

1 + 2 = 3
3 * 4 = 12
12 - 3 = (9)

second popped − first popped

Prefix evaluation

- * + x y z 4
- * + 1 2 4 3 ← (right to left)



2 + 1 = 3

4 * 3 = 12

1 + 2 = 3

3 * 4 = 12

12 - 3 = 9

first element poped — second element pop

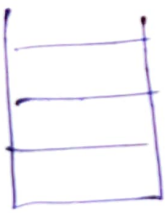
Infix to postfix conversion :-

=> Check the parenthesis is balanced or not before processing

(x + y) * z - 4

x y + z * 4 -

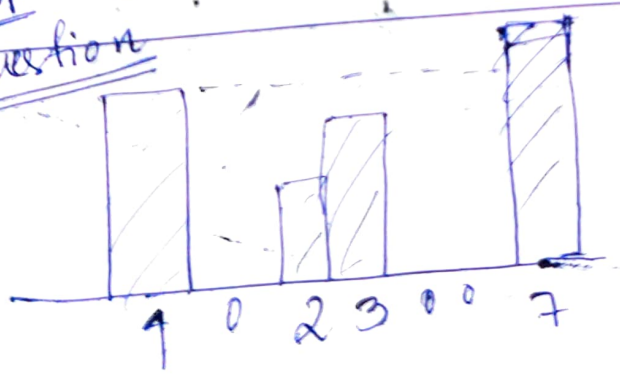
x y + z * 4 -



(x + y / z) * 4 + 9

x y z / + 4 * 9 +

Imp Question (Histogram)



Question

1	4	0	0	7	4	14

↔ next larger number

4	7	7	7	14	14	-

10/08/23

cgraph paper

Dynamic Programming :-

Q) Image resizing — we want to perform 1/3rd reduction in width.

Naive solution — ① cropping
② scaling
③ Letter boxing

Solution: Content aware

Algorithm: seam carving → (OR) saliency

→ gradiant magnitude — compute & give priority
→ find a ~~path~~ connected path & then remove them
→ find the path of minimum cost ~~in~~ in the importance image.

~~dp matrix~~

1	4	3	5	2
4	3	8	4	5
8	5	7	6	5
6	12	8	14	5

cost (matrix)

00 01
10 11

1	4	3	5	2
3	2	5	2	3
5	2	4	2	1
1	7	3	9	4

importance image (M)

$$C(i,j) = M(i,j) + \min \begin{cases} C(i-1,j-1) \\ C(i-1,j) \\ C(i-1,j+1) \end{cases}$$

(machine learning algo can be used)

for importance matrix)

100	200	200	20
40	0	0	100

→

100	0	180

Dynamic Prog.

Algorithm dynamic time warping (DTW)
↳ read this

Q1
#Longest common subsequence

HIERO GLY PHOLOGY
MICHAE LANGELO

IEGLO

Application
⇒ genome sequencing (something in bio)

"aab", "azb"

LCS("aab", "azb") = 1 + LCS("aa", "az")

max(LCS("a", "az"), LCS("aa", "a"))

LCS ("AXYT", "AYZX")

max{LCS("AXY", AYZX), LCS(AXYT, AYZ)}

max{LCS(AX, AYZX), LCS(AXY, AYZ)}

max{LCS(AXY, AYZ), LCS(AXYT, AY)}

computed twice

[computed multiple times in recursion]

as there is a match

	""	A	G	G	T	A	B
""	0	0	0	0	0	0	0
G	0	0	1	1	1	1	1
X	0	0	1	1	1	1	1
T	0	0	1	1	2	2	2
X	0	0	1	1	2	2	2
A	0	1	1	1	2	3	3
Y	0	1	1	1	2	3	3
B	0	1	1	1	2	3	4

→ length of longest common subsequence

GTAB

Q2 Longest Increasing subsequence

	12	26	8	33	25	45	38	70
LIS	1	~~1~~ 2	1	~~1~~ 3	~~1~~ 2	~~1~~ 4	~~1~~ 4	~~1~~ 5
	↑ i	↑						

12, 26, 33, 45, 70

12, 26, 33, 38, 70

$\frac{n(n+1)}{2}$ comparison

$T(n) = O(n^2)$

Q3 0/1 KNAPSACK

* try to implement using

weight	value
5	$60
3	$50
4	$70
2	$30

Item	0	1	2	3	4	5
$I_1, w=5, v=60$	0	0	0	0	0	60
$I_2, w=3, v=50$	0	0	0	50	50	60
$I_3, w=4, v=70$	0	0	0	50	70	70
$I_4, w=2, v=30$	0	0	30	50	70	80

$50 + v[i-1, 2]$

60

$30 + v[3, 3]$

$w \geq w_i$

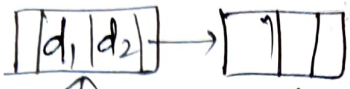
~~$v[i][w] = v_i + v[i-1][w - w_i]$~~

$$v(i, w) = \max \begin{cases} v_i + v[i-1, w - w_i] \\ v[i-1, w] \end{cases}$$

Q4 # Fractional knapsack → Greedy solution

HMM (hidden markov model)

Linklist

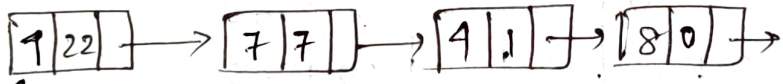


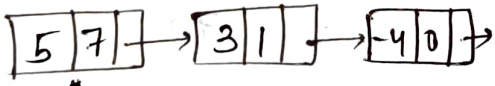
Application of LL :- represent polynomials

$4x^{22} + 7x^7 + 4x + 8$

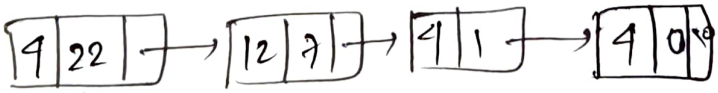
$5x^7 + 3x - 4$

↳ using array → wastage of space
↳ non-integer ~~exp~~ value
↳ cannot be stored
↳ -ve co-efficient





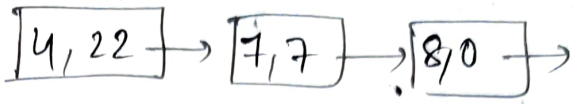
sparsity & unstructured data (for LL)

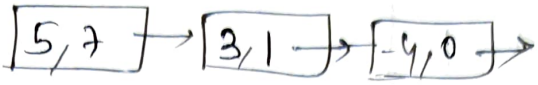


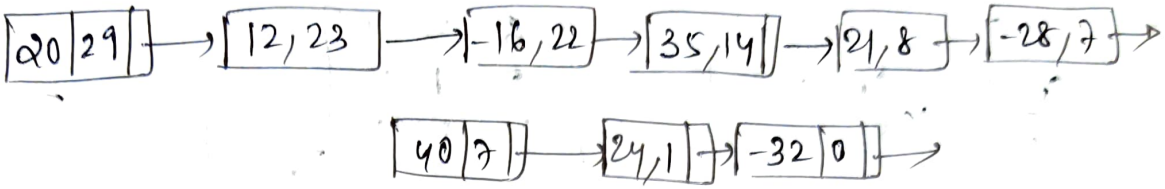
addition – $O(n_1 + n_2)$

multiplication

$T(n) = n \times m + n \log n + m \log m + n \times m$

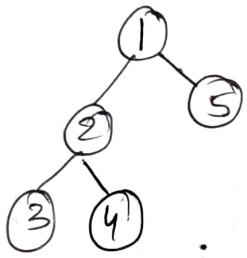






LL Application — mergesort to sort the linklist

Matrix Multiplication :-



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

H	C	V
1	2	10
1	3	12
2	1	1
2	3	2

$$= \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \rightarrow \text{zero}$$

0	10	12
1	0	2
0	0	0

2	5	0
0	1	0
8	0	0

H	C	V
1	1	2
1	2	5
2	2	1
3	1	8

Matrix ⇒ row-column formulation

$$\begin{bmatrix} 3 & 2 \\ 1 & 1 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} 2 & 1 & 3 \\ 3 & 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 \\ 1 \\ 5 \end{bmatrix} \times [2 \;\; 1 \;\; 3] = \begin{bmatrix} 6 & 3 & 9 \\ 2 & 1 & 3 \\ 10 & 5 & 15 \end{bmatrix} \xrightarrow{\text{sum}} =$$

$$\begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \times [3 \;\; 3 \;\; 1] = \begin{bmatrix} 6 & 6 & 2 \\ 3 & 3 & 1 \\ 6 & 6 & 2 \end{bmatrix}$$

Matrix ⇒ row-row formulation

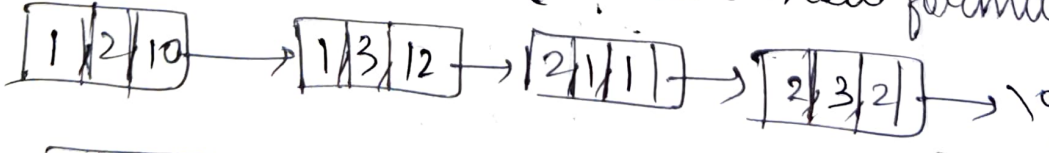
$$\underset{A}{\begin{bmatrix} 3 & 2 & 0 \\ 1 & 1 & 2 \end{bmatrix}} \times \underset{B}{\begin{bmatrix} 2 & 1 & 3 \\ 3 & 3 & 1 \\ 2 & 2 & 1 \end{bmatrix}} = \underset{C}{\begin{bmatrix} 12 & 9 & 11 \\ 9 & 8 & 6 \end{bmatrix}}$$

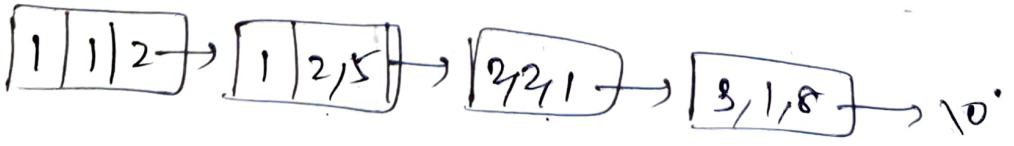
$$3 \times [2 \;\; 1 \;\; 3] + 2 \times [3 \;\; 3 \;\; 1] = [12 \;\; 9 \;\; 11]$$

$$1 \times [2 \;\; 1 \;\; 3] + 1 \times$$

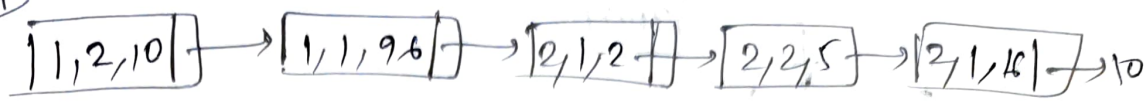
$(1 \times 1) \times (1 \times 2) \longrightarrow (1, 2)$

(column-row formulation)

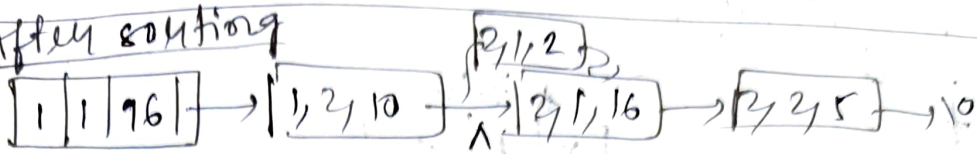


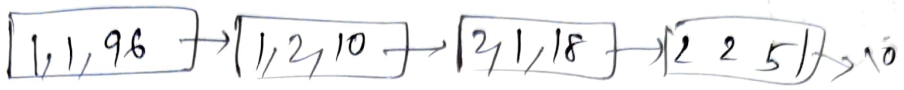


Result



After sorting

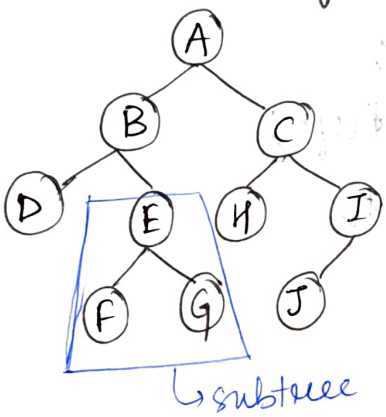




→ If both row & col^n match add the 3rd term.

$$T(n) = n^2 + n\log n + n^2$$

TREES :→ used in classification (machine learning ~~algorithm~~)



{ height → leaf to root
depth → root to leaf
no. of edges

for B, height = 2
depth = 1

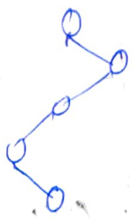
⇒ Full binary tree ⇒ 0 or 2 child

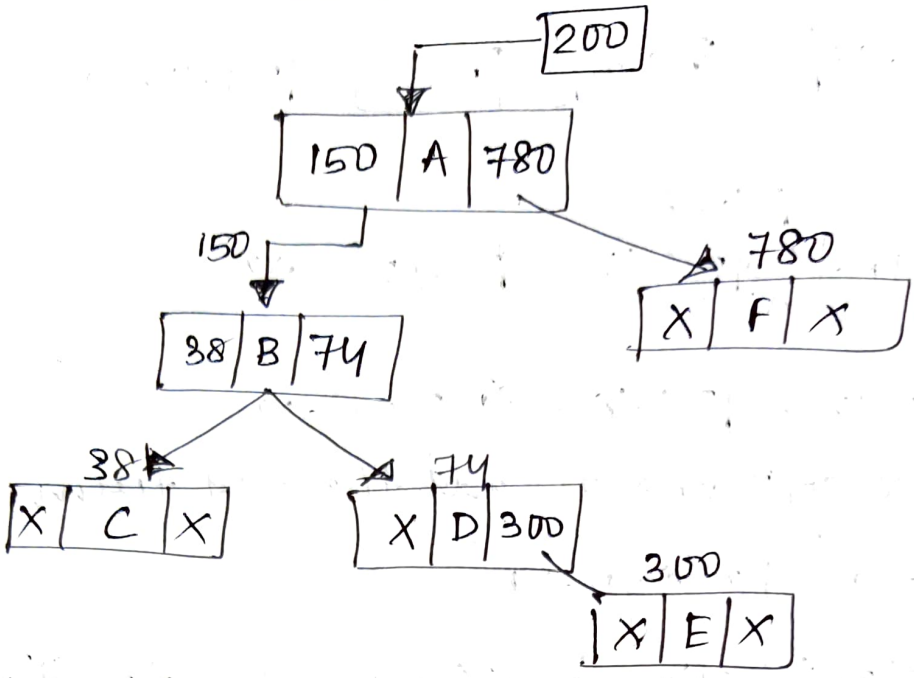
⇒ complete binary tree ⇒ all levels are completely filled except the last level → last level ↑ to ←

⇒ Balanced binary tree ⇒ ex: k balanced binary trees
⇒ if k=1 then AVL tree or Red black tree

Degenerate binary tree : every parent node has one child node

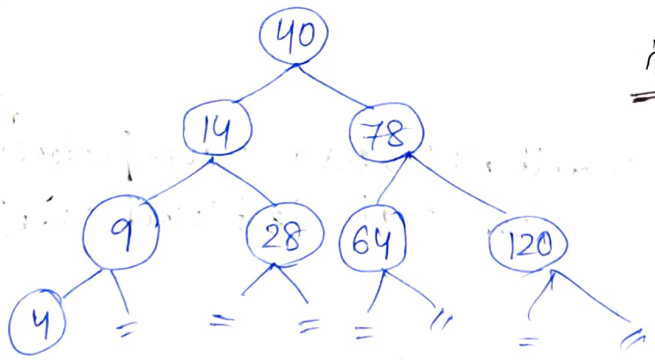
Examples:






```
struct BT {
    int data;
    BT* left;
    BT* right;
};
```

Binary search tree :- (BSTs)



invariance: left node should be smaller than the node & the right node should be greater than the node

	Insert	Delete	search
unsorted array	O(1)	O(n)	O(n)
linked list	O(1)	O(n)	O(n)
sorted array	O(n)	O(n)	O(logn)
BST	O(logn)	O(logn)	O(logn)

```
Findmin (node *root)
{
    if (root == NULL)
        return -1;
    while (root->left != NULL)
    { root = root->left;
    }
    return root->data;
}
```

→ preorder, postorder, inorder traversals

```
preorder (node *root)
{
    if (root == NULL)
    {
    }
    cout << root->data;
    preorder (root->left);
    preorder (root->right);
}
```

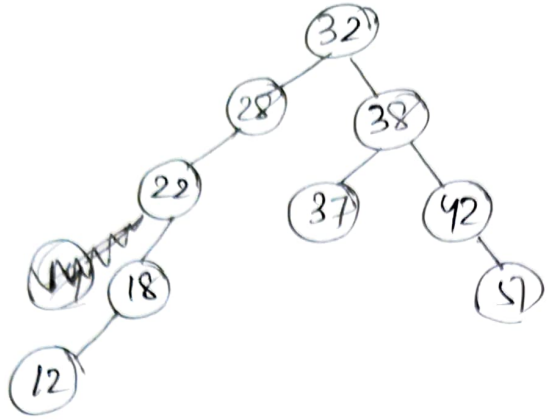
inorder gives sorted order

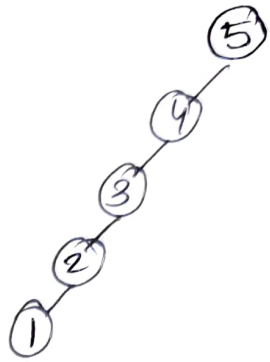
Deletion in BST :- → case 1: it is a leaf node

→ case 2: has two child then we need to replace — the max^m of left subtree (OR) — the min^m of right subtree

⇒ BFS (level order traversal) — using queue

Q) 32 28 22 38 42 51 18 37 12





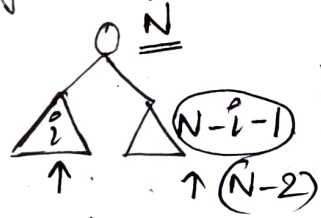
AVL tree ⟩ ① search invariant
② height balanced

BST ⟩

	Avg. case
Addition —	log(n)
Deletion —	log(n)
Search —	log(n)

$D(0)$
$D(1)$
$D(2) = \sum_{i=0}^{2}$

 # proof for avg. O(log n) in BST :-



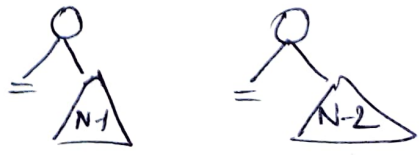
Let, $D(N) = \sum_{i=1}^{N} d(i)$ = sum of depth of nodes in a tree

all possible configuration of subtree

Also, $D(N) = \underbrace{D(i)}_{\text{left subtree}} + \underbrace{D(N-i-1)}_{\text{right subtree}} + (N-1)$ → left & right nodes increase depth by 1 because of root node

avg of sum of depth of nodes in a tree → $D'(i) = \frac{1}{N}\sum_{j=0}^{N-1} D(j)$ → we consider all values of i are equally likely.

avg → $D'(N-i-1) = \frac{1}{N}\sum_{j=0}^{N-1} D(j)$



$$D(N) = \frac{2}{N}\sum_{0}^{N-1} D(j) + (N-1)$$

$$N\,D(N) = 2\sum_{j=0}^{N-1} D(j) + (N-1)\times N \quad \text{---(1)}$$

$$(N-1)\,D(N-1) = 2\sum_{j=0}^{N-2} D(j) + (N-1)(N-2) \quad \text{---(2)}$$

Subtracting (1) & (2)

$$N\,D(N) - (N-1)\,D(N-1) = 2D(N-1) + 2(N-1)$$

$$\boxed{N\,D(N) = (N+1)\,D(N-1) + 2(N-1)} \quad \text{---(5)}$$

divide by $N(N+1)$ both sides

$$\boxed{\frac{D(N)}{N+1} = \frac{D(N-1)}{N} + \frac{2(N-1)}{N(N+1)}} \quad \text{---(6)}$$

$$\frac{D(N-1)}{N} = \frac{D(N-2)}{N-1} + \frac{2(N-2)}{(N-1)N}$$

$$\frac{D(N-2)}{N-1} = \frac{D(N-3)}{N-2} + \frac{2(N-3)}{(N-2)(N-1)}$$

$$\frac{D(N)}{N+1} = 2\sum \frac{N}{(N+1)(N+2)}$$

$$= 2\sum\left[\frac{2}{(N+2)} - \frac{1}{(N+1)}\right]$$

$$\frac{D(N)}{(N+1)} = 2\left\{\sum \frac{2}{N+2} - \sum \frac{1}{N+1}\right\} = \frac{2N}{(N+1)(N+2)}$$

$$= 2\left[2\ln N - \ln N\right] = \frac{1}{N}$$

$$D(N) = \frac{2\log(N)}{N} = 2\log N \approx O(\log N)$$

$$\sum \frac{1}{N}$$

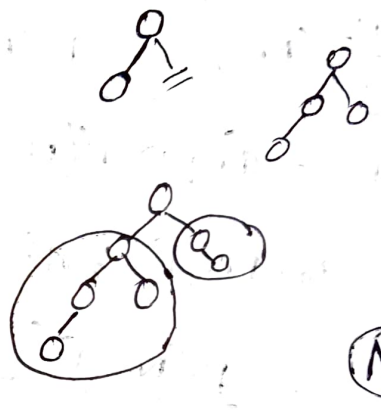
$$\int \frac{1}{N} = \ln N$$

$$\frac{D(N)}{N+1} \approx 2\log(N)$$

AVL Tree :- (Balanced BST) → worst case $O(\log N)$

Let N_h ⇒ minimum no. of nodes in an AVL tree of height h

h	N_h
0	1
1	2
2	4
3	7
4	12



$N=4$

$$N_h = N_{h-1} + N_{h-2} + 1$$

$$T(N) = T(N-1) + T(N-2) + 1$$

→ $2^{k/2}$

→ 2^k

$N_h > 2^{h/2}$ or $N_h < 2^h$

Recursion

$$\frac{h}{2} < \log_2 N_h$$

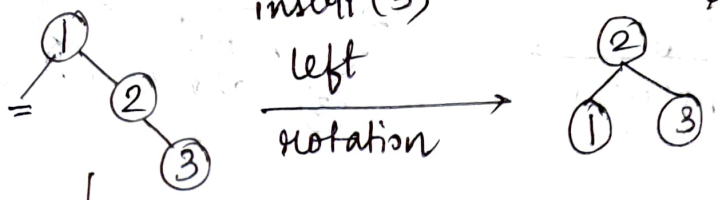
$$\Rightarrow h < 2\log_2 N_h \rightarrow \log N$$

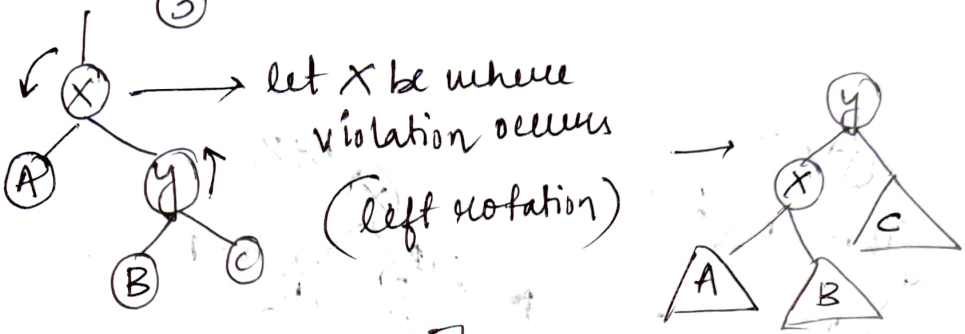
$= O(\log N)$

$$N_h > \frac{\phi^h}{\sqrt{5}}$$

$$h < 1.440 \log_2 N$$

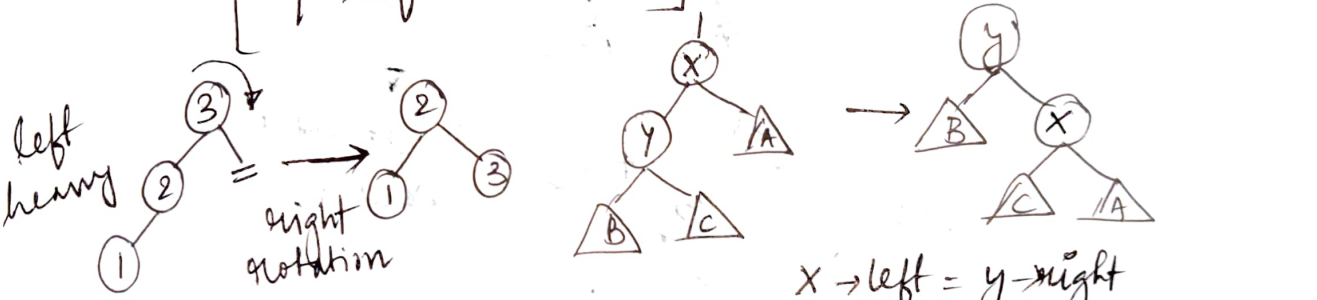
Insertion in AVL tree :-



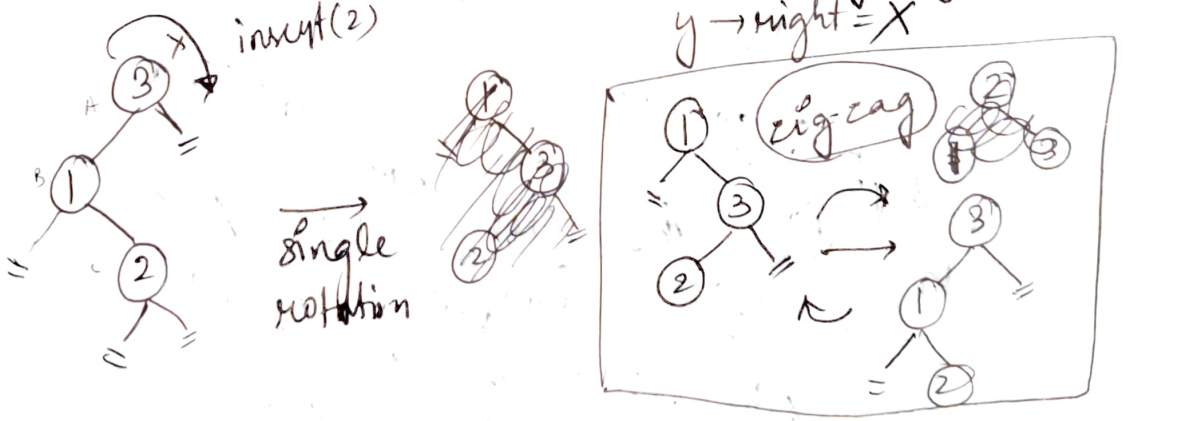


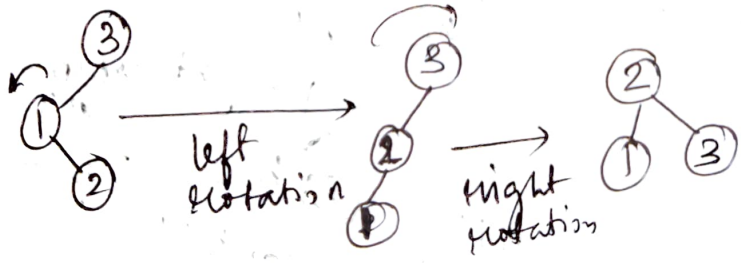
$$\left[\begin{array}{l} x \to right = y \to left \\ y \to left = x \end{array}\right]$$

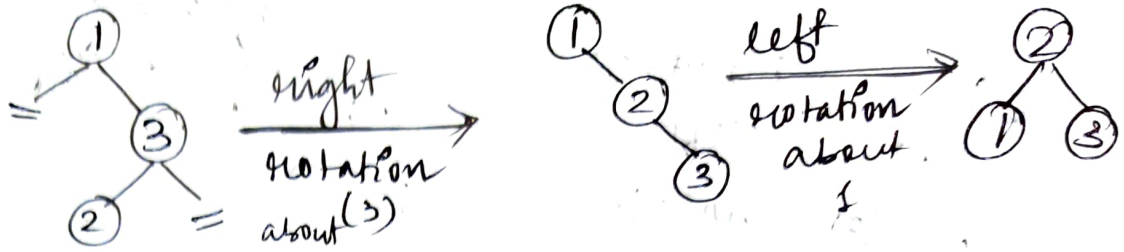
[BST property is intact]

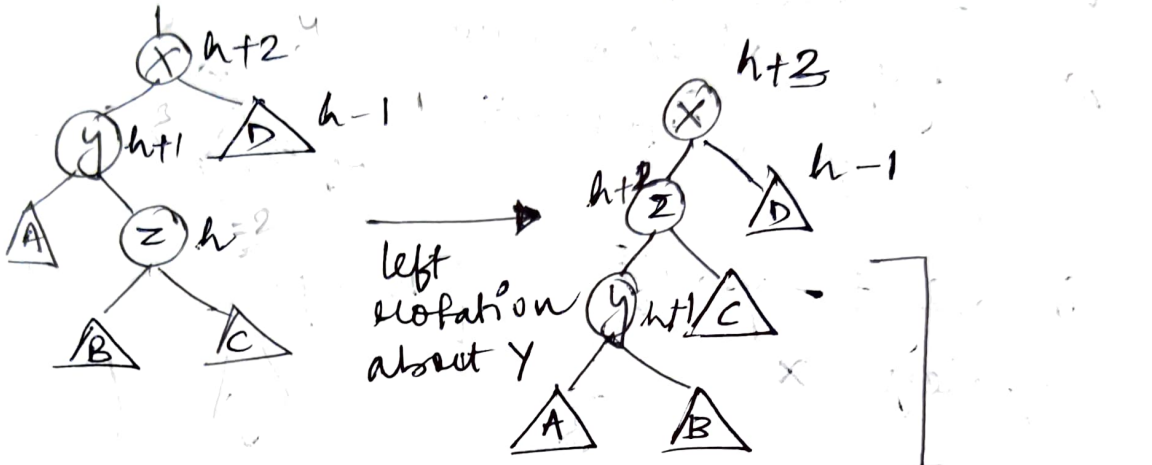


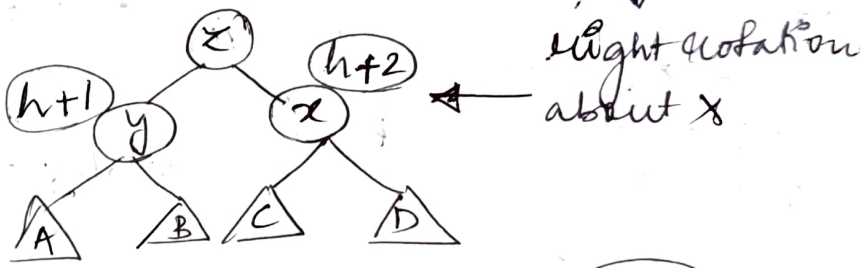
$$\left[\begin{array}{l} X \to left = y \to right \\ y \to right = X \end{array}\right]$$

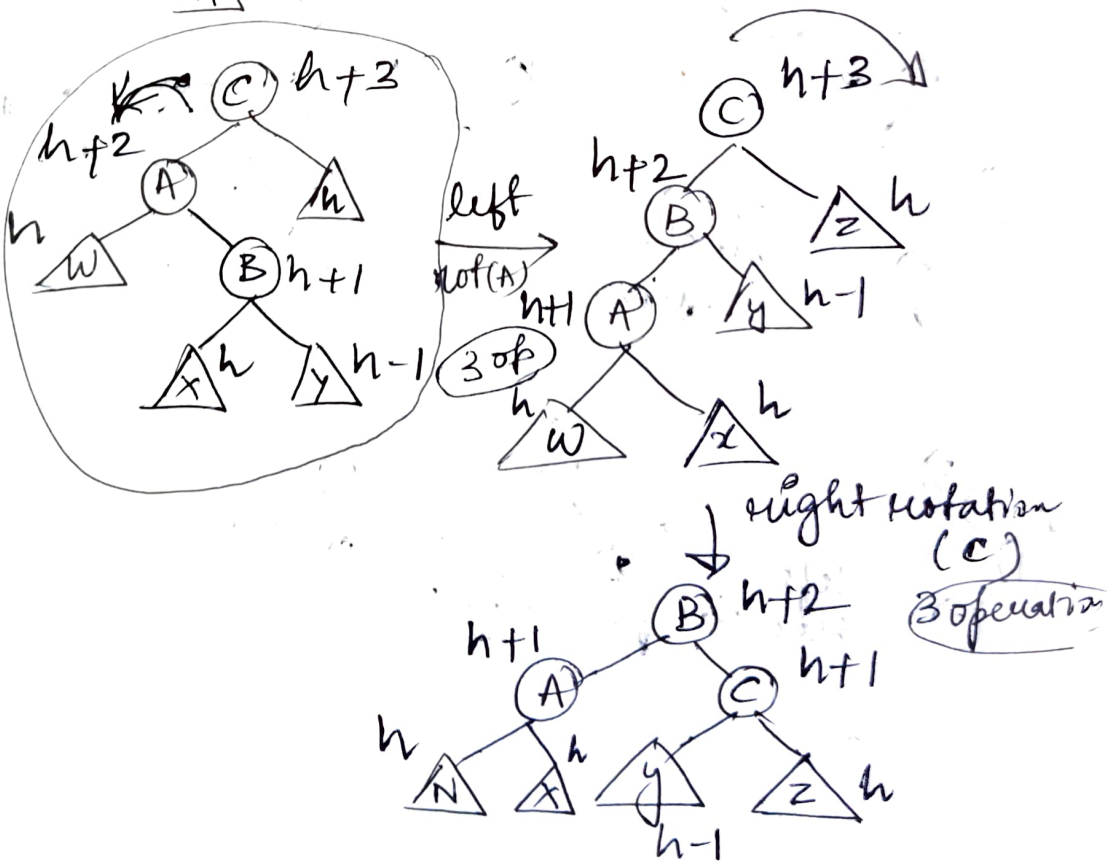




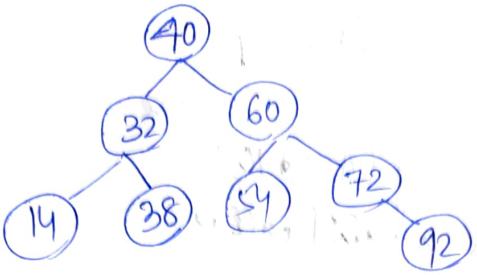

1
3
2
right
rotation
about (3)
1
2
3
left
rotation
about
1
2
1
3


x
h+2
y
h+1
D
h-1
A
z
h
B
C
left
rotation
about y
h+3
x
h+2
z
h+2
D
h-1
y
h+1
C
A
B


right rotation
about x
x
h+1
y
h+2
x
A
B
C
D


C
h+3
h+2
A
h
h
w
B
h+1
x
h
y
h-1
left
rot(A)
3 op
C
h+3
h+2
B
h
z
h+1
A
y
h-1
h
w
x
h
right rotation
(C)
3 operation
B
h+2
h+1
A
C
h+1
h
N
h
x
y
z
h
h-1

ALL Trees



left, root, right

inorder traversal

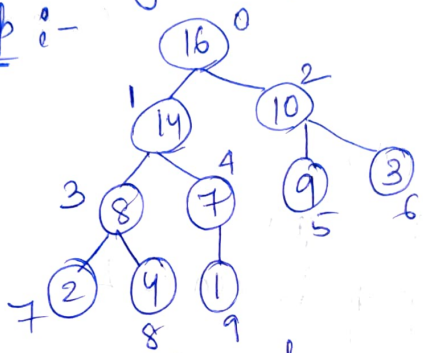
14, 32, 38, 40, 54, 60, 72, 92

AVL sort $\Rightarrow O(n \log n) + \Theta(n)$ ⇓ to traverse

HEAPS

→ Implement using arrays

→ max heap :-



c → index of child

O(1) → min/max

↳ median

0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

↳ heap

length = 10

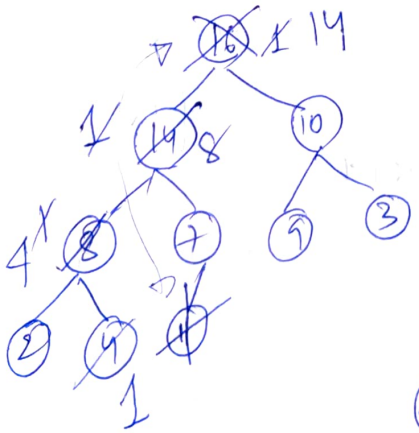
child → $2p+1$, $2p+2$ (p)

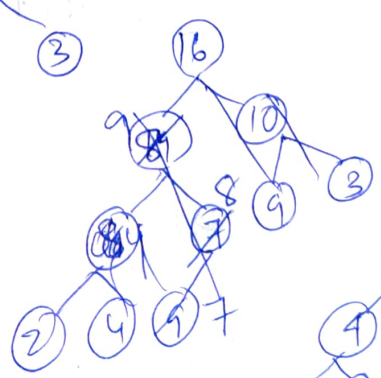
parent (c) → $\left\lfloor \frac{c-1}{2} \right\rfloor$

extract_max()

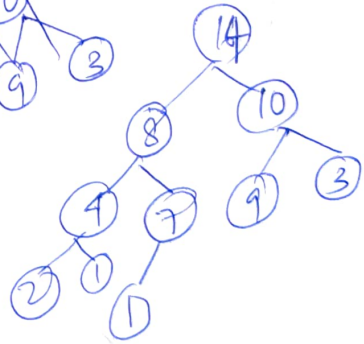
(trickle it down)

① Insert





pushing 9

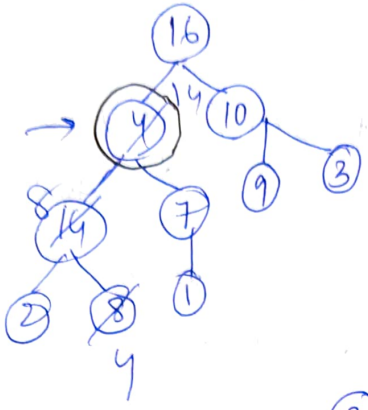


① Insert at next available position

② trickle it up

max-heapify

① called for a node
② It assumes that the left subtree & right subtree are valid heaps



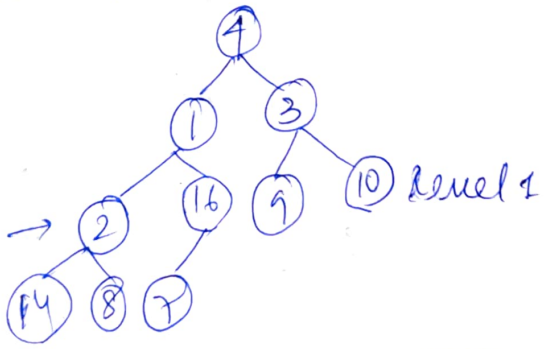
max_heapify (A, 1, 10) ← size
is it is a valid call?
index of the node

① → both children are less than the node
② → children index ≤ n (length)

Building a heap

```
Build_max_heap (A)
    A.heapsize = A.length;
from i = ⌊A.length/2⌋ down to 1
    max_heapify (A, i);
```

0	1	2	3	4	5	6	7	8	9
4	1	3	2	16	9	10	14	8	7



$\frac{n}{4}, \frac{n}{8}$

$\frac{n}{4}[1 \times c] + \frac{n}{8}[2 \times c] + \frac{n}{16}[3 \times c] + \cdots$

Set $n = 2^k$ → $\frac{n}{4}$ → 2^k → $n = 2^{k+2}$

$$= 2^k(1 \cdot c) + \frac{2^k}{2}(2 \cdot c) + \frac{2^k}{4}(3 \cdot c) + \cdots$$

$$= c \cdot 2^k \left[\frac{1}{2^0} + \frac{2}{2^1} + \frac{3}{2^2} + \frac{4}{2^3} + \frac{5}{2^4} \cdots \right]$$

$\leq c \cdot 2^k [4] \approx c \cdot n \cdot (4) \longrightarrow O(n)$

$$S = \frac{1}{2} + \frac{2}{2^1} + \frac{3}{2^2} + \cdots$$

$$\frac{S}{2} = \frac{1}{4} + \frac{2}{2^2} + \frac{3}{2^3} + \cdots$$

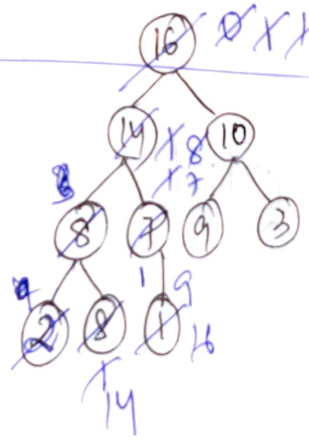
$\therefore S - \frac{S}{2} = \frac{1}{4} + 2$

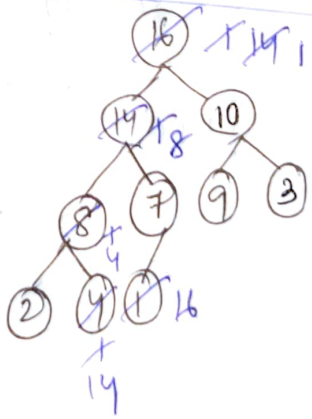
→ to prove (prove yourself)

Heap construction – O(n)

Heapsort

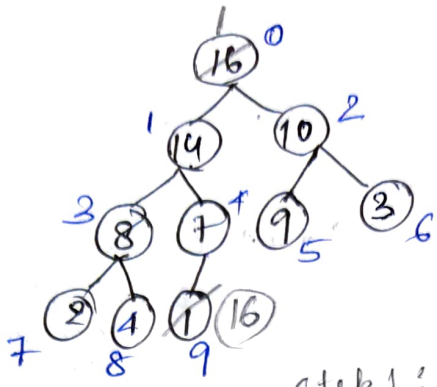
1. Build max heap
2. find max (extract_max()) A[0]
3. swap A(n-1) with A[0]
   ↳ now the max element is at the end of the array.
4. discard (n) → reduce length by 1
5. Step 3 swap may lead to a violation max_heapify (A, 0)





11/09/23

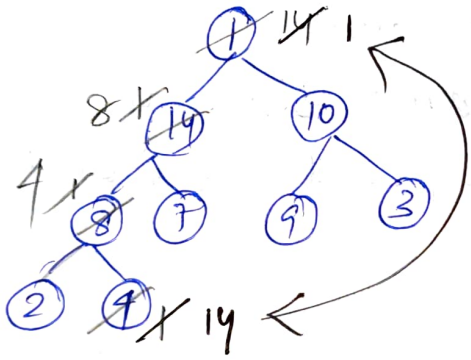
16, 14, 10, 8, 7, 9, 3, 2, 4, 1



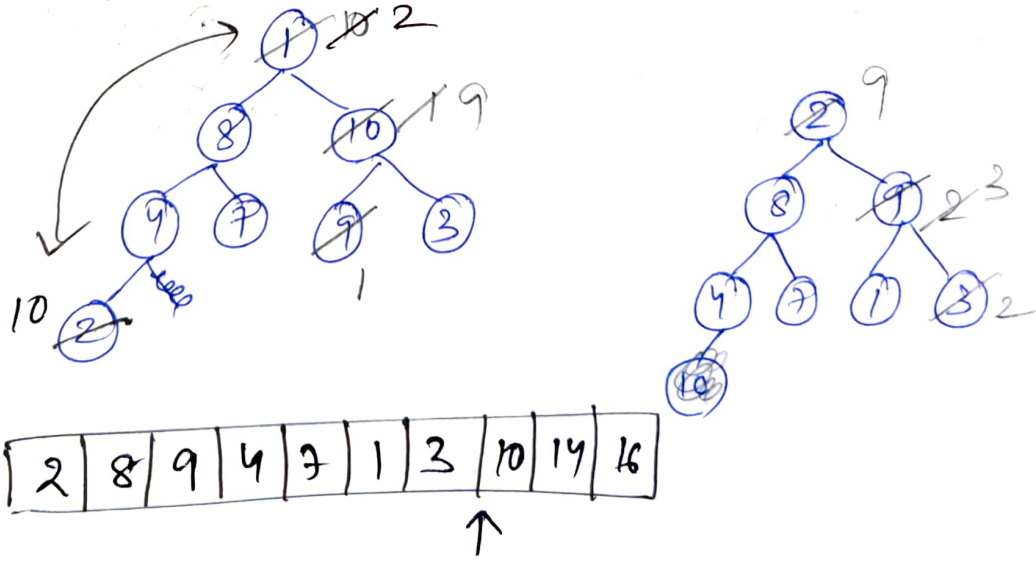
step 1 : swap 1 & 16

~~step 2 : reduce by~~ 1 step 2 : reduce the length by 1

1	14	10	8	7	9	3	2	4	16



1	8	10	4	7	9	3	2	14	16



2	8	9	4	7	1	3	10	14	16

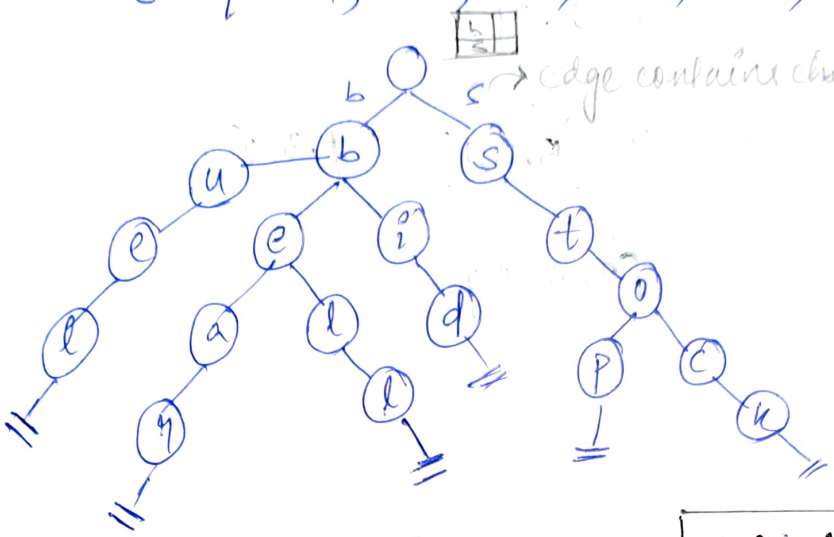
~~Soon~~ similarly,

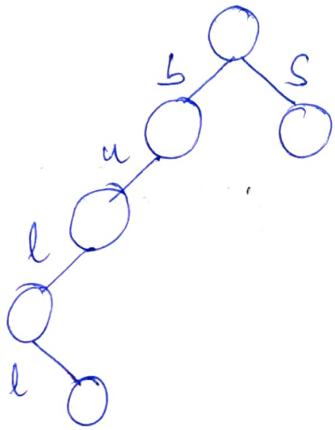
=> Mergesort requires extra space that is not the case here

=> to get the k^{th} smallest element → efficient

String Matching :- (TRIE)

S = { bear, bell, bid, bull, stock, stop }





```
node {
    boolean eos;
    array str[26];
}
```

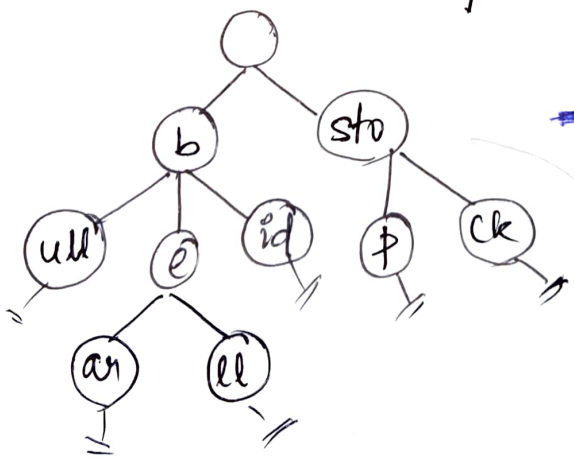
(for each node)

⇒ if length of string → P → search time = O(P)

⇒ SC = O(26 × n)

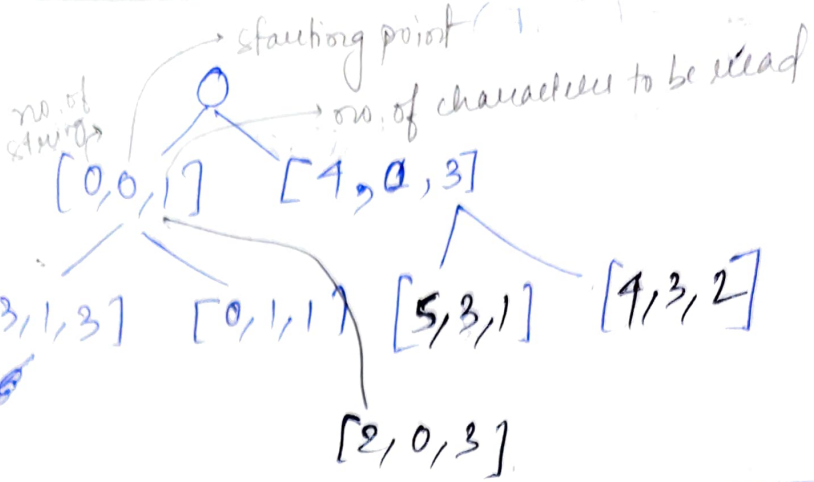
→ String matching – genome sequences

```
node {
    boolean eos;
    Dictionary (c, b);
}
```



→ compressed tries

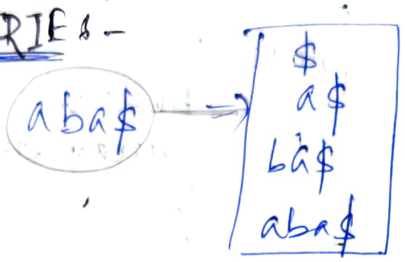
If we add steel we need to bifurcate this node.



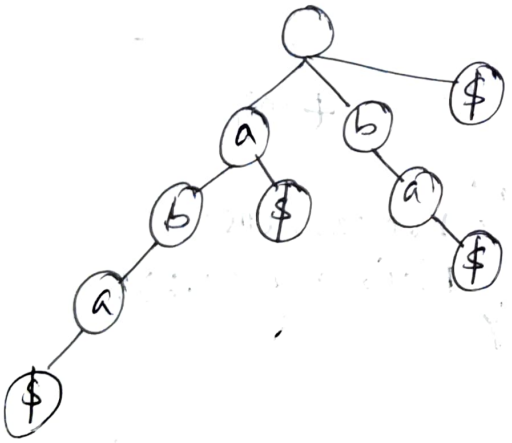
⇒ Substring ??

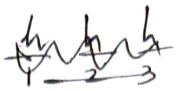
⇒ Suffix ??

Suffix TRIEs -



aaaabbbbb





1	2	3	4
5	6	7	8
9	10	11	12

1	2	3	4
1	3	6	10
6	8	10	12
15	18	18	20

1 2 3 4 5 6 7 8 9 10

1 1 2 1 2 2 3

4,2 A A B A B B A B

7

4,2,1 / 3

8

4,4 / A B

8

(1) (2)

1 2 3 4 5 6 7 8 9 10 11 12

A A B A B B A A B A

3

(2,1)

4 5→4,1 / A B

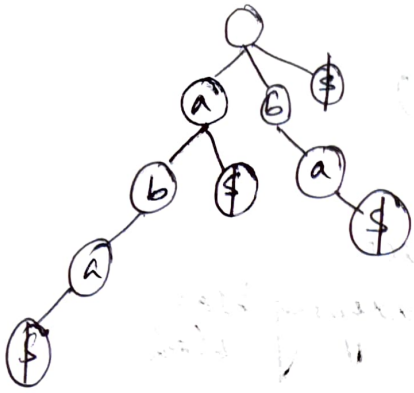
6→4,2 / A B

7→4,2,1 / A B A

8→4,2,2 / A B A

Suffix trees :-

aba$



abaaba$

⇒ 1 root

⇒ m nodes with incoming a edge

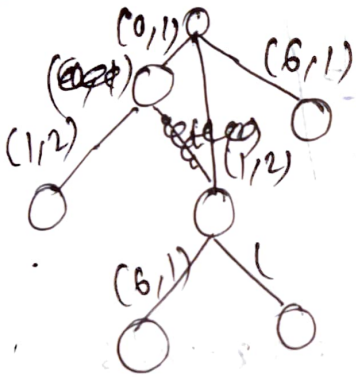
⇒ (m+1) nodes

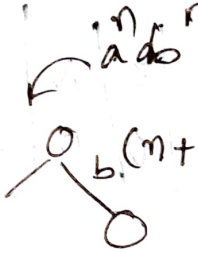
Q) $a^n b^n \rightarrow O(n^2)$

(suffix tree for this example)

Ullman - suffix tree can be constructed in $O(\_\_)$ $O(n)$ or something (not clear)

suffix tree ⇒ (starting point, length of the string)

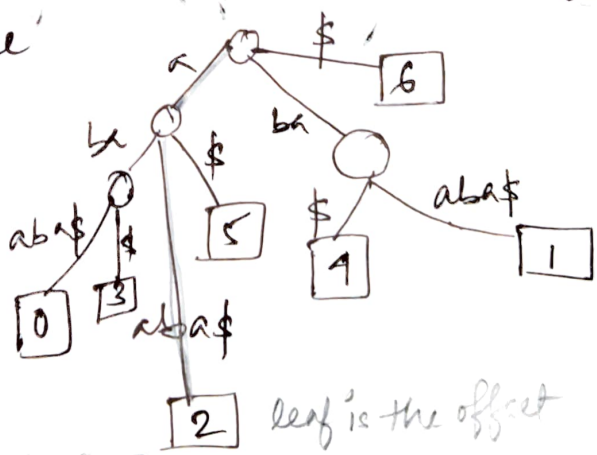




Conversion ⇒ suffix tree

abaaba$

0 1 2 3 4 5 6
abaaba$



To find the longest common substring of X & Y

⇒ find the deepest node (XY) to get the longest common subsequence

a a a a a b $O(m \times n)$

aab $BC = O(n)$

searching in

This is an algorithm class. We are studying that algorithm

alg → search

KMP String matching

$BC = O(\frac{n}{m})$ $WC = O(n)$

$AC = O(n)$

x	y	z	a	a	a	x	y	z	b
0	0	0	0	0	0	1	2	3	0

→ prefix table

10 - 3 - 1 = (6)

→ x y z a a a x y z b q q r s x y z a a a x y z b s s

x y z a a a x y z b

x y z a a a x y z b

no comparison of the 1st 3 characters as the prefix value of (z = 3)

x y z a a a x y z b

shift by 5 - 3 - 1 = (1)

a a a a a a a a a a b

a	a	a	a	b
0	1	2	3	0

→ WC

John Hopkins University
Ben Langmead

Boyer Moore :-

0 1 2 3 4
T O O T H

→ for the last character include length to the array

T	O	H	*
3	3	5	5

1 2

len - index - 1

5 - 0 - 1 =

5 - 1 - 1 = 3

BAD MATCH INDEX

T	O	H	*
1	2	5	5

move by 5 · move by 2 · move by 1

T R U S T H A R D T O O T H B R U S H E S

T O O T H

T O O T H

T O O T H

T O O T H

T O O T H

T O O T H

T O O T H

(1 after match)

T O O T H

When there is a match only move by the first character value

Text :- DOES THE TEXT CORRE (Write the code)

BC = O(n/m)

WC = O(n)

0 1 2 3

C O R R

C	O	R	*
3	2	1	4

4 - 2 - 1 = 1

→ as R has occurred before

Text:

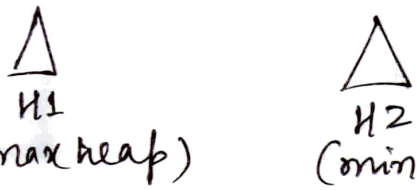
S	T	R	I	N	G	*
5	4	3	2	1	6	6

0 1 2 3 4 5
STRING

length - index - 1

DOES THE TEXT CORRECTLY MATCHES THE STRING PATTERN

STRING
STRING
STRING
STRING
STRING
STRING
STRING



[Median calculation using heaps]

ML

Bag of words (K-nearest neighbour)

each doc → bag of words (5000 D -)

As the dataset grows KNN is not efficient (takes a lot of time)

$TC = O(n \times d)$