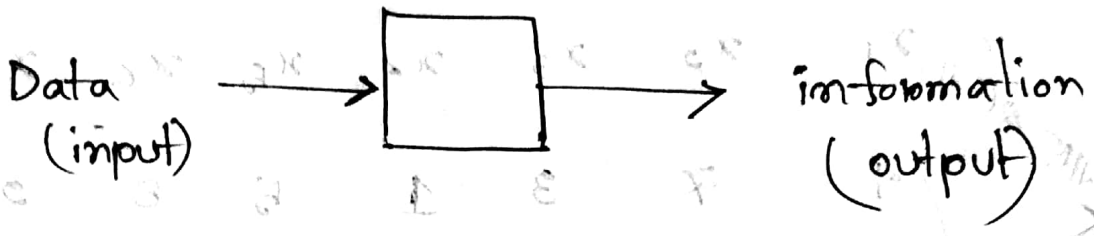
Chapter - 1

Lec - 1



Define data structure with an example:

⇒ formal definition:

1) A set of function definitions :-
2) A storage structure :- . .
3) A set of algorithms :- . . .

Lec-2

Chapter-8

Bubble Sort → O(n) / O(1)
↳ আগানো থাকলে 1 বার

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
	4	7	3	1	5	8	2	6
	4	3	7	1	5	8	2	6
	4	3	1	7	5	8	2	6
	4	3	1	5	7	8	2	6
	4	3	1	5	7	8	2	6
	4	3	1	5	7	2	8	6
pass 1:	4	3	1	5	7	2	6	8
pass 2:	3	1	4	5	2	6	7	8
pass 3:	1	3	4	2	5	6	7	8
pass 4:	1	3	2	4	5	6	7	8
pass 5:	1	2	3	4	5	6	7	8
pass 6:	1	2	3	4	5	6	7	8
pass 7:	1	2	3	4	5	6	7	8

Algorithm: $k \leftarrow n$

while $\{k \neq 0\}$

$t \leftarrow 0$

for $j = 1$ to $k-1$ do ($j=0$, $k-2$)

if $x_j > x_{j+1}$ then (comparison)

$x_j \leftrightarrow x_{j+1}$ (array ডাটা move করে)

$temp = x/x_j$

$x_j/x = y/x_{j+1}$

$x_{j+1}/y = temp$

$t \leftarrow j$

$k \leftarrow t$

$n-i$ (i no ডাটা ততবার move করবে যতবার left -এ বড় ডাটা থাকবে)

d_j: Number of data larger than x_j and to its left

d_6 6 নং এর আগে যত ডাটা আছে তার চেয়ে বড় ডাটা বাকী আছে

$d_6 = 0$ (কেননা বড় নাই)

$d_7 = 5$ (5 টা ডাটা বাকী এর চেয়ে বড়) → swapping এর আগে

$d_1 = 0$ (always)

Lec-3

Total no of data movement:

$d_1 = 0$

$d_1 + d_2$ (2 no আইটেম এর জন্য movement)

$d_1 + d_2 + d_3$ (3 " " ")

total no of data swap → $d_1 + d_2 + d_3 + \ldots\ldots + d_n = 3\sum_{j=2}^{n} d_j$ — ①

($d_1 = 0$)

$\therefore d_2 + d_3 + d_4 + \ldots\ldots + d_n = 3\sum_{j=2}^{n} d_j$ — ①

3 টা লাইন এর জন্য

1) Best Case: $d_j = 0$

যত বার move হবে, সবগুলো sorted-ই থাকবে

sorted আছে, so বারো বার কেটে নাই

Putting $d_j = 0$ in eqn ① we get;

$$d_2 + d_3 + \ldots + d_n = 3\sum_{j=2}^{n} 0$$

$$= 0$$

2) Worst Case: $d_j = j - 1$

বড় থেকে ছোট সাজালে আসবে

Putting $d_j = j - 1$ in eqn ① we get;

$$d_2 + d_3 + d_4 + \ldots\ldots + d_n = 3\sum_{j=2}^{n} (j-1)$$

$$= 3\{(2-1) + (3-1) + (4-1) + \cdots (n-1)\}$$

$$= 3\{1 + 2 + 3 + \cdots (n-1)\}$$

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

3) Avg. Case: $dj = \frac{j-1}{2}$

$$= \sum_{j=2}^{n} \frac{j-1}{2}$$

$$= 3 \left\{ \frac{(2-1) + (3-1) + \cdots (n-1)}{2} \right\}$$

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

$$= \frac{3n(n-1)}{4}$$

Lec-4

Number of Companison's In Bubble Sort:

1) While loop: largest dj + 1

সবার বামে সবচেয়ে বেশী আছে তাকে সাজালে আর পরে automatic সাজানো হবে। তার চেয়ে 1 বেশী হবে। সাজানোর সময় সবসময় check করতে হয়।

* Best Case: 0 + 1 (dj = 0)
= 1

For loop: n - 1 (k - 1 পর্যন্ত for loop ঘুরে
k = n
so n - 1)

* Worst Case: n - 1 + 1 ← largest dj
n

For loop: (n-1) + (n-2) + . . . + 1 ← শেষ কম হবে না।

= n(n-1)/2

ques
Write: n(n-1)/2
Calculate:
← যেভাবে চেয়েছে

* Avg. কেস Case: ✗

Insertion Sort

cards, bus এর contuctor এর কাছে tk

1st ধরে কোন comp করে, 2nd এ temp এ রাখে → পরের ঘরে যাবে

	x_0	x_1	x_2	x_3	x_4	x_5
	$-\infty$	8	7	2	4	6
Pass 1:	$-\infty$	7	8	2	4	6
Pass 2:	$-\infty$	2	7	8	4	6
Pass 3:	$-\infty$	2	4	7	8	6
Pass 4:	$-\infty$	2	4	6	7	8

Lec-5

Total Number of Data Comparison:

$$(d_2+1)+(d_3+1)+\dots\dots+(d_n+1)$$

2nd data

$$= d_2+1+d_3+1+\dots+d_n+1$$

$$= 1+1+\dots\dots+d_2+d_3+\dots+d_n$$

(n-1) সংখ্যক 1

$$= (n-1)+\sum_{j=2}^{n} d_j \quad \text{—— (i)}$$

Total number of data movement:

$$(n-1)+\sum_{j=2}^{n} d_j+(n-1)$$

temp-এ প্রথমে রাখার কাজ

$\sum_{j=2}^{n}$ প্রত্যেক ডাটার জন্য

d_j move হবে

d_j+1 না, কেননা compare হলেও move হবেনা

(n-1) temp থেকে আবার রাখার জন্য

$$= (2n-2)+\sum_{j=2}^{n} d_j \quad \text{—— (ii)}$$

1) Best case: $dj = 0$

Comparison: ① $\Rightarrow$

$(n-1)$

movement: ⑪ $\Rightarrow$

$(2n-2)$

2) Worst Case: $dj = j-1$

Comparison: ① $\Rightarrow (n-1) + \sum_{j=2}^{n} (j-1)$

$= (n-1) + (2-1) + (3-1) + \cdots \quad (n-1)$

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

$= \frac{n^2 - n + 2n - 2}{n}$

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

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

Movement: $(2n-2) + \frac{n(n-1)}{2}$

$= \frac{4n - 4 + n^2 - n}{2}$

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

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

3) Avg. Case: $d_j = \frac{j-1}{2}$

Comparison: $(n-1) + \sum_{j=2}^{n} d_j$

$$= (n-1) + \sum_{j=2}^{n} \frac{j-1}{2}$$

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

$$= (n-1) + \frac{n^2-n}{4}$$

$$= \frac{4n-4+n^2-n}{4}$$

$$= \frac{4(n-1) + n(n-1)}{4}$$

$$= \frac{(n-1)(n+4)}{4}$$

Movement: $(3n-2) + \frac{1}{2} \times \frac{n(n-1)}{2}$

$$= (3n-2) + \frac{n^2-n}{4}$$

$$= \frac{8n-8+n^2-n}{4}$$

$$= \frac{8(n-1) + n(n-1)}{4}$$

(Bubble Sort + Insertion Sort → imp for exam)

$$= \frac{(n-1)(n+8)}{4}$$

Algorithm: $x_0 \leftarrow -\infty$

for $j = 2$ to n

$t \leftarrow x_j$

$i \leftarrow j-1$

While $t < x_i$

$x_{i+1} \leftarrow x_i$

$i \leftarrow i-1$

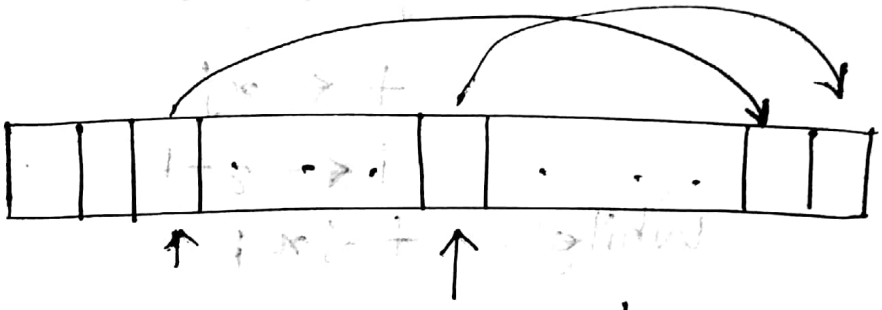
$x_{i+1} \leftarrow t$

j এর মানের সাথে compare করে $i+1$ এ বসানো

13/5 → 1st quiz

Lec-6

Selection Sort



1) আগে select হবে কে বড়
   তার exchange, last-এ যাবে

2) এরকম চলবে

2য় element-কে প্রয়োজনে swap

	8	7	2	4	6
Pass 1:	6	7	2	4	8
Pass 2:	6	4	2	7	8
Pass 3:	2	4	6	7	8

```
for  j = n  to  2  by  -1  do
        t ← 1
        for  k = 2  to  j  do
            if  x[t] < x[k]
                t ← k
        x[t] ↔ x[j]
```

1 নং মান বড় ভাটা

কত পর্যন্ত যাবে

ডাটা movement

swap

∴ Best/Worst case → ডাটা movement

$= 3(n-1)$

Why? (3 marks)

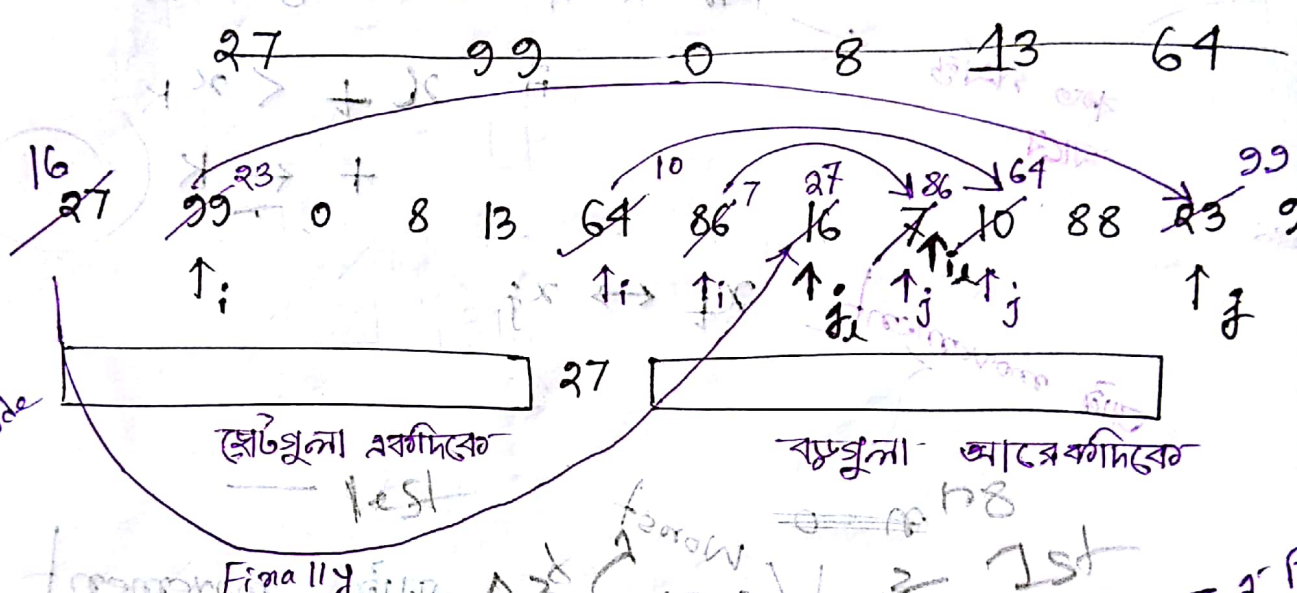
শেষ ঘরে বড় ডাটা, নিজের সাথে নিজে exchange।

প্রতিটি ঘর data move না করলেও নিজের সাথে move করবে 3 বার।

$(n-1) \rightarrow$ swap

Lec - 7

Quick Sort



Pass 1: 16 23 10 8 13 10 7 | 27 | 86 64 88 99 90

1st element → mid → ছোট গুলা left → cross করলে 16 এর সাথে swap

Pass 2: 10 7 0 8 13 | 16 | 23 | 27 | 64 | 86 | 88 99 90

3 টা ডাটা আজানো already

Pass 3: 8 7 0 | 10 | 13 | 16 | 23 | 27 | 64 | 86 | 88 90 | 99 |

Pass 4: 0 7 | 8 | 10 | 13 | 16 | 23 | 27 | 64 | 86 | 88 90 | 99 |

disadvantg ?

advantage ?

1) processor বেশী - কাজে লাগানো যাবে।

2) parallel processing.

← QuickSort (f, l)

if $f < l$ then

$i \leftarrow f+1$

while $x_i < x_f$ do $i \leftarrow i+1$

$j \leftarrow l$

while $x_j > x_f$ do $j \leftarrow j-1$

while $(i < j)$ do

$x_i \leftrightarrow x_j$

repeat $i \leftarrow i+1$ until $x_i \geq x_f$

repeat $j \leftarrow j-1$ until $x_j \leq x_f$

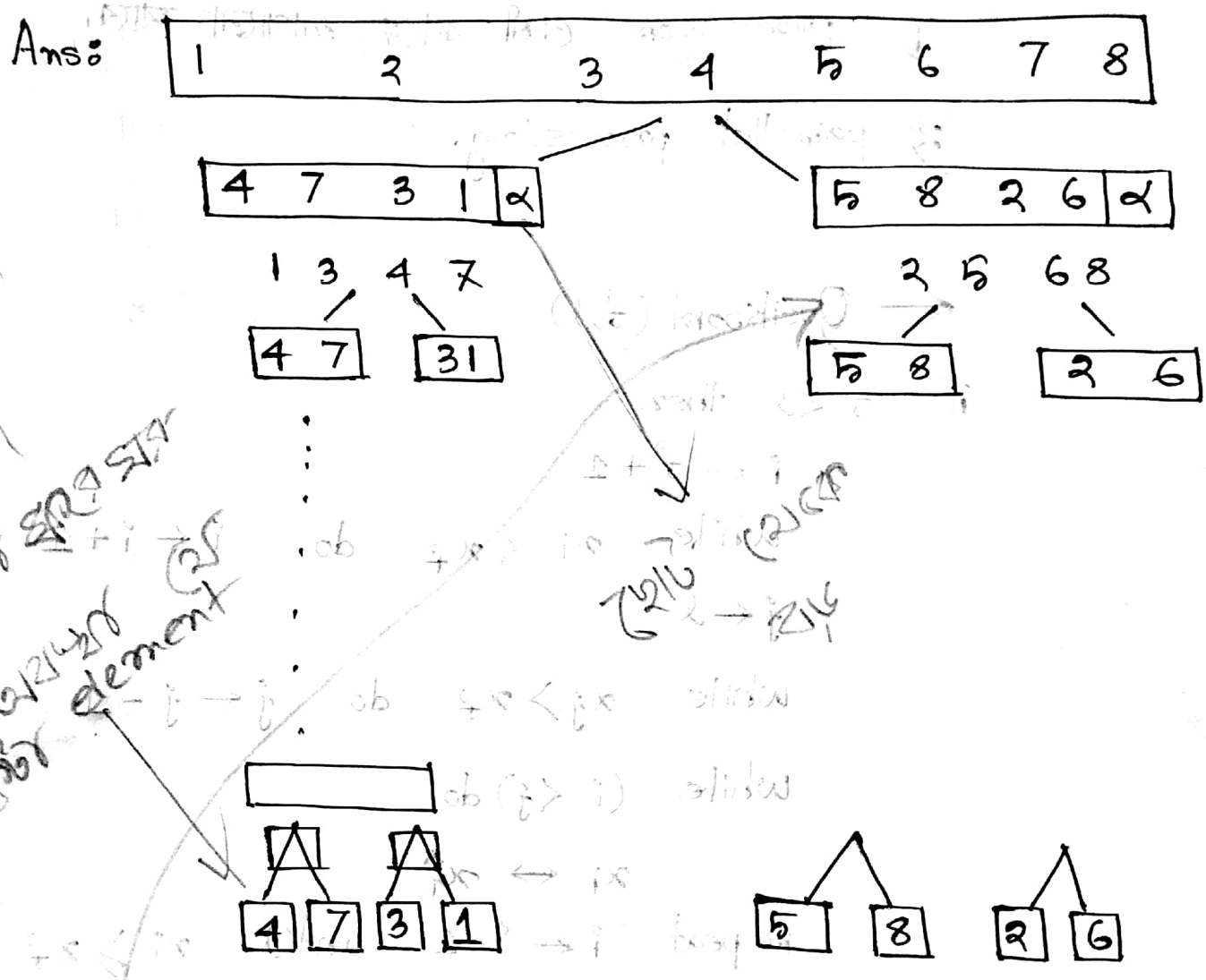
$x_j \leftrightarrow x_f$

Quicksort $(f, j-1)$

" " $(j-1, l)$

Lec-8

Merge Sort



Merge_sort (A, p, r)

1) if p < r
2) $q = \left[\frac{p+r}{2}\right]$
3) Merge (A, p, q)
4) Merge (A, q+1, r)
5) Merge (A, p, q, r)

Merge (A, p, q, r)

1) $n_1 = q - p + 1$
2) $n_2 = r - q$
3) let L [1 . . $n_1 + 1$] and R [1 . $n_2 + 1$]
4) for i = 1 to n_1
5) L [i] = A [p + i - 1]
6) for j = 1 to n_2
7) R [j] = A [q + j]
8) L [$n_1 + 1$] = ∞
9) R [$n_2 + 1$] = ∞
10) i = 1
11) j = 1

12) for k = p to h

13) if L [i] ≤ R [j]

14) A [k] = L [i]

15) i = i + 1

16) else A [k] = R [j]

17) j = j + 1

Lec-9 Radix Sort

207 095 646 108 809 376 917 534 310 209

0	– 310	207 → 108 → 809 → 209	095
1	–	310 → 917	108
2	–		207 → 209
3	–	534	310 → 376
4	– 534	646	
5	– 095		534
6	– 646 → 376		646
7	– 207 → 917	376	
8	– 108		809
9	– 809 → 209	095	917

last digit অনুযায়ী এখানে লিখবো

Pass 1: 310 534 095 646 376 207 917 108 809 209

Pass 2: 207 108 809 209 310 917 534 646 376 095

Pass 3: 095 108 207 209 310 376 534 646 809 917

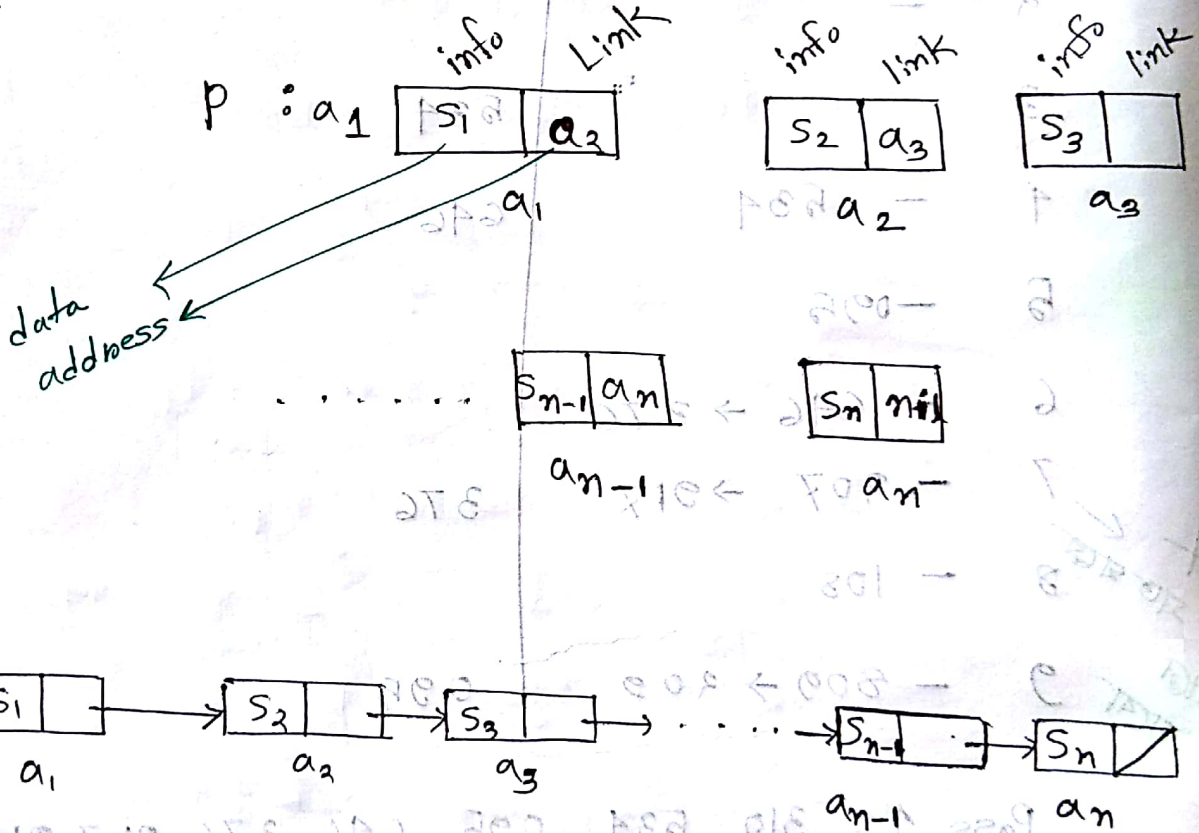
Chapter-4 থেকে
2 টা ques final-এ

L-10

Imp ~~Chapter-4~~

Exam ~~Linked List~~

int x
x = 10
x = 10.5 x
float y
Type-2 ← *p
pointer int/float *p



info ~~←~~ S2

info (a2) ← S2

address নিতে হবে

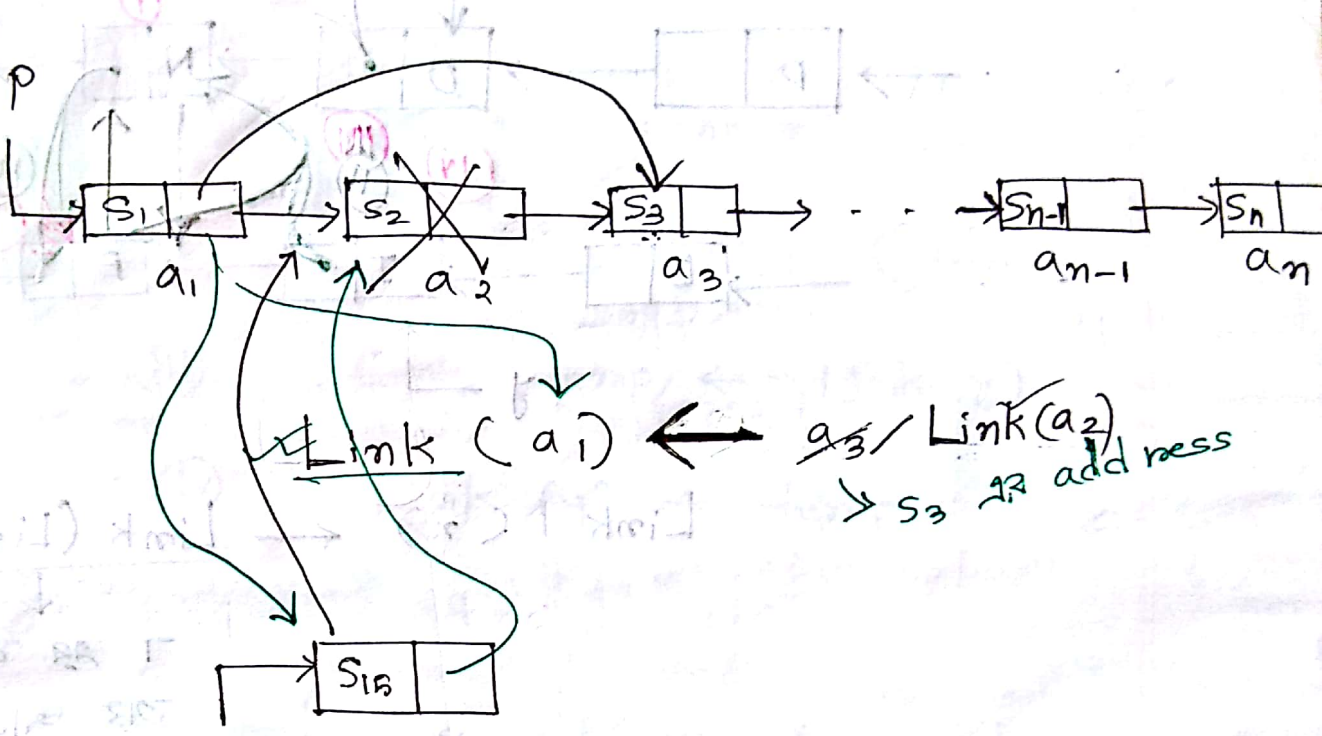
ফাইলের address আছে

ফাইলের address আছে

নীরা ← address

Link $(a_2) \leftarrow a_3$

Delete



Link (a_1) ⟸ ~~a_3~~ / Link (a_2)

$\gg S_3$ এর address

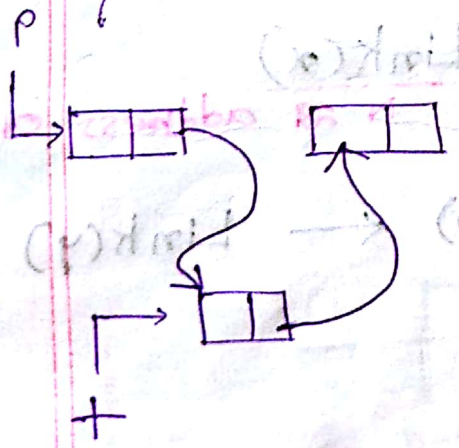
t

address ← t

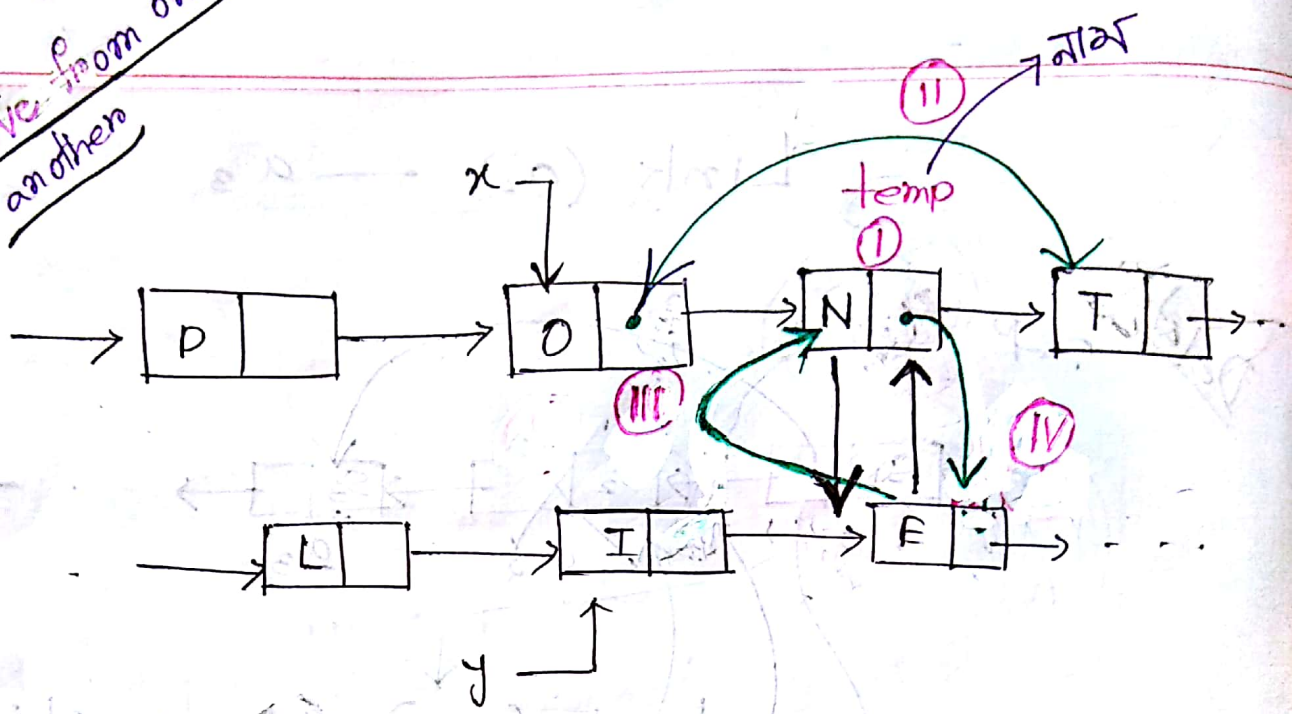
normal insert:

Link $(a_1) \leftarrow +$

Link $(+) \leftarrow a_2$ / Link (a_1)



Exam ⇒ Move from one list to another



Link (x) ← Link (Link(x))

T এর address নেই, তার আগে যেটার address দেয়া

আজাজো খুশি

(1) Link (x) ← Link (Link (x))
নাম address — T এর address

(2) Link (y) ← Link (x)
N এর address আগেরটা

(3) Link (Link (x)) ← Link (y)

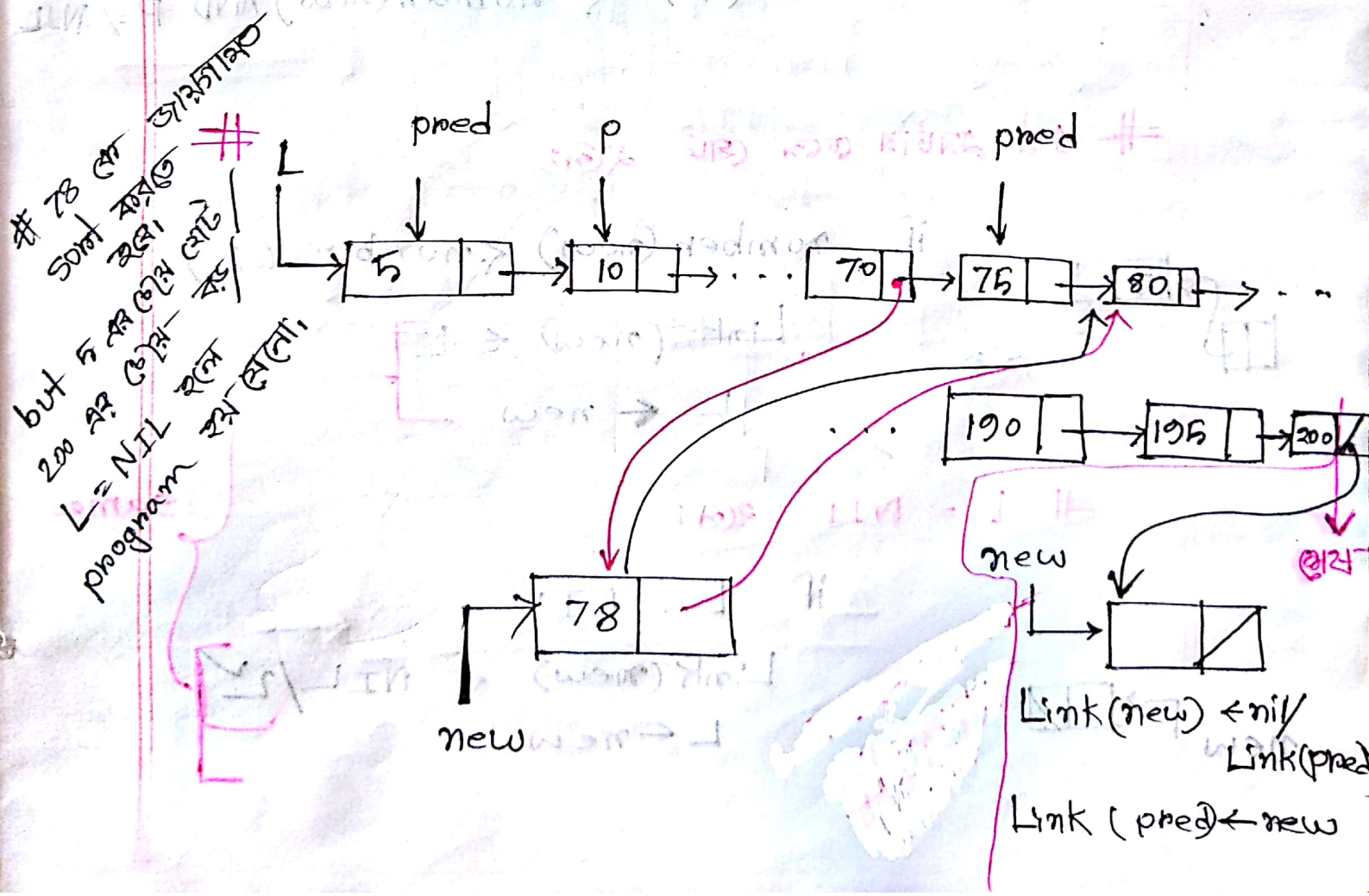
Lec-11

(i) temp ← Link(x)

(ii) Link(x) ← Link(temp)

(iii) Link(temp) ← Link(y)

(iv) Link(y) ← temp



pred কিভাবে আগের মিল হবে.

এটার node এর advantage

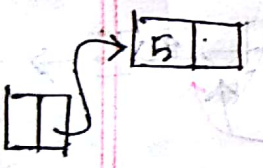
```
While number (P) < number (new)
        pred ← P
        P ← Link (P)
Link (new) ← Link (pred)
Link (pred) ← new
```

2nd এর চেয়ে বড় হলো:

```
While number (P) < number (new) AND P ≠ NIL
```

1st এরটার চেয়ে ছোট হলো:

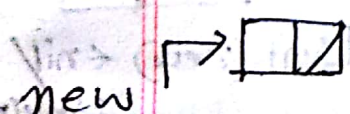
```
if number (new) < number (L)
        Link (new) ← L
        L ← new
```



L = NIL হলো

```
if L = NIL
        Link (new) ← NIL/2
        L ← new
```

same



Link List এর সুবিধা / অসুবিধা.

→ direct delete / insert, dynamic memory allocate
array - static

অসুবিধা → জায়গা বেশী; searching → prob (binary)

Full algo:

if number (new) < number (L) or L = NIL

Link (new) ← L

L ← new

else pred ← L

p ← Link (L)

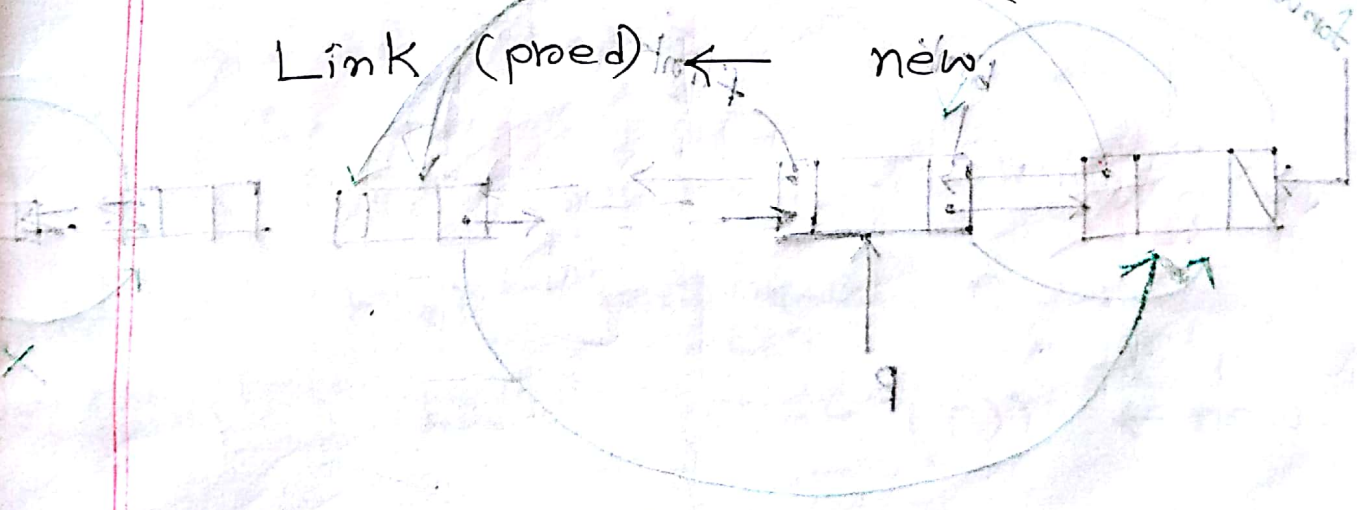
while number (P) < number (new) And P ≠ NI

pred ← p

p ← Link (P)

Link (new) ← Link (pred)

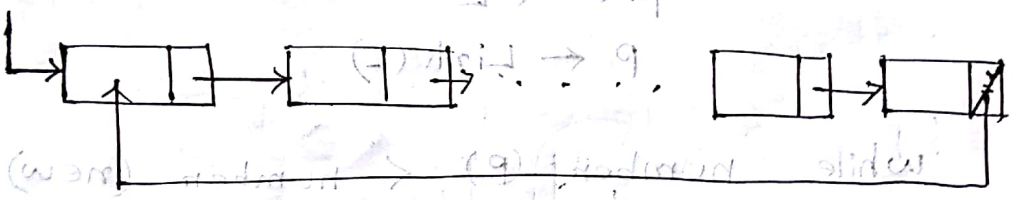
Link (pred) ← new



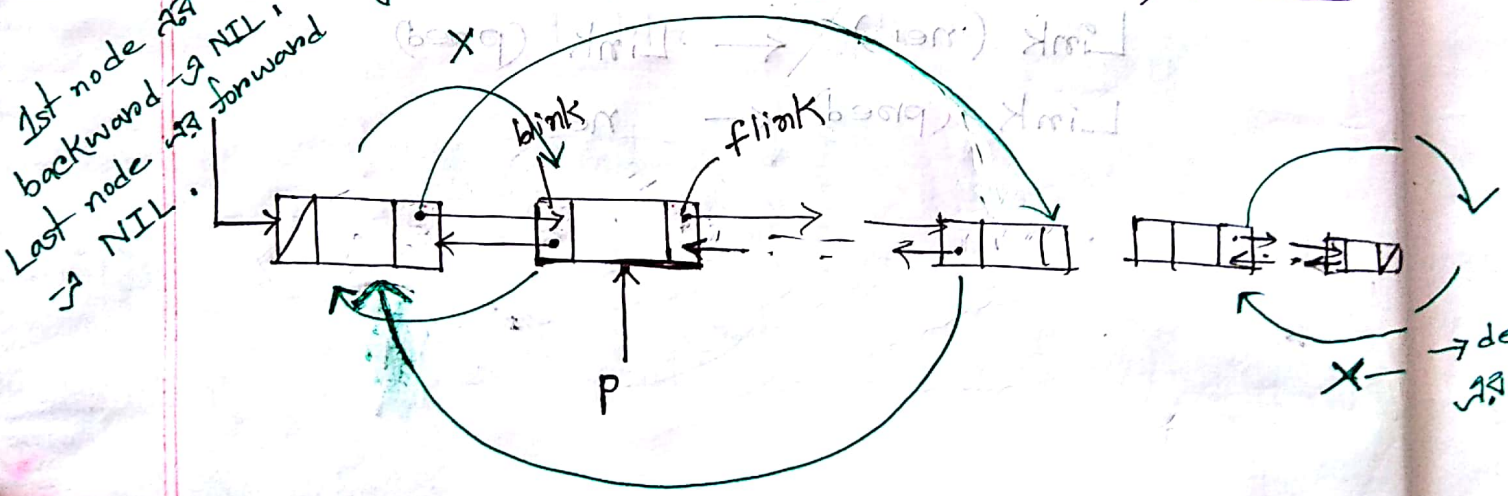
Lec-13

Common Variants Of Linked List

Circular Linked List:



Doubly Linked List:



flink (blink (p)) ⟷ flink (p)
বাম

blink (flink (p)) ← blink (p)

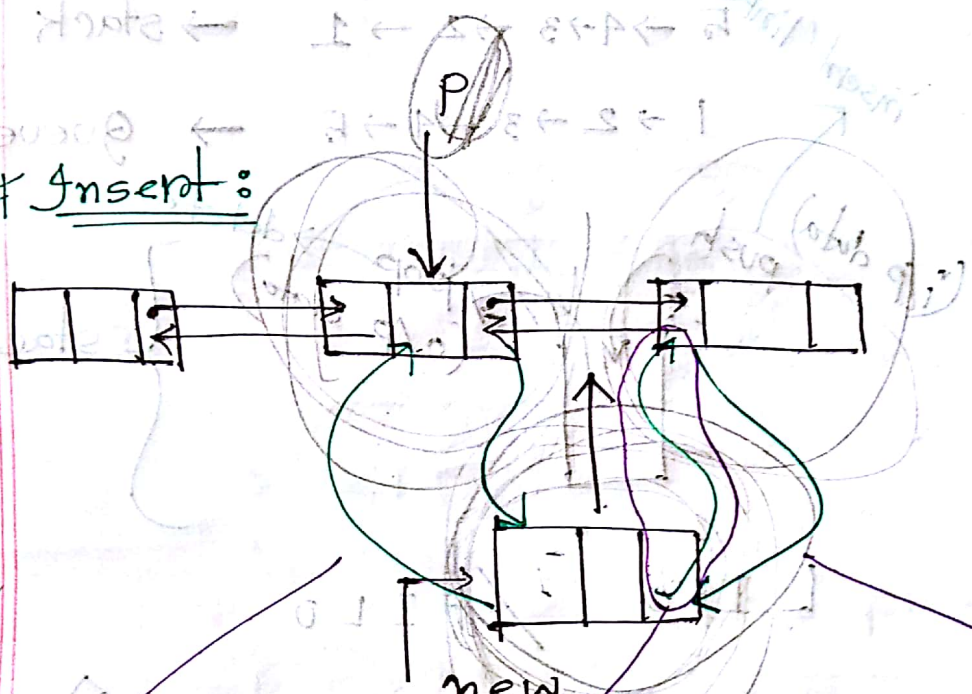
বুঝার জন্য

Delete

If blink (P) ≠ Nil then flink (blink (P)) ← flink(P)

If flink (P) ≠ Nil blink (flink (P)) ← blink(P)

Insert:



flink (new) ← flink (P)

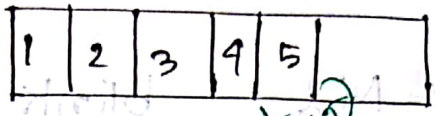
blink (new) ← P

blink (flink (P)) ← new

flink (P) → new

→ delete এর জন্য

Stack / Queue:

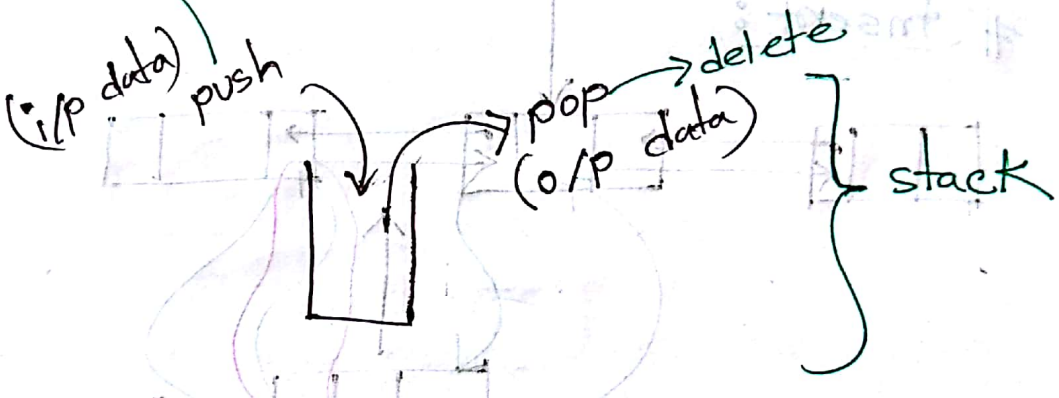


insert (linked list - 2)

5 → 4 → 3 → 2 → 1 → stack

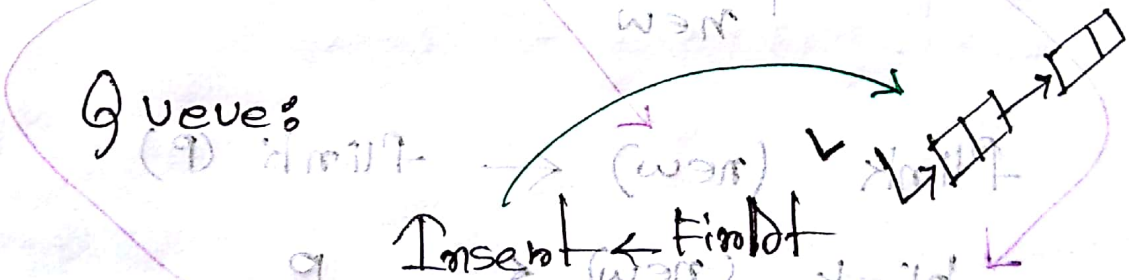
1 → 2 → 3 → 4 → 5 → Queue

Exam: push, pop

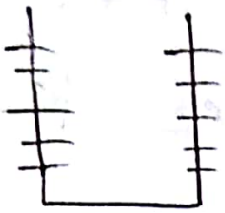


LIFO / FILO

Queue:



Lec-13



S ← empty stack

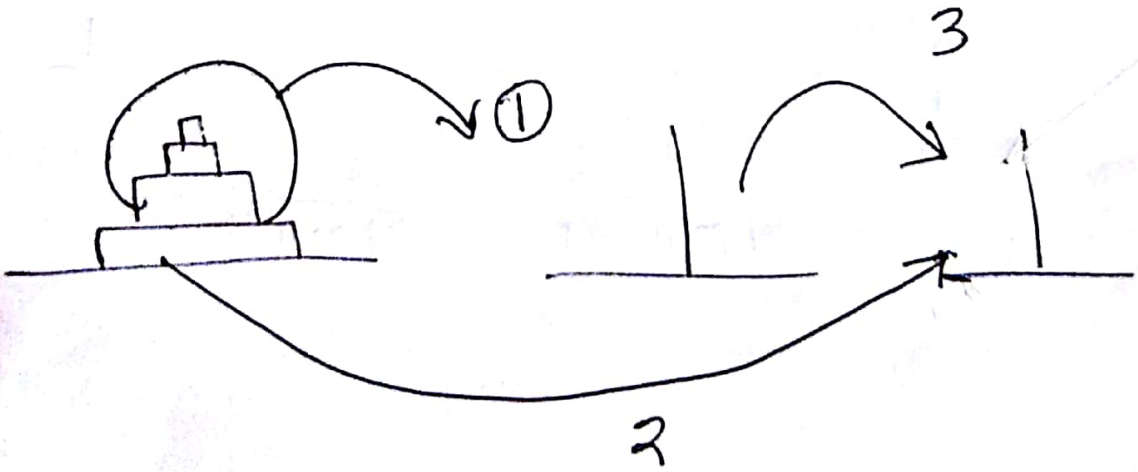
Q ← emty Queue

D ⟸ x

x ⟸ D

top (S)

Tower Hanoi



Recursive

Hanoi (n, i, j, k)

if n = 1, then move the top disk from pole i to pole k

Hanoi (n-1, i, k, j)

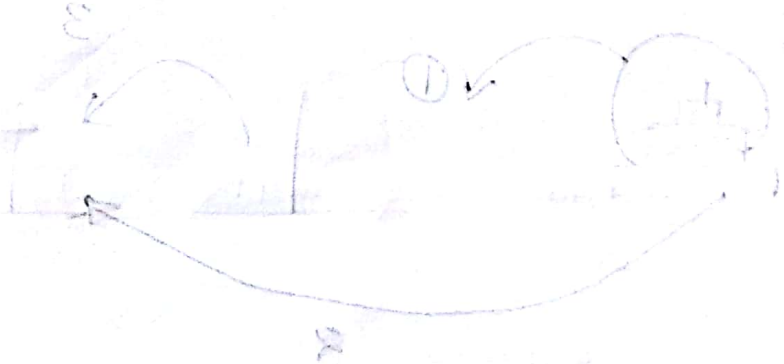
move the top disk from pole i to pole k

Hanoi (n-1, j, i, k)

যেই data আগে ঢুকাবো সেই আউট আগে হবে

↓

stack এর property



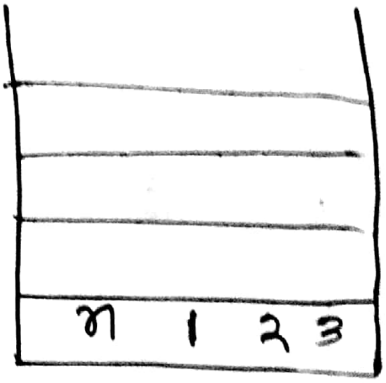
Lec - 14

Non-rec:

S ← empty stack

S ⇐ (n, 1, 2, 3)

S এর মধ্যে একটি data ঢুকে push হচ্ছে



while S ≠ empty do

(n, i, j, K) ⇐ S

pop / এর value সবগুলো variable এর

if n = 1 then move i to K

else

S ⇐ (n-1, j, i, K)

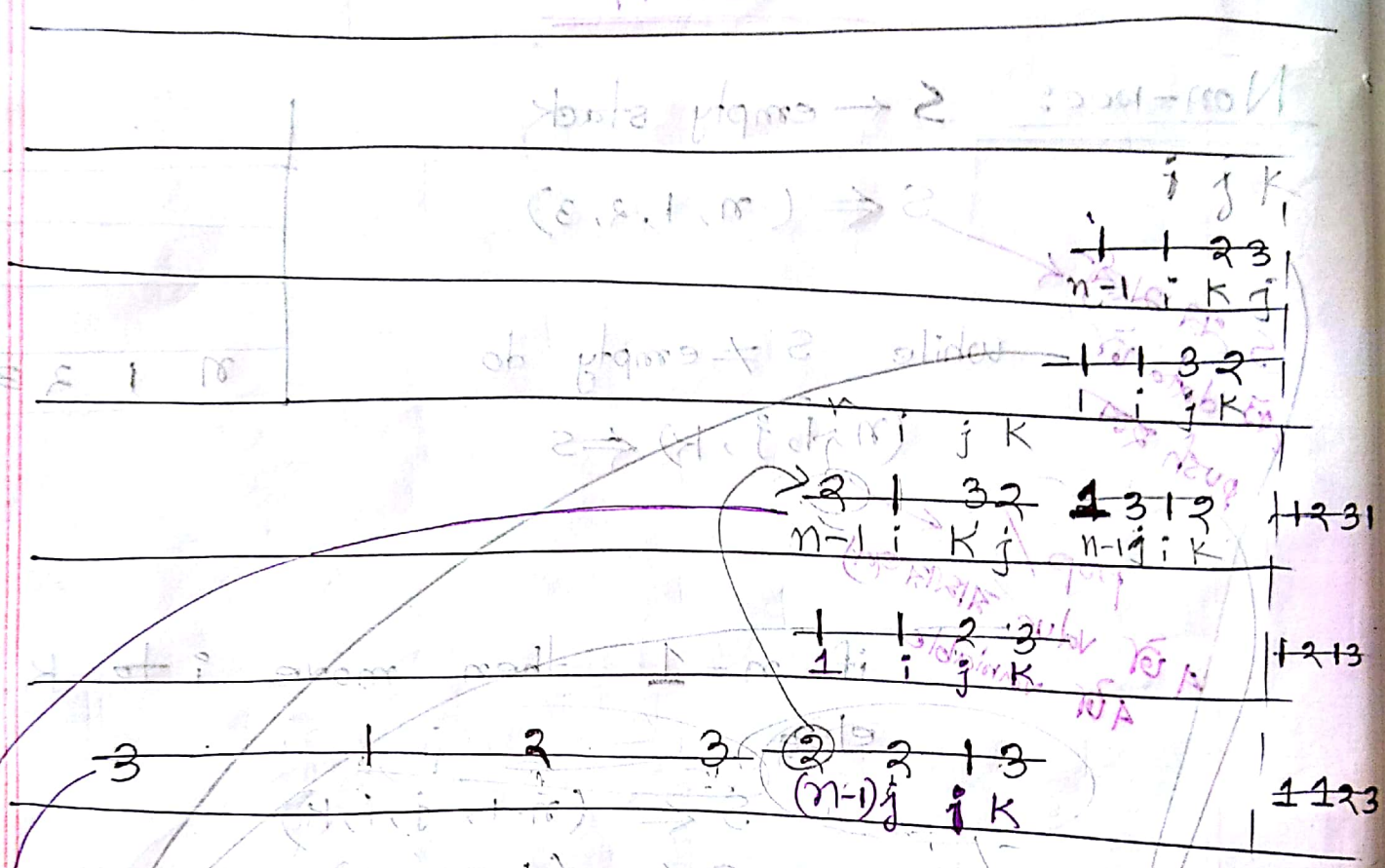
S ⇐ (1, i, j, K)

S ⇐ (n-1, i, K, j)

stack property

Algorithm/ demonstration → exam

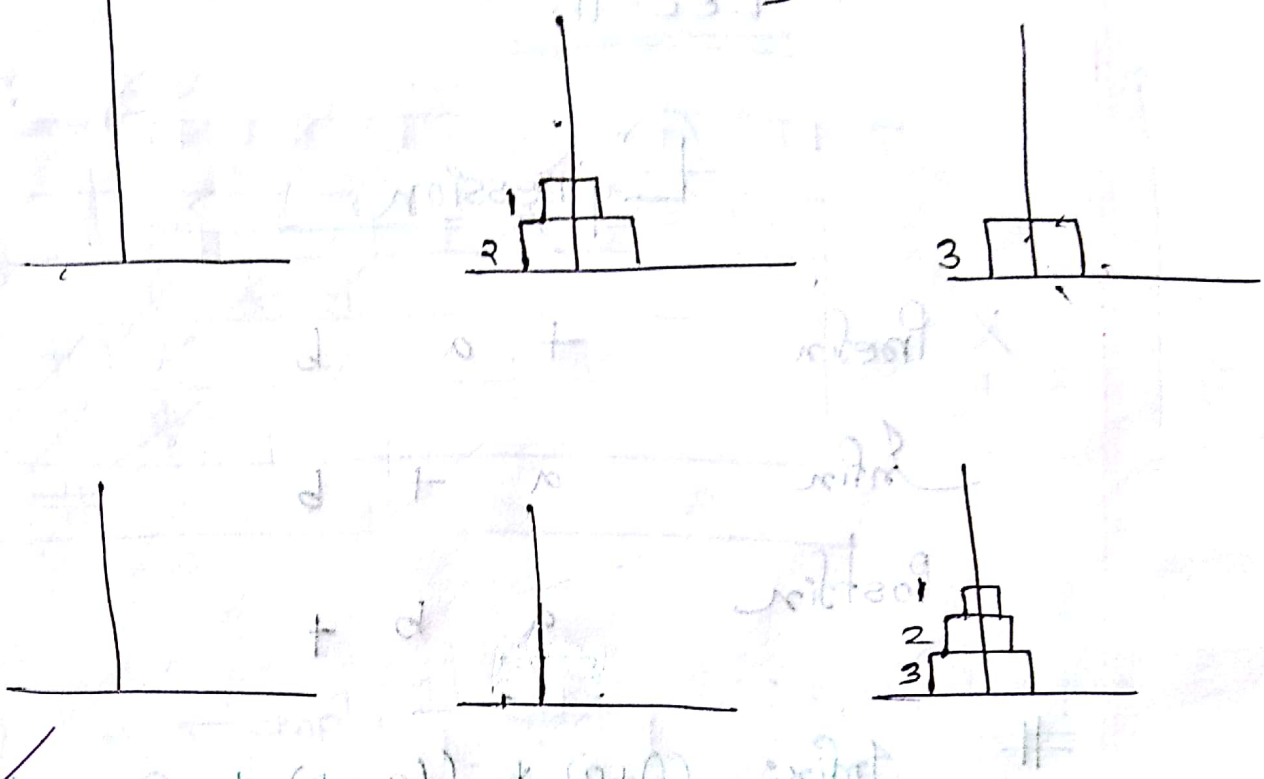
Demonstration :



n	i	j	K
3	1	2	3
2	1	3	2
1	1	2	3
1	1	3	2
1	3	1	2
1	3	2	1

ধীরে ধীরে নীচে নামাতে হবে

h	i	j	k
2	2	1	3
1	2	3	1
1	2	1	3
1	1	2	3



Final stack

Algorithm / Postfix / জানা বের করা → Exam

Lec-16

Expression

X Prefix + a b

Infix a + b

Postfix a b +

Infix: (A+B) * ((C-D) * E + F) $ = 3

opening, operand এলে direct push

Post: A B + C D - E * F + *

close এর পর যা আছে তা push

যদি বড় থাকে তবে আগের operation pop করবে recent টা push

(3) (-1)

A = 1 B = 2 C = 3 D = 4

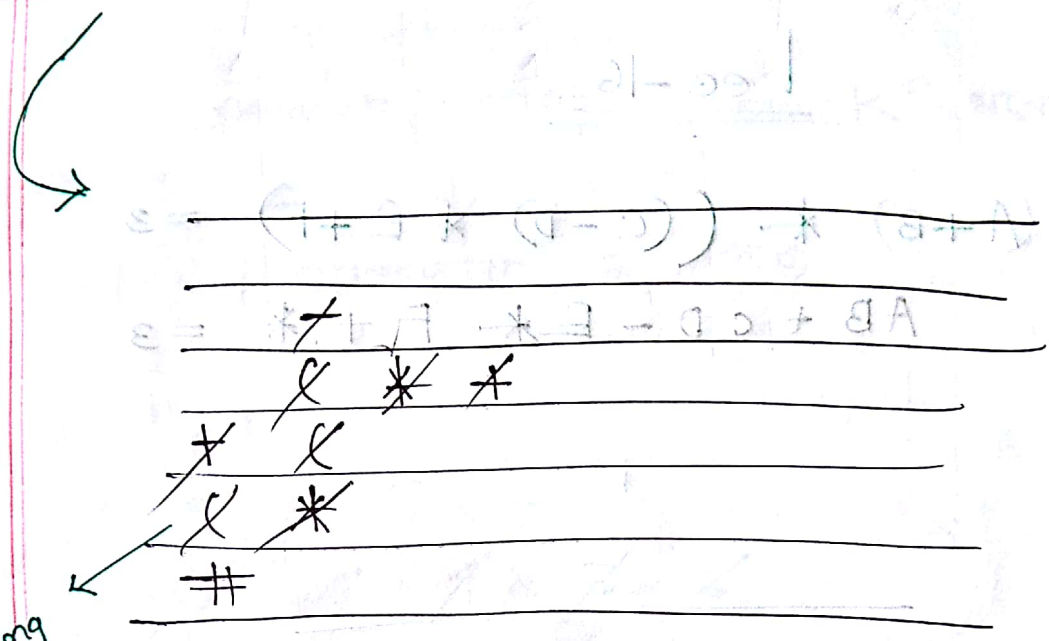
E = 5 F = 6

⇒ 3 (-1) E * F + *

∴ 3 -5 F + *

3 1 * = 3 (Ans)

exam → using stack বলতে যেমন বোঝো



opening হলে stack → push

1) S ← empty stack

2) $ add

3) # add

4) Case

closing হলে যতগুলা আগে যেগুলা pop করবো এতক্ষণ stack → opening না পাওয়া যাবে →

'(' : Stack → push

')' :

'operator' : আগে operator থাকলে সেটা post → push করে তারপর pop করে operator টা stack → push

'operand' :

L.L → Tower Hanoi

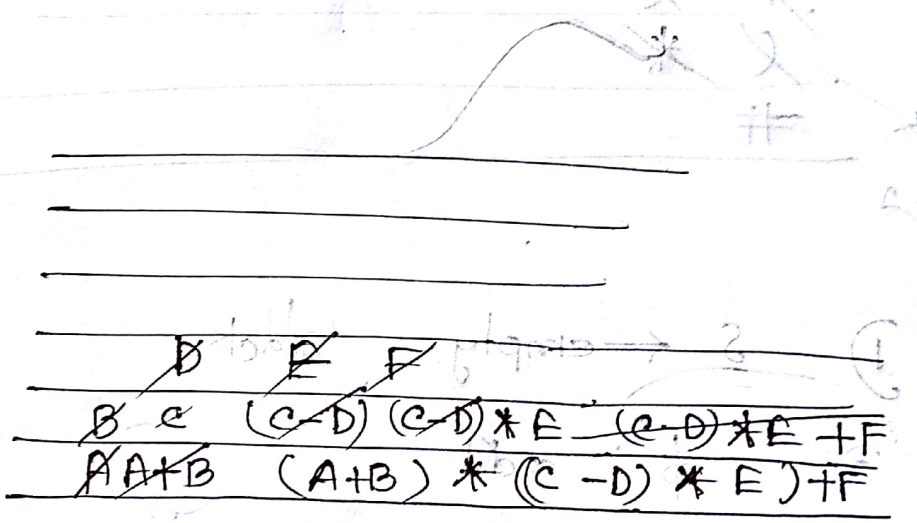
CT-02

10

Lec-16

$$(A+B) * ((C-D) * E+F) = 3$$

$$\Rightarrow \quad AB + cD - E * F + * = 3$$



~~D E F~~

~~B C (C-D) (C-D)*E (C-D)*E+F~~

~~A A+B~~ (A+B) * ((C-D) * E)+F

post থেকে infix এ convert

Algo:

stack

↓

loop তৈরি

↓

চলতে চলতে ২ ধরণের

↓

operator & operand

↓

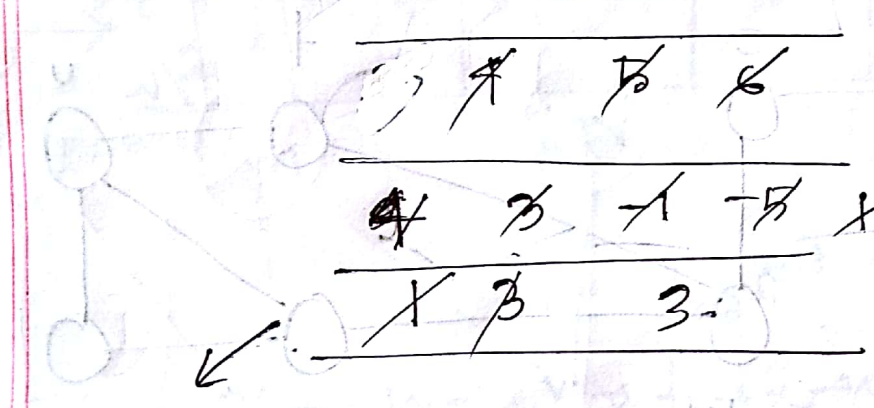
operand গেলে stack-এ push

↓

operator পেলে stack থেকে pop

↓

operator এর কাজ



Value দেয়ার প্রয়োজন

✓ Quiz (2)

search ⇒ sequence (start-end)

Algo / 3 no ques

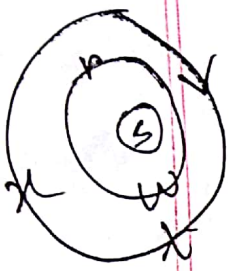
Final

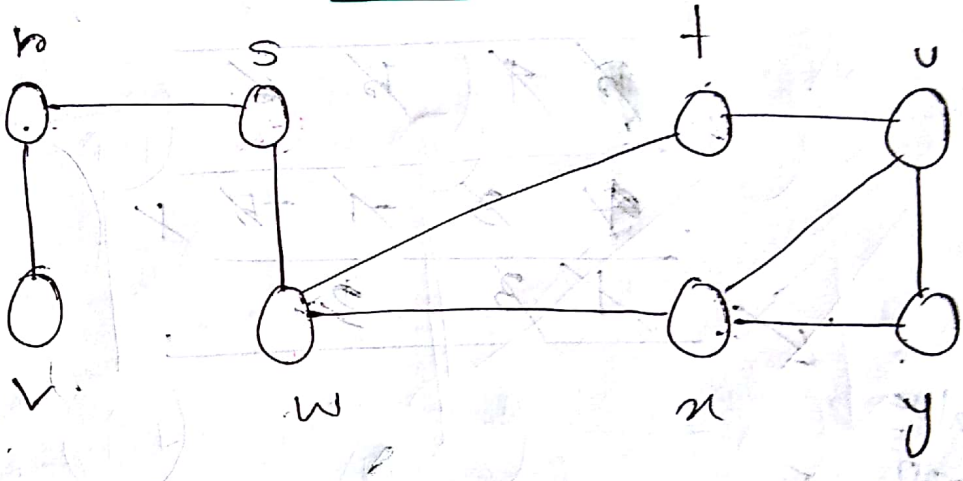
Lec - 17

Breadth First Search (BFS)

Depth First Search (DFS)

BFS





connect

Way: 1 ‖ $s \to r \to w \to v \to t \to x \to u \to y$

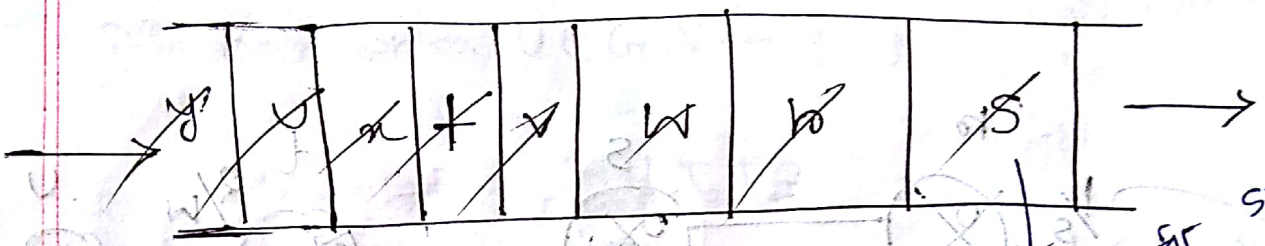
$\to x \to t$

Way: 2 ‖ $s \to w \to r \to t \to x \to v$

$\to x \to \to v$

এভাবে লিখবো:

Queue



S → h → W → V → + → x → U → y

Q তৈরি → Q এর মধ্যে add → source → Q খালি না হওয়া loop চলবে → Q-র প্রথম থেকে pop, sequence = add → এর adjacent যারা তাদের push করবো

এই technique apply করে

3) কোন path follow?

↓

start থেকে end -এ যাবো?

↓

cost/value কত হবে?

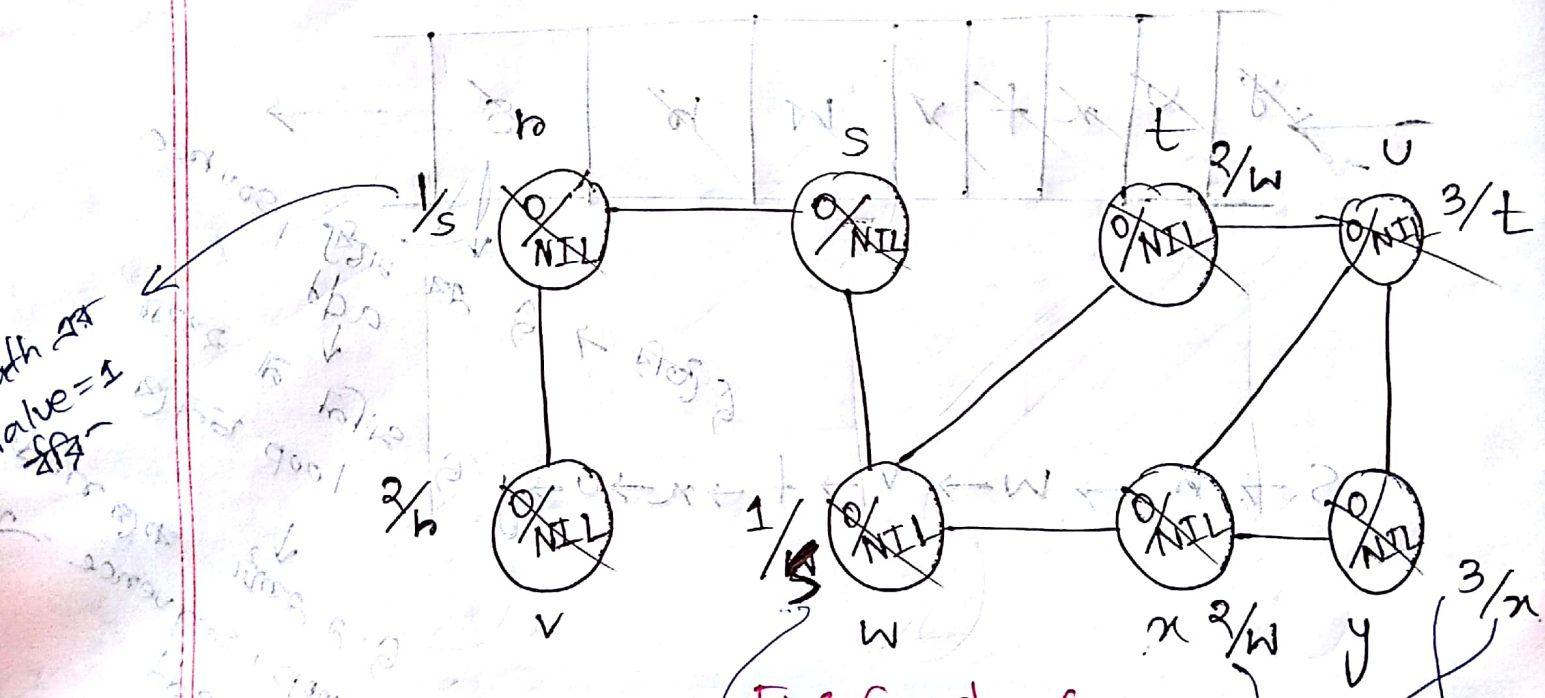


Fig: Graph G

(কোন node cross করলে back করবো না আর)

shortest path [s → w → x → y : 3

G.V = { r, s, t, u, v, ... }

G.E = { r.v, r.s, ... }

vertex

edge

sequence print করতে বললে 3, 4, 6, 7, 15, 16 লাইন দিবো না।

Lec – 18

BFS (G)

edge বাদে সব vertex এর graph

1) for each vertex $U \in G.V - \{S\}$
2) U.color = WHITE
3) $U.d = \infty$
4) $U.\pi$ = NIL
5) S.color = GRAY
6) S.d = 0
7) $S.\pi$ = NIL
8) $Q = \emptyset$
9) Enqueue (Q, S) / Q ⇐ S

S এর 3টা property

S থেকে start, তাই distance = 0

Q এ ঢুকলে WHITE হয়ে যাবে।

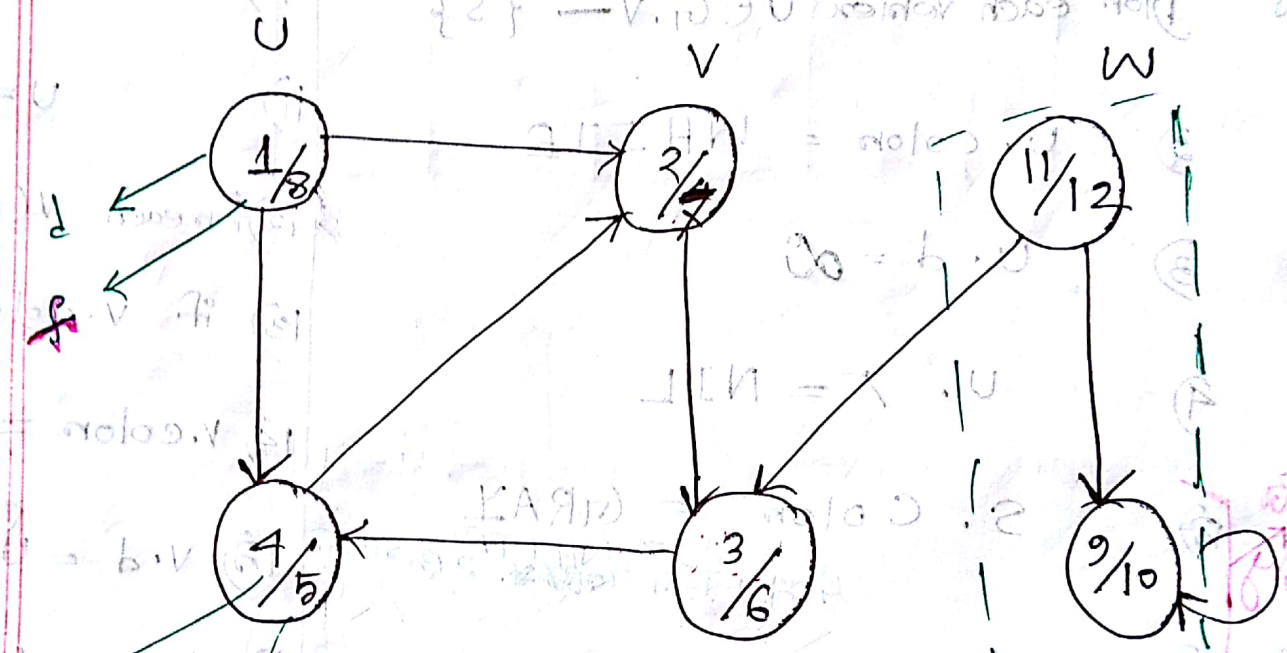
কোথা থেকে আসলাম একটা variable

Q তৈরির syntax

queue এর মধ্যে S push করতে হবে

10) while $Q \neq \emptyset$ — যতক্ষণ চলবে
11) U = Dequeue (Q) — pop
12) for each $V \in G.Adj[U]$
13) if V.color = WHITE
14) V.color = GRAY
15) V.d = U.d + 1 — U এর adj তাই
16) $V.\pi = U$
17) Enqueue (Q, V)
18) U.color = BLACK — বের হয়ে যাবে Q থেকে

DFS



v থেকে v তে যাওয়া যাবে না। কেননা 2 বারের বেশিবার

চক্র থেকে

$x \to y \to v \to u \to w \to z$

$u \to z \to w$

depth এর কাজ আছে DFS এ

Lec - 19

DFS (G)

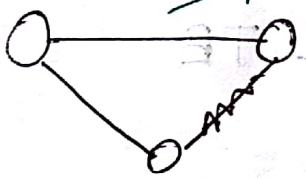
1) for each vertex $u \in G.V$
2) $u.color = WHITE$
3) $u.\pi = NIL$
4) $time = 0$ (initially 0)
5) for each vertex $u \in G.V$ (যে কোনো node এর জন্য)
6) if $u.color = WHITE$ (প্রতিটা node এর জন্য, যেকোনো একটা node এর জন্য)
7) DFS_Visit (G, u) (graph এর জন্য)

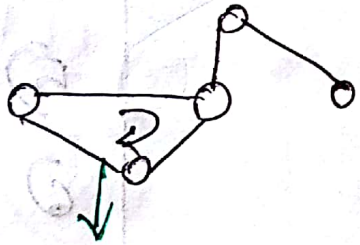
DFS Visit (G, u)

1) time = time + 1
2) u.d = time
3) u.color = GRAY (কাজ শুরু)
4) for each $v \in G.Adj[u]$
5) if v.color = WHITE
6) $v.\pi = u$
7) DFS_Visit (G, v)
8) u.color = BLACK
9) time = time + 1
10) u.f = time (প্রতিটা node এর জন্য, time 1 করে + শেষ হওয়ার)

Chapter - 5

Tree

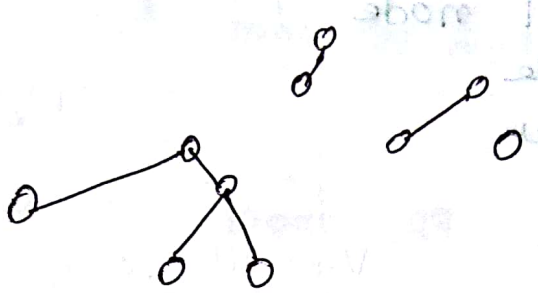




cyclic graph যে কোন এক জায়গায় cycle থাকবে।

Definition with fig

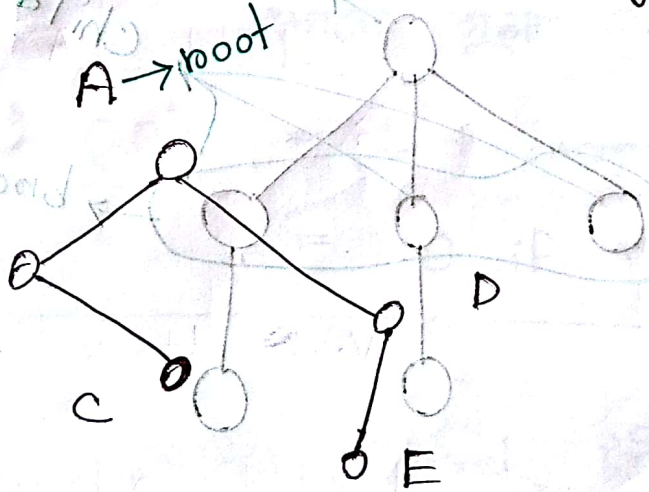
Forest : Acyclic graph.

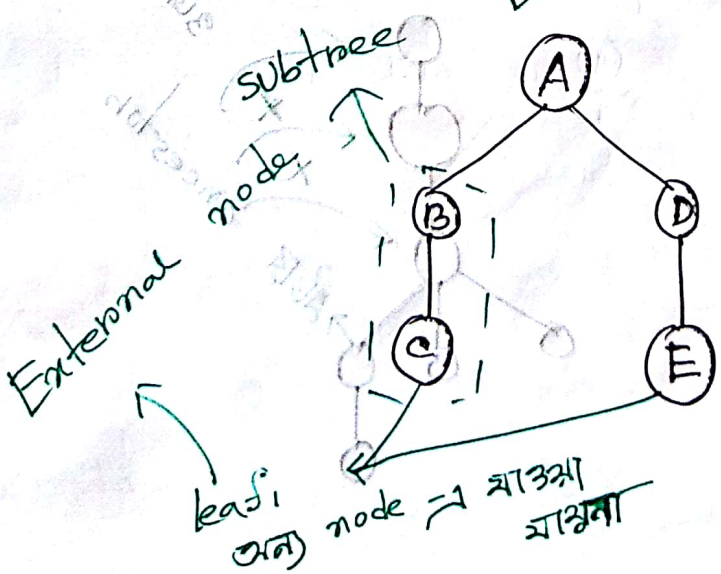


Tree: Connected acyclic graph.

Subtree:

tree এর একটা অংশ





→ এই ছবি দিবো।

অন্য node → যাওয়া যাবে না

root

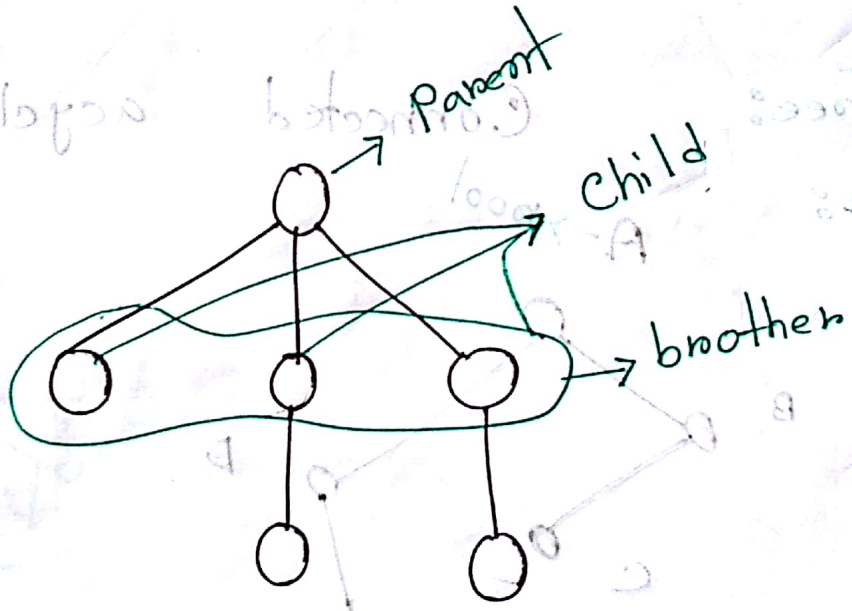
leaf / external node

internal node

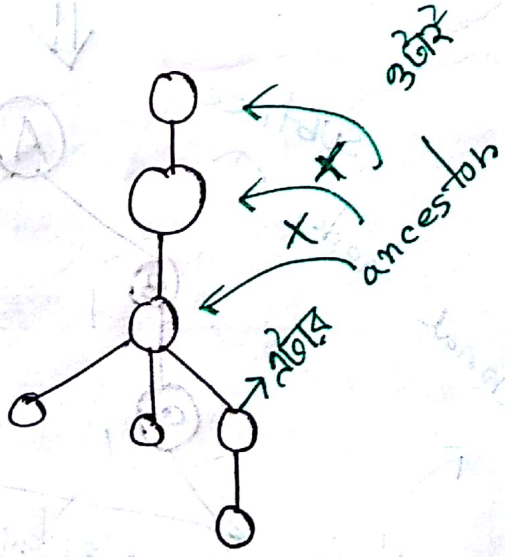
Parent / Father

Child / Son

Sibling / brother



Ancestor

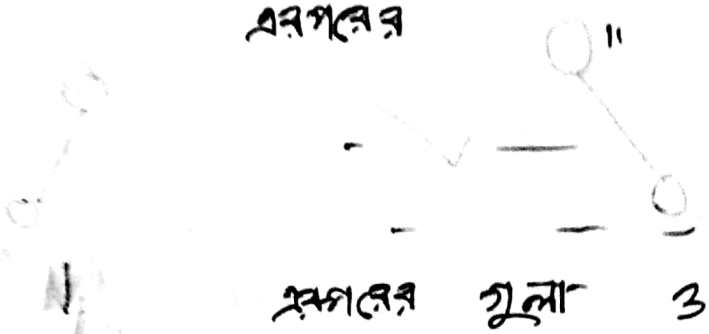


Descendant :

↘ tree এর নীচের সব

level : root এর level 0

এরপরের " 1

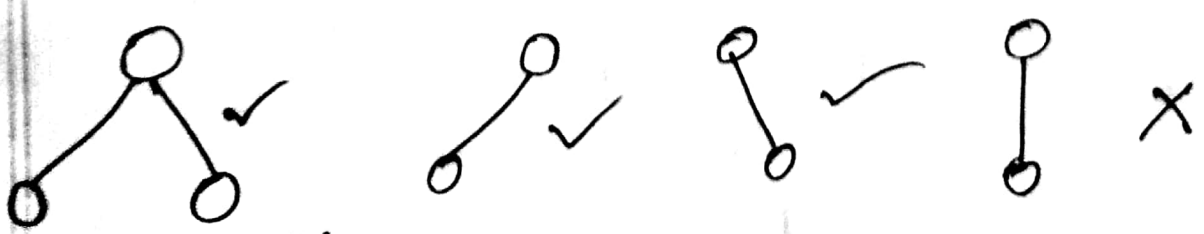


এরপরের দুলা 3

level (P) = 1 + level (Father (P)) = 0 if P is root

height: সর্বোচ্চ level টাই height.

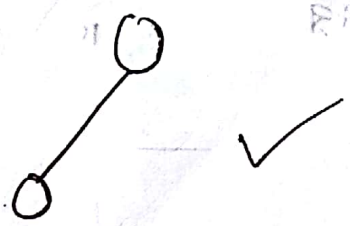
binary tree: এমন tree- node এর max দুটা side.

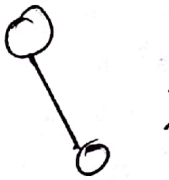


leftchild rightchild

Complete binary tree:

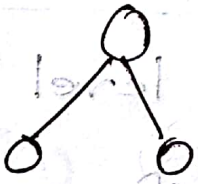
(একটা node নিয়ে আঁকলে)



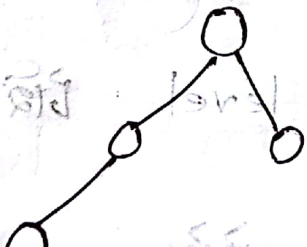


(২টা node নিয়ে আঁকলে)

Complete binary tree তে না এ আলো যাবে না এঁকা

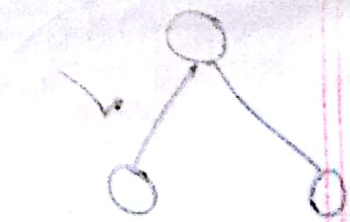


(3টা node নিয়ে আঁকলে)



(4টা node নিয়ে আঁকলে)

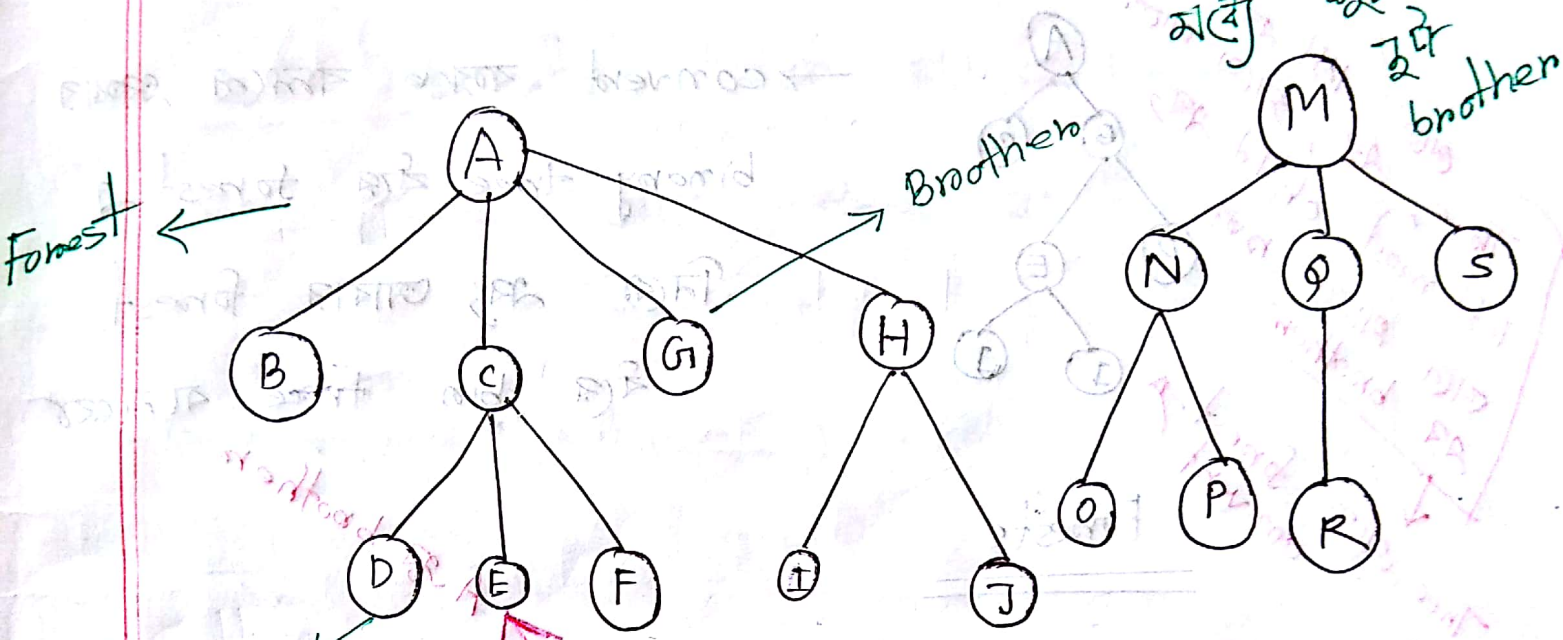
আর একটা level জোস করে যেতে হবে



Lec-20

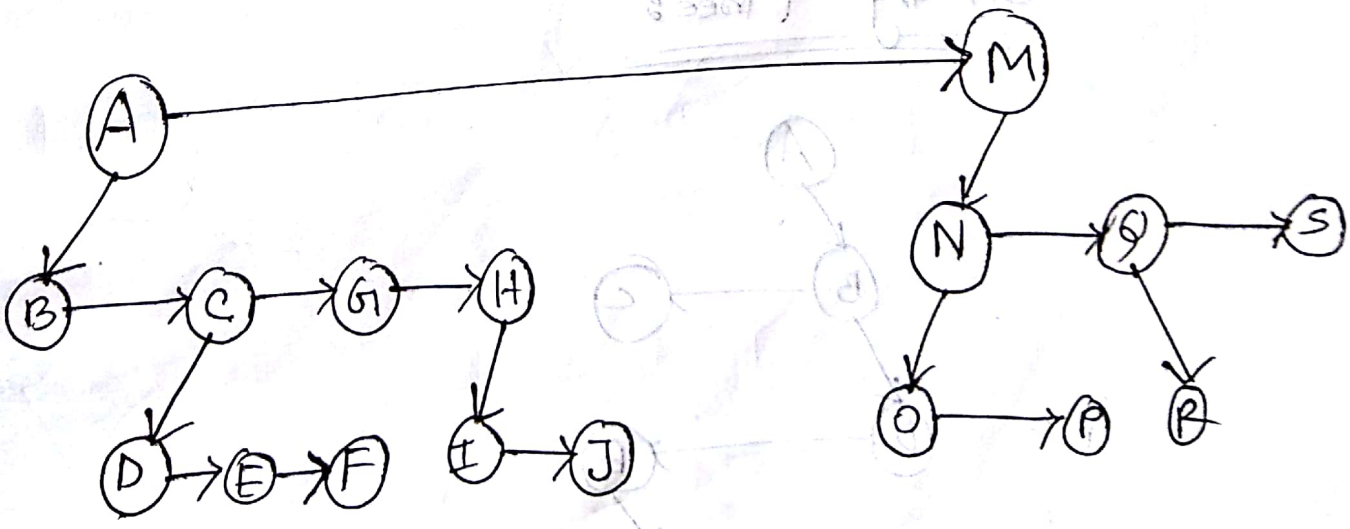
দুটা tree এর মধ্যে

অন্তত Root দুটা brother

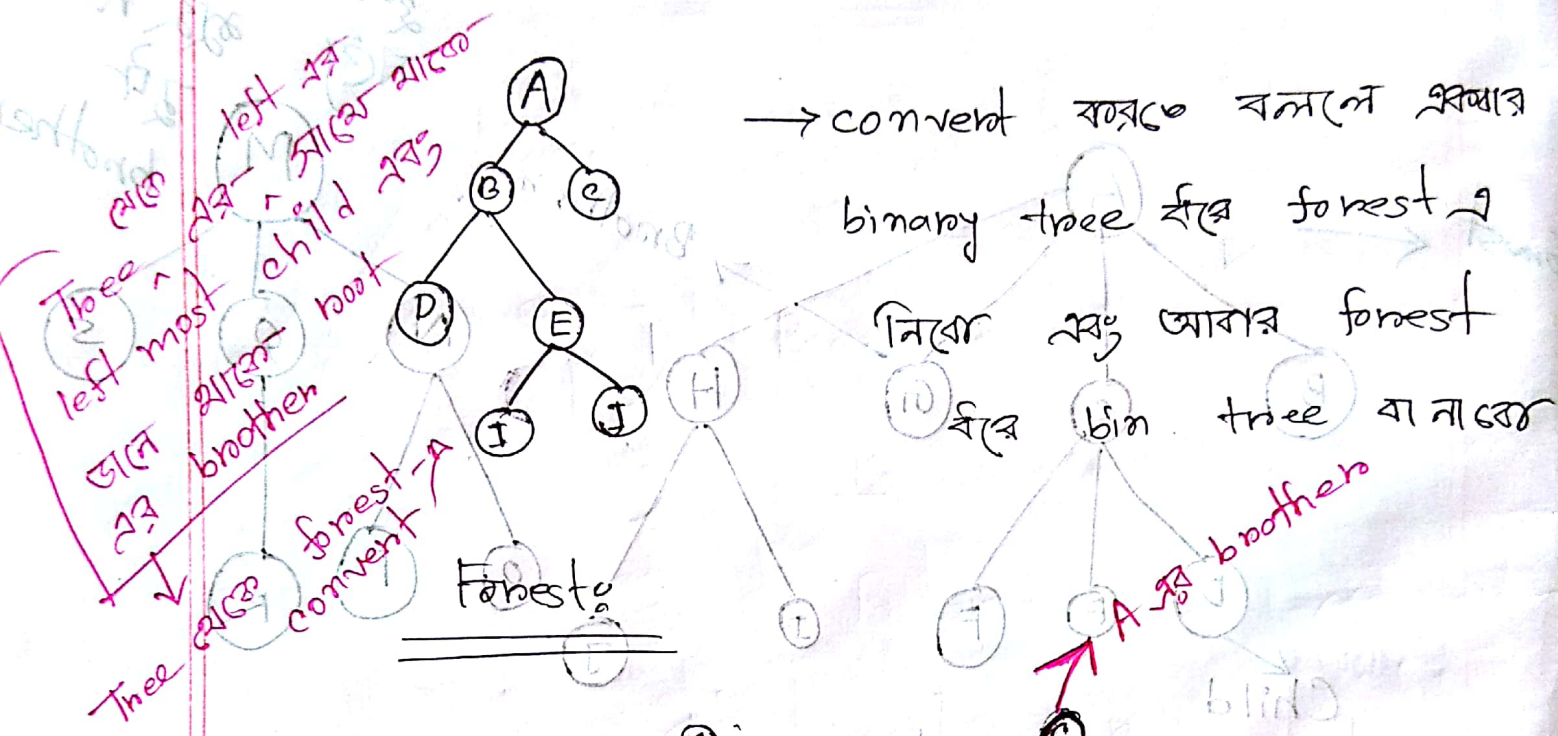


Child tree বা forest কে computer এ represent করা যায় না।

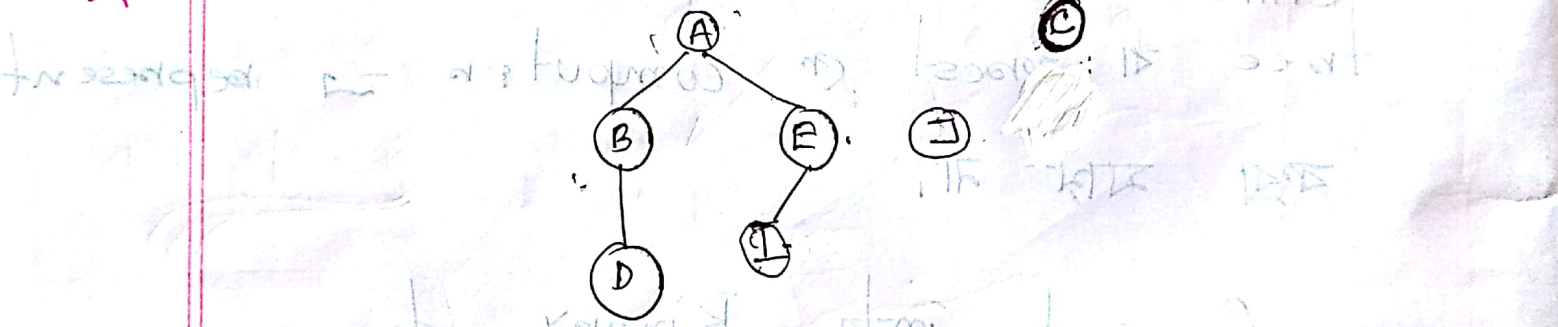
Convert into Binary tree:



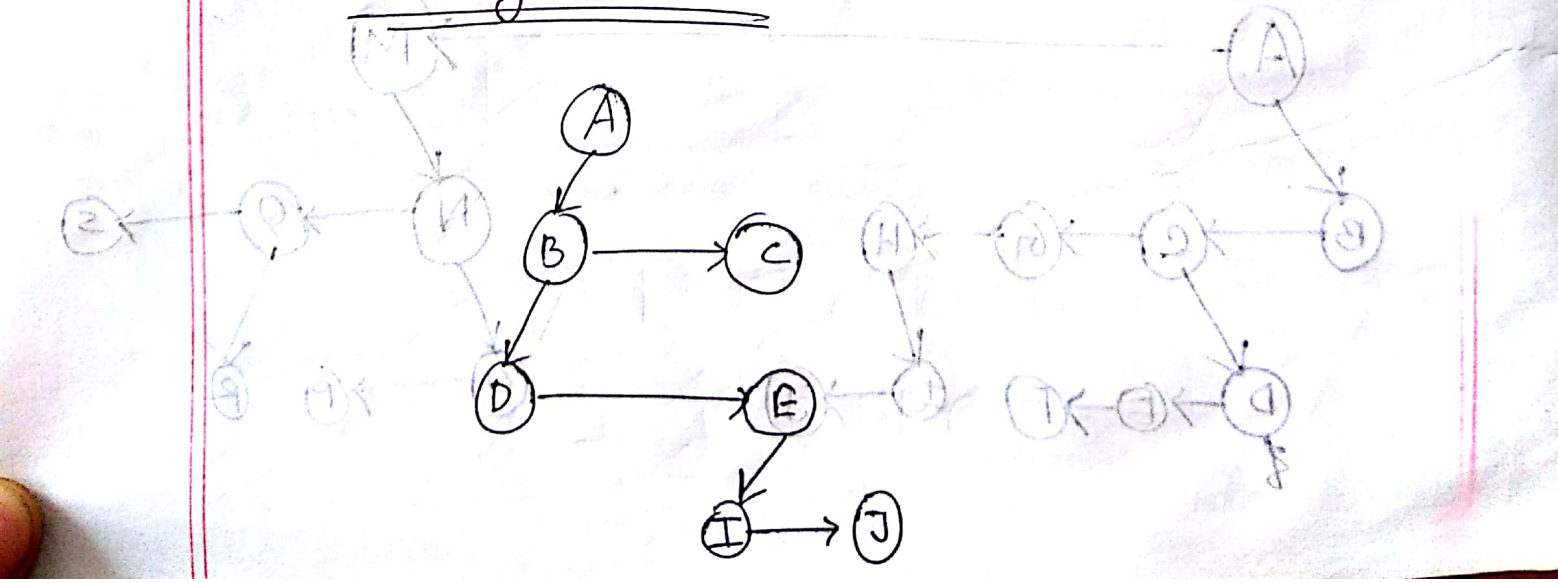
Natural correspondence or convert:



→ convert করতে বললে প্রথমে binary tree করে forest নিবো এবং আবার forest করে bin tree বানাবো



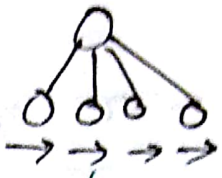
Binary Tree:



Natural correspondence : Tree বা forest থেকে bin tree এবং bin tree থেকে tree বা forest convert.

Lec - 21

Traversal



Exam / algo / traverse

1) Preorder : root → left → rt

2) Inorder : left → root → rt

3) Postorder : left → rt → root

4) Level order

4) Level Order :

traverse

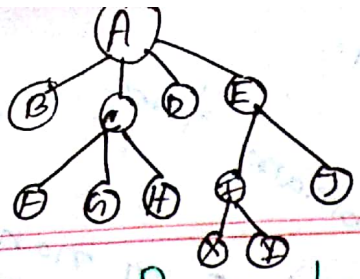
0 level ↓

1 " ↓

left to rt traverse always

figure ← A M B C G H N g s D E F I J O P R

21/6/17



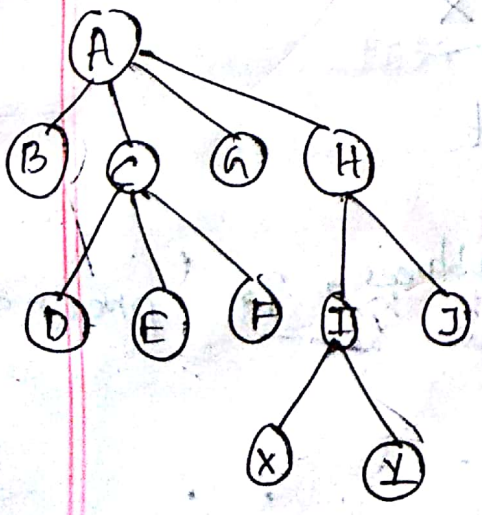
1) Preorder: left থেকে নাও

A B C f g h d e I X Y J

B এর নীচে
নাই কিছু

X 2) Inorder: অনেক বার traverse হবে না
forest/tree " " "
but binary tree তে যাবে
left একটা
right একটা

3) Post order:



B D E F C G X Y I J H A ... M

opNRgS: M

Algorithm of Preorder of Forest: Recursive

(annotations: in/post/level order; বা binary tree; বা non-rec)

1) Visit the root of the first tree
(Visit — কাজ নেই; first tree — যেহেতু forest)

2) Traverse the subtrees, if any of the first tree in preorder.
(Traverse — বাকি আছে নাকি; subtrees — root এর under-এ যদি থাকে; preorder — rec process)

3) Traverse the remaining trees if any, in preorder.

tree হলে 3 X

1) Visit the root

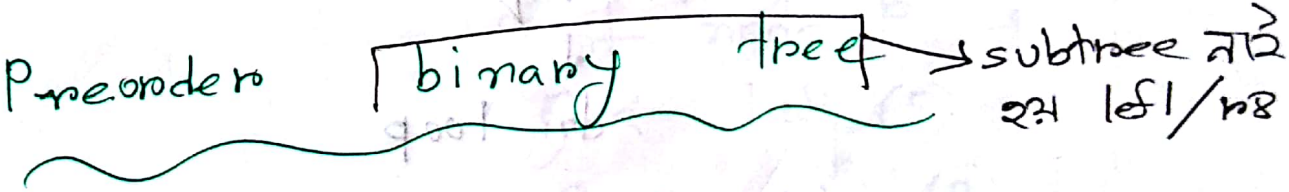
2) Traverse the subtrees, in preorder

Postorder + Inorder + level Forest:

2 নং line 1 -2

1 " " 2 "

3 নং line 3 -2



1) Visit the root

2) Traverse the subtrees of left of right ...

in preorder

3) traverse ...

3 নং line যাক

Lec-29

Preorder Binary Tree:

Non-Rec

↙

stack / queue

↓

dn loop

Algo:

S ← empty stack

S ⇐ root

while S ≠ empty do

P ⇐ S
If P ≠ NIL
visit node P

stack → Right (P)
or other

S ⇐ Right (P)

S ⇐ Left (P)

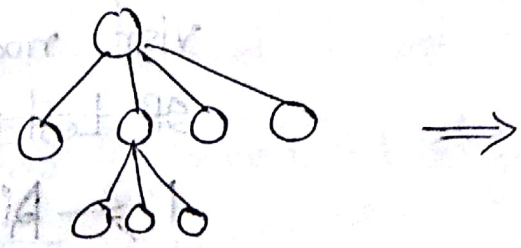
Level Order Binary Tree:

```
Q ← empty Queue
Q ← root
while Q ≠ empty do
    P ← Q
    if P ≠ NIL
    visit node P
    Q ← left(P)
    Q ← right(P)
```

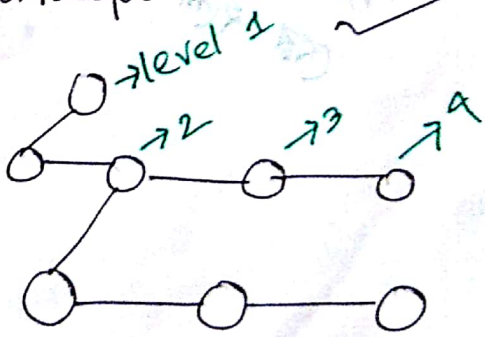
Level order Traversal of a forest:

একটা level পাড় করে

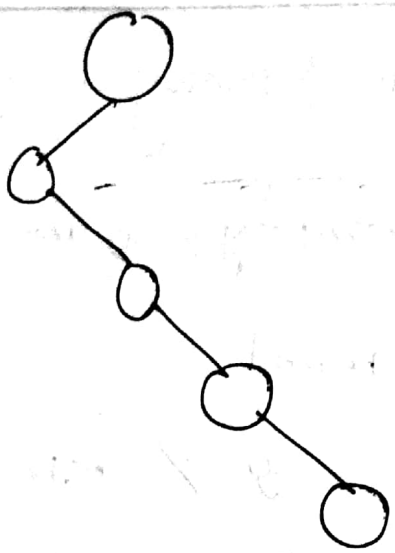
represented as a binary tree via natural correspondence



forest হিসেবে নাই



bin tree হিসেবে আছে



Algo:

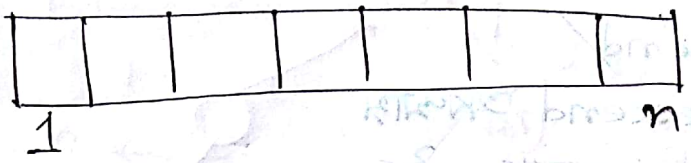
```
Q ← empty queue
Q ⇐ root
while Q ≠ empty do
      P ⇐ Q
      while P ≠ NIL do
            visit node P
            if Left(P) ≠ NIL then Q ← Left(P)
            P ← Right(P)
```

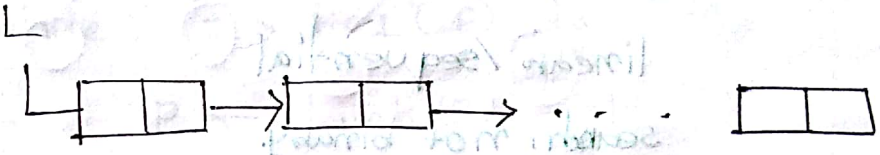
bin tree সহজে বের হলো

Link

Quiz 3

Chapter – 7 (Searching)

1) 

2) 

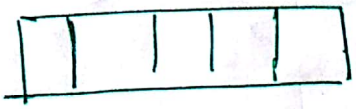
① array তে 1 থেকে n পর্যন্ত data

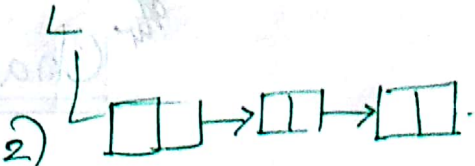
Unsorted

```
i ← 1
While i ≤ n and z ≠ xi do
        i ← i + 1
if i ≤ n then "Found"
else "Not Found"
```

② For Linked List:

```
p ← L
While p ≠ NIL and z ≠ key(p)        → info ফিল্ড linked list-এ
        p ← Link(p)
if p ≠ nil then "Found"
else "Not Found"
```

1) 

2) 

ascend/
descend অবস্থায়
সাজানো, এটার
linear/sequential
search, not binary.

↓

sorted থাকলে

যেই key বলবে search
করতে সেটা যেখানে
আছে তার পর থেকে break
করে দিবো, আগের code
modify

```
i ← 1
while z > xi do
    i ← i + 1
if xi = z then "found"
else "Not Found"
```

```
p ← L
While  z > key(p)
        p ← Link(p)
if key(p) = z then "Found"
else "NOT Found"
```

Lec -23

Sequential search using sorted Linked List:

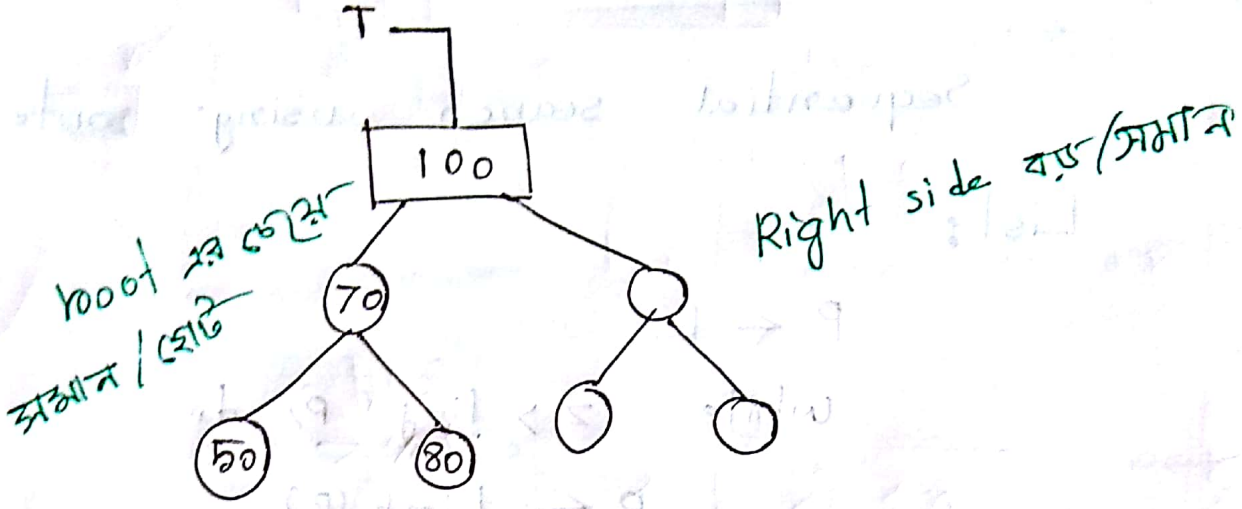
```
P ← L
while  z > key(P) do
        P ← Link(P)
if key(P) = z then "Found"
else "NOT Found"
```

Binary search:

```
l ← 1
h ← n
found ← False
while l<n and not found do
        m ← ⌈(l+h)/2⌉
        case :
                z < xm : h ← m - 1
                z > xm : l ← m + 1
                z = xm : found ← true
if found = true then "Found"
else "NOT Found"
```

→ Sorted list আছে তাহলে

Binary Seach Tree:



P ← T

root এর address

found ← false

while P ≠ nil and not found

Case :

Z < Key(P) : P ← left (P)

Z > Key(P) : P ← right(P)

Z = Key(P) : found ← true

if found = true then "Found"

else "Not Found"

কতবার search?

Binary search এর number of searches = $\log_2 n$

(but sorted algo থাকতে হবে)

Binary search linear seach এর চেয়ে কত time faster?

⇒ 1 বার search করতে 500

2 " " 250

3 " " 125

⋮

10 " " 9 → but linear search 1000 বার

2000

এর জন্য 11 times faster bin search

Hashing

f(1-10)

f(1-10) এর মধ্যে

index অনুযায়ী value রাখলে ভালো but problem : Range

Duplicate data

range এর মধ্যে আনতে

1234 % 10 → এই নিয়ম

=

z কে array তে রাখতে হবে

h is hash function

$h(z)$

$z \% 10$

$0 \le h(z) < m$

$T[h(z)] \leftarrow z$

index নিবেই $h(z)$ আসা মানে collision

Lec -24

Collision Algo

next lec

10 টা ঘর হলেও 3 bit .

(যে কোন 3 bit নিতে হবে
↓
but প্রতিটা data র জন্য same 3 টা bit)

Extraction Method:

8 টা data কে সাজাও,

ques:

0	1	2	3	4	5	6	7

→ T, H, E এর value এখান থেকে

index -এ convert
জন্য h funct
0-7 -এ convert করে লেখাগুলো

Given,
A : 00001
B : 00010
⋮
Z : 11010

THE : 101000100000101 = 101 = 5 *
15 bit আছে but 3 bit এর জায়গা আছে

OF : = 6

AND : = 0

TO : = 7

A : 00001 = 1

IN : = 2

THAT : = 4

IS : = 3

Hash Functions:

1) Extraction
2) Compression
3) Division $h(z) = z \bmod m$ → $0-(m-1)$
4) Multiplication $h(z) = [m (z\theta \bmod 1)]$; where : 0<θ

Extraction:

AND	A	IN	IS	THAT	THE *	OF	TO
0	1	2	3	4	5	6	7

→ Ans

৮ টা ঘর → so 3 bit এর বেশী নেয়া যাবে না

15 " " → so 4 bits " "

Design Principles (কী কী মনে রাখতে হবে)

Primary | Clustering (একই typ value হতো যদি
group
Natural " আগের ques এর চার্ট data এর - - -)
↓
আলাদা value হবে

Data যত বেশী involve করবে তত কম হবে collision

1) The hash function should be a function of every bit of the element.

2) A hash func should break up naturally occuring clusters of elements.

3) A hash function should be very quick and easy to compute.

আগের ques - এই এটা compression method:

#

T : 1 0 1 0 0

H : 0 1 0 0 0

E : 0 0 1 0 1

1 1 0 0 1

AND/OR/NOT. result এক, but এক bit involve এলে 0

X-OR

Lec - 25

h_1 ↓

compression method apply

h_2 ↓

Division " "

h_3 ↓

Multiplication " " "

$m = 31$

Exam ✓ Collision :

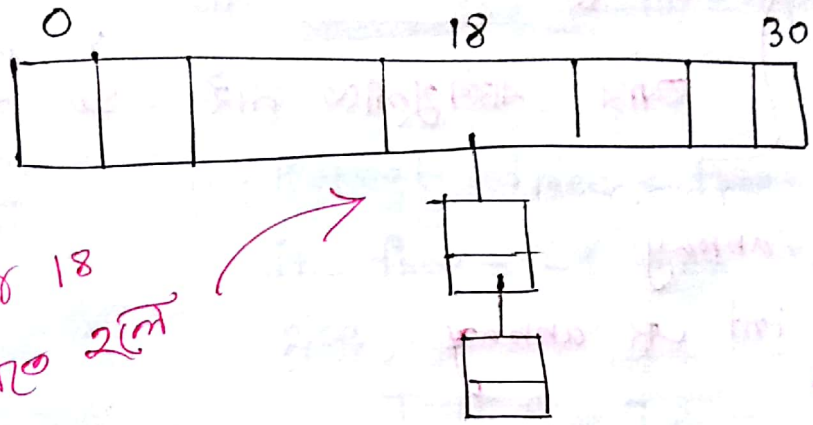
1) Separate Chaining
2) Coalesced "
3) Open addressing / Linear Probing
4) Double hashing

29 → quiz (4)

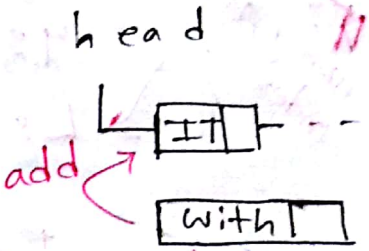
page: (336 + 342)

1) Separate Chaining:

h2 column এর value same হলে



bucket 18 এখানে এলো



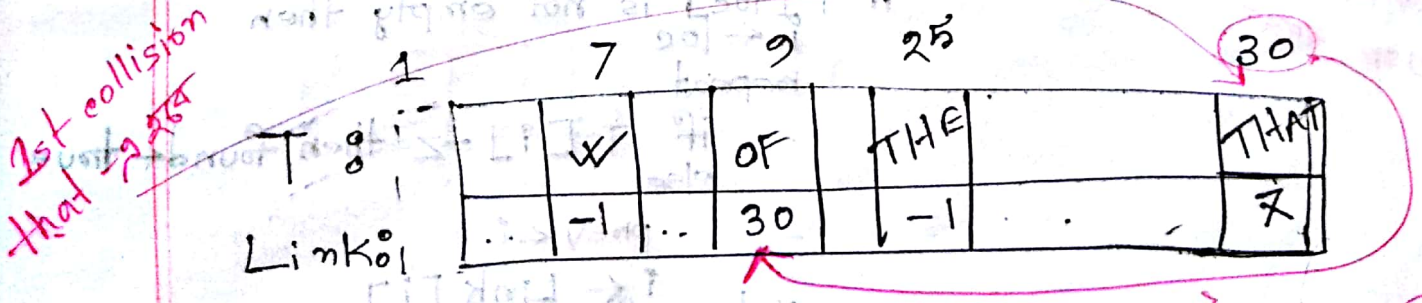
Linked List এর algo: insert func ↓ head এ add easy

প্রতিটা আলাদা আলাদা chain

2) Coaleseed chaining:

page: (336+344)

h1 column অনুযায়ী



collision হলে কোন দিক থেকে খালি ঘর আছে তাহাতে রাখবে।

9 - এ না পাওয়া গেলে 30 তে
30 তে " " 7 "
আর বাকী গুলাতে নাই so -1

T: array
L: int এর array

[Exam → last একটা pic → ans

344 → last pic final ans

Algo:

loc ← h(z)

location → hash func use করে বের করলাম

found ← false
if T[loc] is not empty then
  i ← loc
  repeat
    if T[i] = z then found ← true
    else
      prev ← i
    i ← Link[i]
  Until found OR i = -1

data না পাওয়া গেলে
পাওয়া গেলে লিঙ্কড
না

if not found then

if T [loc] is empty then T[loc] ← z

else

collision হলে

repeat free ← free - 1 until T [free] is empty

if free = -1 then "Overflow"

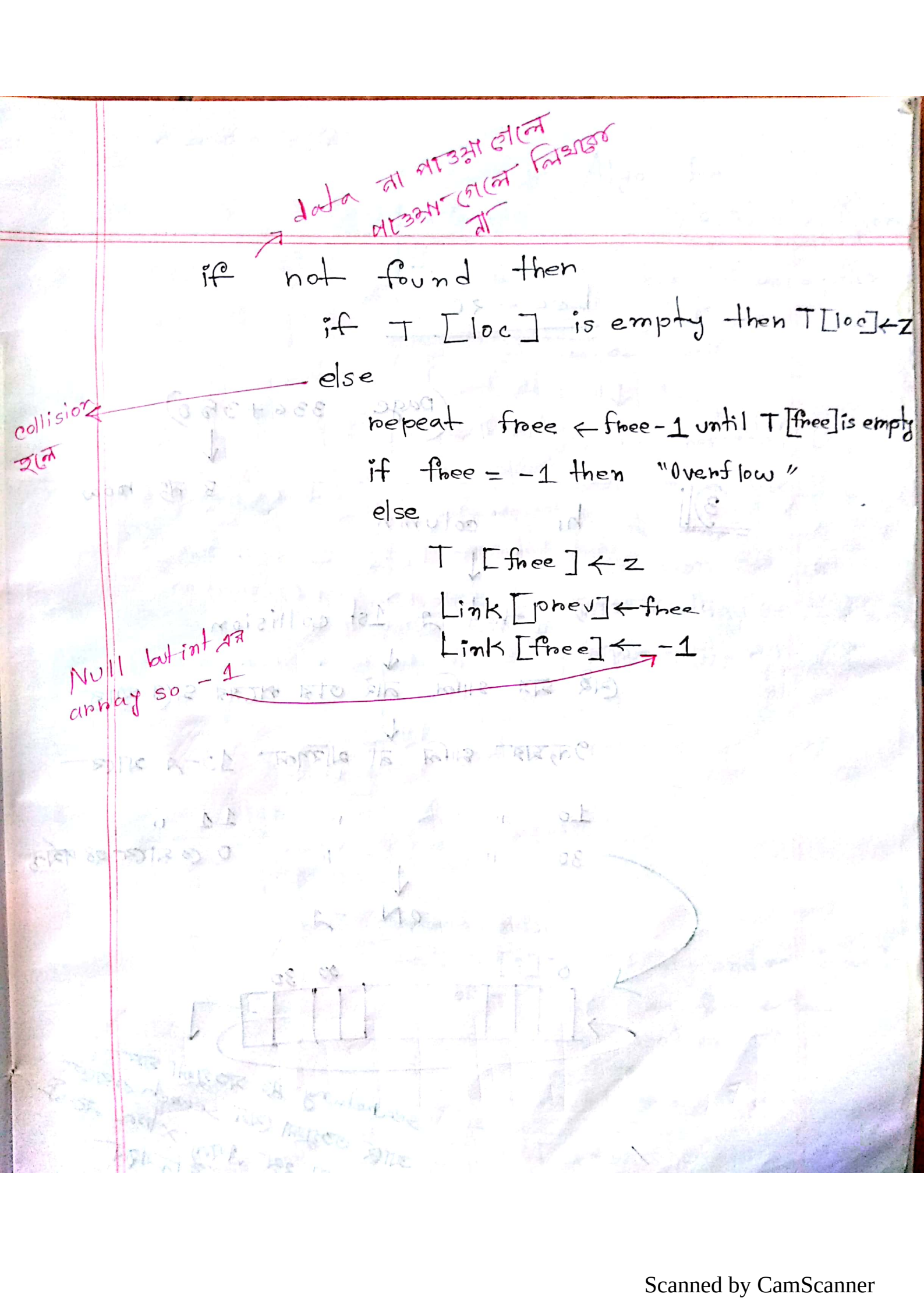
else

T [free] ← z

Link [prev] ← free

Link [free] ← -1

Null but int এর array so -1



Lec - 26

(page 336+350)

↓

3 নং row

3) h_1 column

↓

that -> 1st collision

↓

যেই ঘর খালি নাই তার পরের ঘরে যাবে

↓

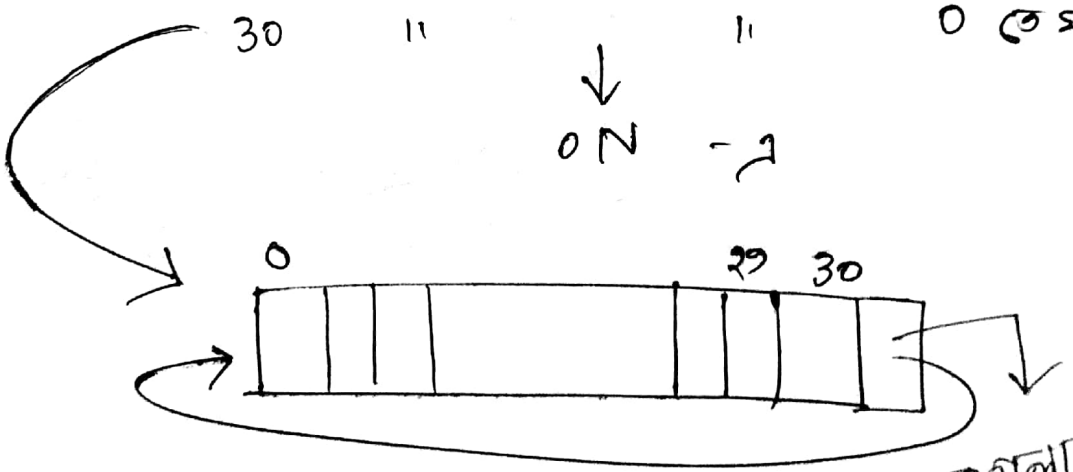
0 নং ঘর খালি না থাকলে 10-এ যাবে

10 " " 11 "

30 " " 0 তে যাবে ২৪ পর্যন্ত

↓

ON ->



searching এর সময়গুলা যদি আরো কমানো যেন search করতে না হয় এজন্য xtra একটা ঘর নিলাম

Algo for 1, 2, 3, 4 collision method.

hash func এর description

এই data নেয়ার

↓

division method নিয়ে

hash func use

↓

Collision হলে যে

কোন method (1, 2, 3, 4)

দিয়ে solve

ques for exam

Algo

i ← h(z)

found ← false

while T[i] is not empty and not found do

    if T[i] = z then found ← true

    else i ← (i+1) mod m

if not found then

    if n = m − 1 then "Overflow"

    else

        T[i] ← z

        n ← n+1

page (336+353)

4) Algo

২টি method এর column
এর hash
↓
collision হলে (h_1)
আগের column
এর সাথে যোগ

$i \leftarrow h(z)$

$\Delta \leftarrow \delta(z)$

found ← false

while T[i] is not empty and not found

    if T[i] = z then found ← true

    else $i \leftarrow (i + \Delta) \bmod m$

if not found then

    if $n = m - 1$ then "Overflow"

    else

        T[i] = z

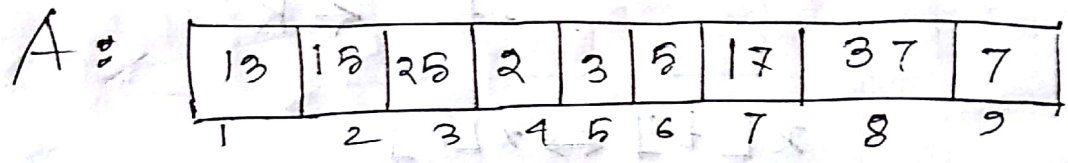
        $n \leftarrow n+1$

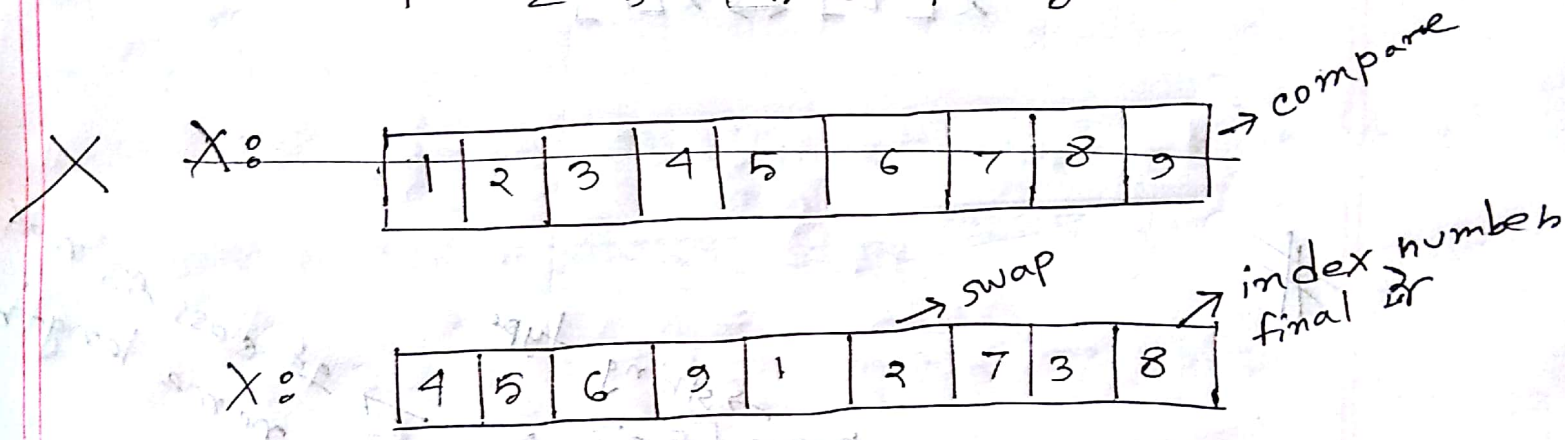
That

Scanned by CamScanner

Lec - 27

Chapter - 3





~~#~~ Array এর ডাটা এর length বড় হলে কোন sorting ?

একটা array তে ডাটা রাখলে আমরা index ছোট থেকে বড়

⇒ যেই sorting ই হোক

↓

compare করবে

↓

index num change

↓

এটা index sorting

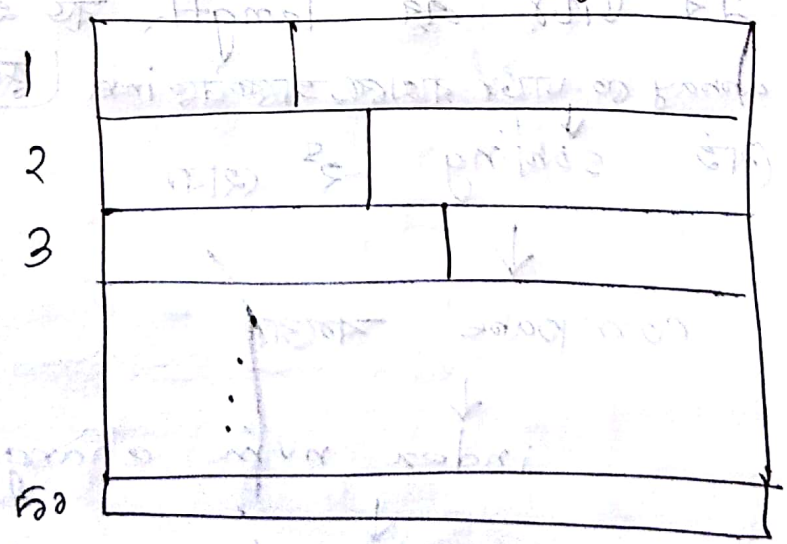
Algo:

```
for t = n to 2 by -1 do
    i ← 1
    for k = 2 to t do
        if A[x[i]] < A[x[k]] then
            t ← k
    x[t] ↔ x[i]
```

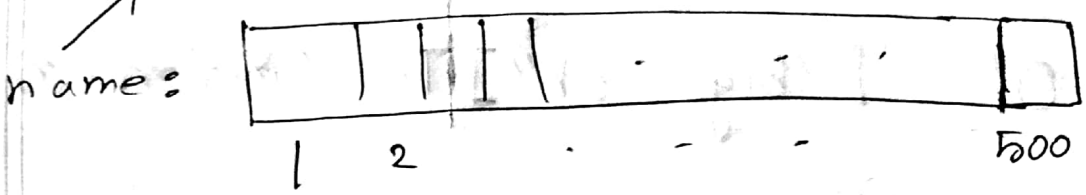
#

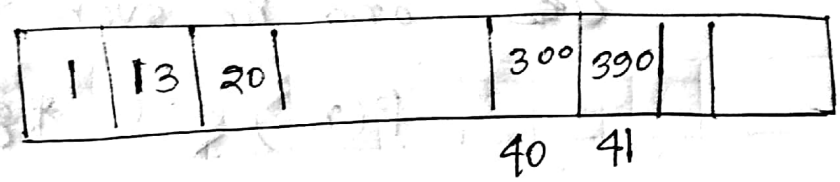
name [50] [30] → string type → এই class এর max name length

student এর array



একটা array র





যেই array র element র length vary করে তাদের কীভাবে store করবো?
(আগের ques -এ length বড় ছিলো)

Notation s: → 3টা (3 marks র ques ↓ definition)

$\frac{n(n+1)}{4} = O(n^2)$ ↑ n^2

$2n+5 = O(n)$ ↓ কী হচ্ছে notation এটাই

"O" [Big "oh"]

The function $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that

$f(n) \leq c * g(n)$ for all n, $n \geq n_0$

Examples:

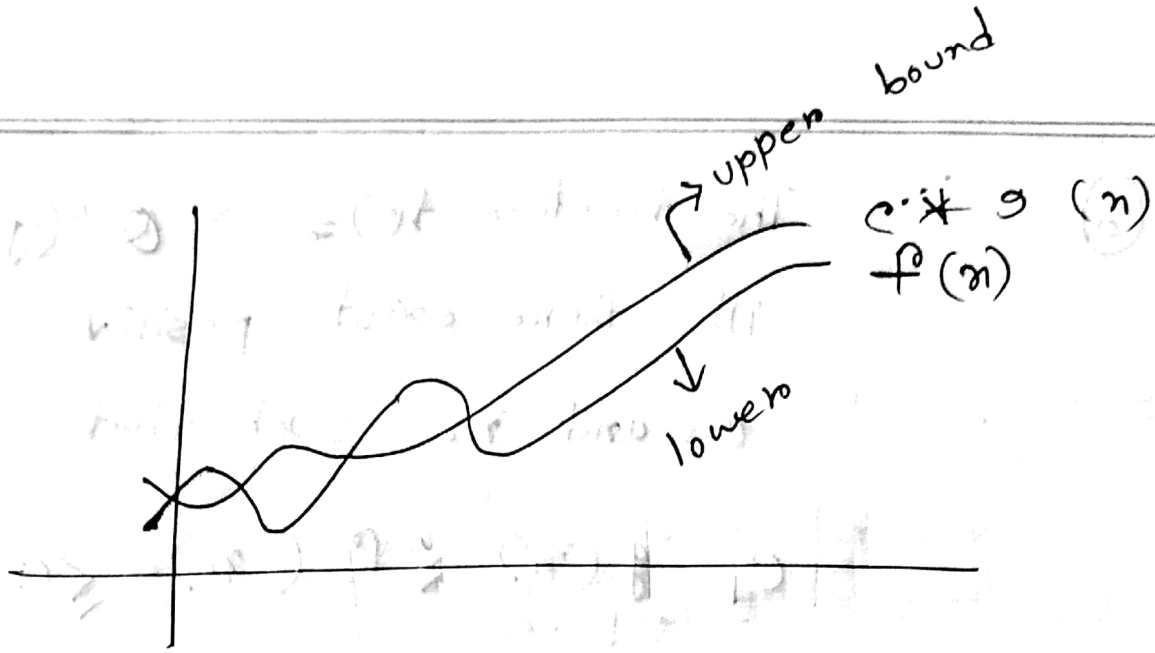
$3n+2 = O(n)$ ($f(n) \to 3n+2$, $g(n) \to n$)

as $3n+2 \leq c$ → 1 হলে হবে না only, 4, 2, 3 হলে হবে

$4n$

$c = 4$

$n \geq 1$



$f(n) = O(g(n))$

(2) The function $f(n) = \Omega(g(n))$ iff there exist positive constants c and n_0 such that

$f(n) \geq c * g(n)$ for all $n, n \geq n_0$

(3) The function $f(n) = \Theta(g(n))$ iff there exist positive constants c and n_0 such that

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n)$$

for all n, $n \geq n_0$

Ex - 2, 6, 7(a)
Chap - 3
last ছিল notation

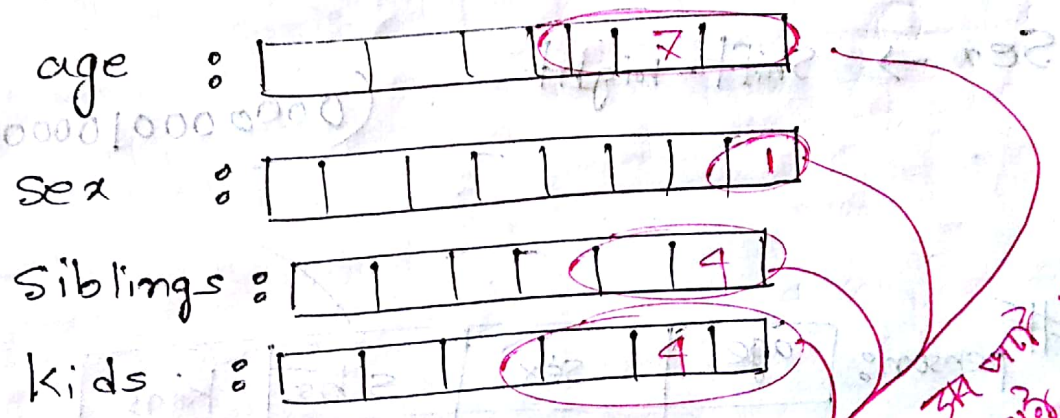
Lec - 28

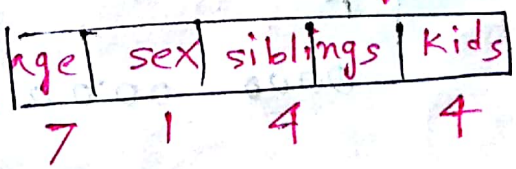
Chapter - 3

Packed Word

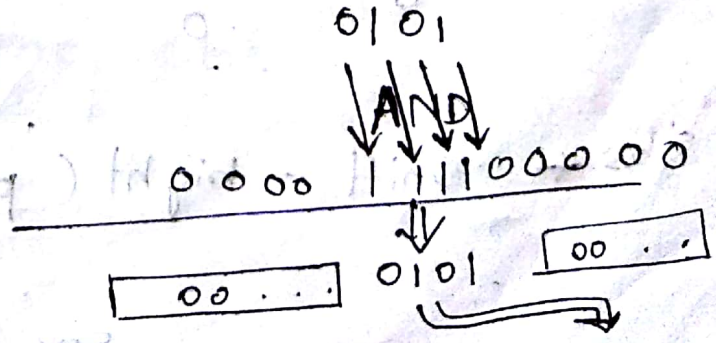
age = 38

= 130





Max 16 bit এর মধ্যে সব



sibs → Shift right (person data $00F0_{16}$, 4)

Hex value

Sex → Shift right (person - data AND 0100_{16}, 8)

age → " 0101 " AND FE 00_{16}, 9)

AND

0100

Sex → Shift - right (0000000100000000_2, 8)

person: | age | sex | sibs | kids |

7 1 3 3

0000 0000 1110 0000

AND

0 0 E 0

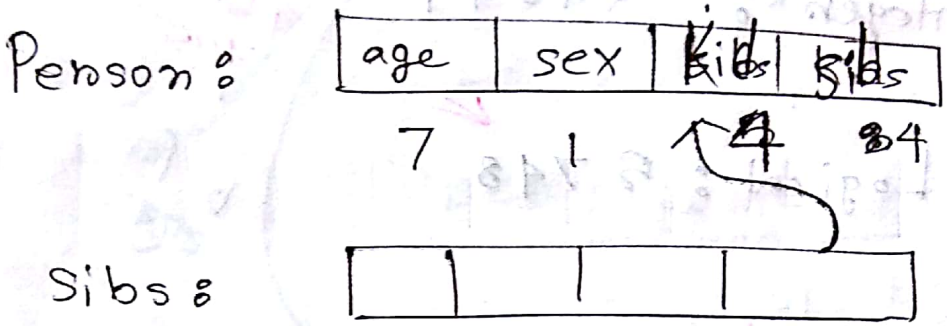
sibs ← Shift - right (person data AND 0018_{16}, 3)

sex (,8)

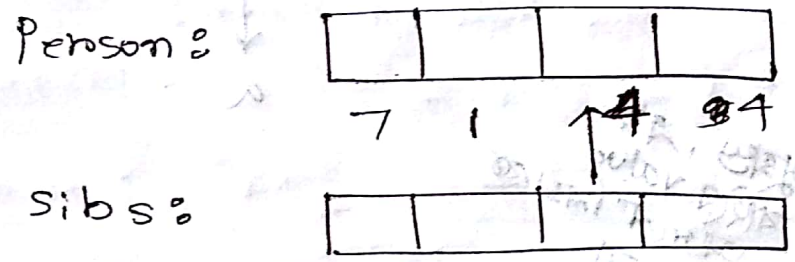
age (,9)

retrieve/ store/ unpack age sex sibs kids data

Packing :



left shift করলে + OR করতে হলে 0 AND 3 করা যায়
করে নিতে হবে



Person-data ← Shift-left (age, 9) age কে রাখার জন্য 9 বার shift করতে হবে

OR Shift-left (sex AND 1₁₆, 8)

OR Shift-left (siblings AND F₁₆, 4)

OR Shift-left (kids AND F₁₆)

↓
২ জায়গাতেই

dec ⇒ 4 int value নিবে

0 1 0 1 0 1 1 1 0 1 0 0 1 0 0 6

⇒ Integer : 22344

4 bit করে

Logical : 5748

5 কিভাবে করে

Logical value মানে Hex value

convert করতে হবে

Character string: 97

a

A – Z
a – z
0 – 9
→ মূলত এর বাইরে value আসলে লিখবে না

(10011)2 Calculate it.

⇒ (1 × 2^4 + 0 × 2^3 + 0 × 2^2 + 1 × 2^1 + 1 × 2^0)

$(10011)_{10}$

$\Rightarrow (1 \times 10^4 + 0 \times 10^3 + 0 + 0 + 1 \times 10^1 + 1 \times 10^0)$

$= 10000 + 10 + 1$

$= 10011$

Lec: 29

Bin to Dec:

$(10011)_2$

$= 1\times2^4 + 0\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$

Dec to Dec:

$(10011)_{10}$

$= 1\times10^4 + 0\times10^3 + 0\times10^2 + 1\times10^1 + 1\times10^0$

$= 10011$

b base এর একটা number:

$a_{k-1}\ a_{k-2}\ a_{k-3} \ldots\ldots a_3 a_2 a_1 a_0$

$= a_{k-1}\times b^{k-1} + a_{k-2}\times b^{k-2} + a_{k-1}\times b^{k-1}$ → b=1

$+ \ldots\ldots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0 \quad \cdots (1)$

$= (a_{k-1} b^{k-2} + a_{k-2} b^{k-3} + \ldots + a_2 b + a_1) b + a_0$

$= ((a_{k-1} b^{k-3} + a_{k-2} b^{k-4} + \ldots$

$\vdots$

$\ldots + a_3 b + a_2) b + a_1) b + a_0$

$\vdots$

$= (((\ldots((a_{k-1} b + a_{k-2}) b + a_{k-3}) b$

$+ \ldots\ldots + a_2) b + a_1) b + a_0$

$\quad \cdots (11)$

... (digit × base + next digit) base ...

eqn (11) is called Horners Method.

Q11 Derive Horners Method?

100 11 এর traditional এবং Horner's Method এ value বের করে দেখাও।

Multiplication

$1 \times 10^4 + 0 \times 10^3 + 0 \times 10^2 + 1 \times 10^1 + 1 \times 10$

$= 1 \times 10 \times 10 \times 10 \times 10 +$

Horners method এ কতবার multiplication? ⇒ $(10011)_{10}$

$= (((*(1 \times 10 + 0)10 + 0)10 + 1)10 +$

4 বার multi

=

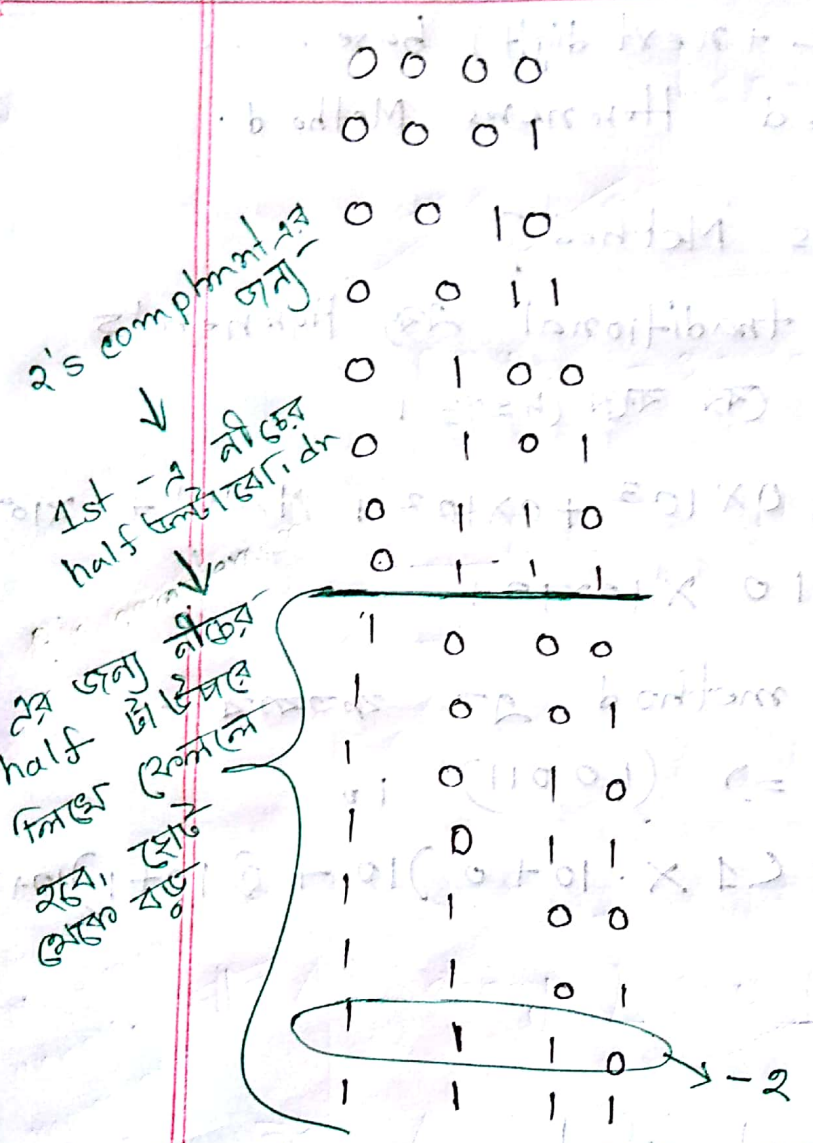
Signed Integer

2's complement:

−8 to +7

1's complement:

−7 to +7



নীচের half উপরে লিখবো.
0000, 1111 overlap করবে
কারণ 0000 = +0, 1111 = -0
যে কোন একটা নিলেই হবে

Negative সংখ্যা বোঝানোর কয়েকটি পদ্ধতি:

① 2's complement: $-2 \Rightarrow 2 \rightarrow 0010 \rightarrow 1101 \Rightarrow 1110$

1's com ↙ 2's com ↙ ⇓ -2

2's complement eqn;

$-n = 2^k - n$; K = number of bits represented

$-2 = 2^4 - 2 = 16 - 2 = 14 = 1110$

Bin

Float থেকে ১০ bit → bin → লিখ?

⇒ **Float**

Float to binary

2.5 = 10.1

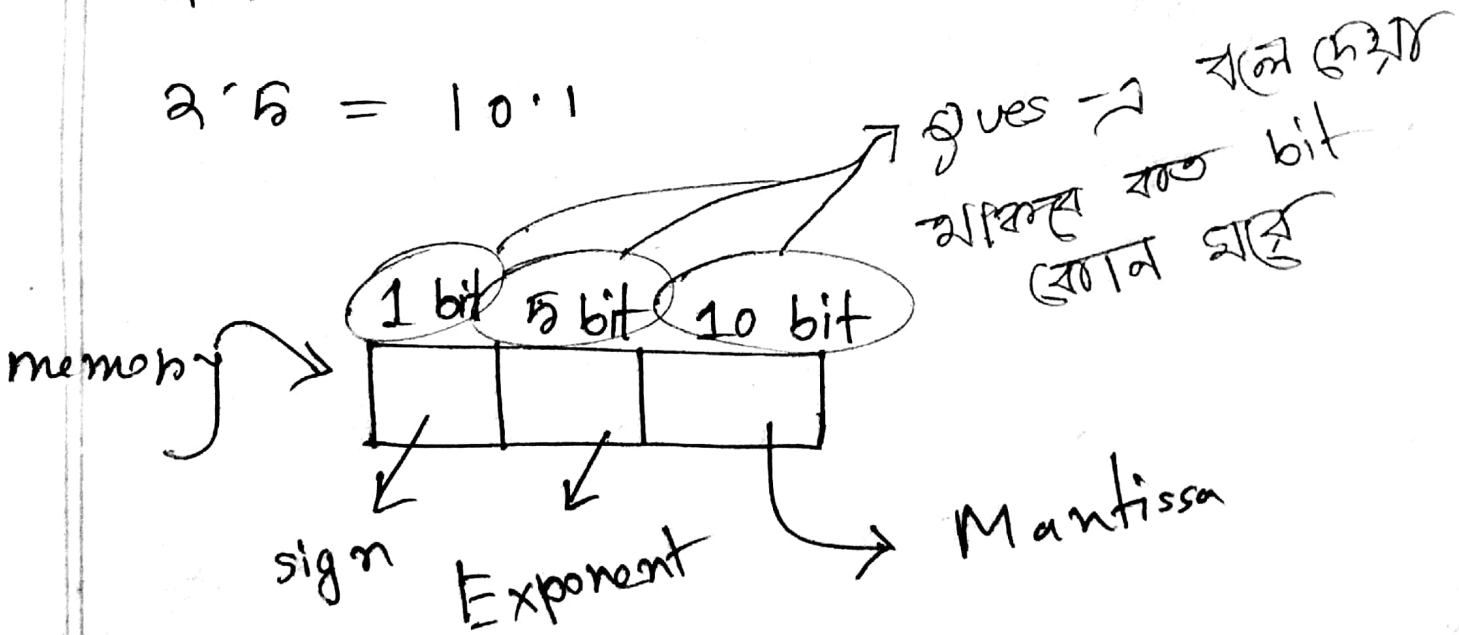
2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
16	8	4	2	1	.5	.25	.125	.625

13 = 0 1 1 0 1 1

0 1 1 0

2.4 = - - -

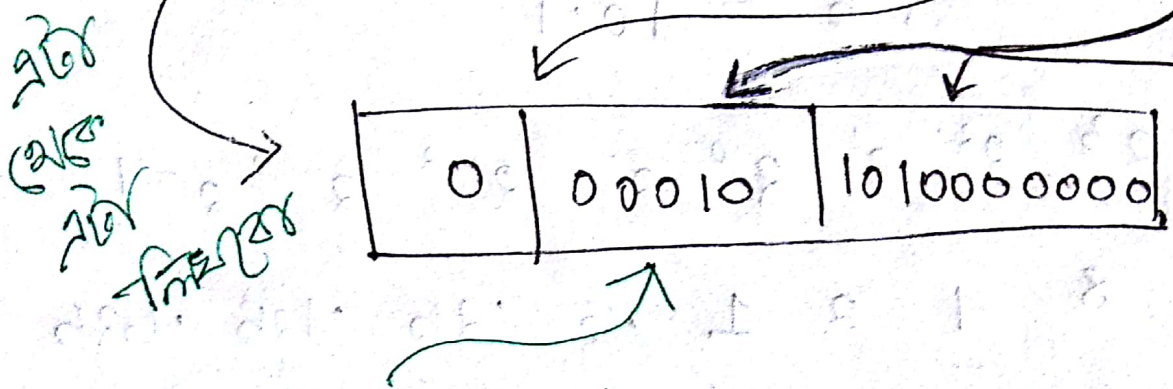
2.5 = 10.1



$$12.35 = 1235 \times 10^{-2}$$

$$= .1235 \times 10^{2}$$

$$2.5 = 10.1 = + .101 \times 2^{2}$$



power minus হলে 2's complmnt নিতে হবে।

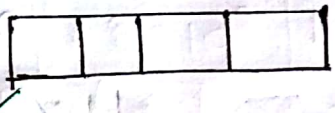
② 1's complement: $-2 \Rightarrow 2 \Rightarrow 0010 \Rightarrow 1101$

$1101 \Rightarrow -2$

$-n = 2^k - n - 1$

$-2 = 2^4 - 2 - 1 = 16 - 2 - 1 = 13 = 1101$

③ Sign Magnitude:

$-2 \Rightarrow$  $\Rightarrow 1010$

প্রথম bit sign এর জন্য

0 মানে pos

1 " neg

$-n = 2^{k-1} + n$

$-2 = 2^{4-1} + 2 = 8 + 2 = 10 = 1010$

④ Offset: 0000 মানে -8

⋮

1111 মানে 0

$2^{k-1} - n$

$-2 = 2^3 - 2 = 8 - 2 = 6$

$= 0110$

$= -2$

Ques || চার রকম পদ্ধতির description? eqn দিয়ে value ?

Ques || 4 bit দিয়ে যতগুলো সংখ্যা লিখা যায় সবগুলো ছোট থেকে বড় সাজাও −, (+ve) 3 (−ve) সব একসাথে আছে.

Lec - 30

Chapter - 3

page - 92, 94 Ex - 2, 6, 7 (a)

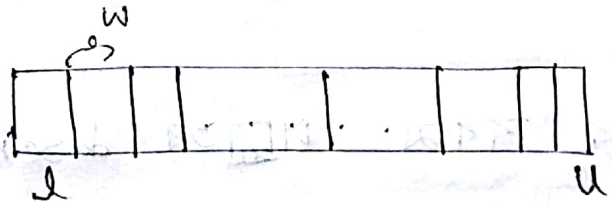
page — 117, 118 Ex - 4, 5

Upper triangular matrix

বর্গ-এ lower

Find N dimensional array, memory position

A: array [l:u] of items of length w



$(u-l+1)w$ ← memory তে জায়গা নিবে

ঘর আছে $u-l+1$

Mark ← distribution

data struc ⇒ definition

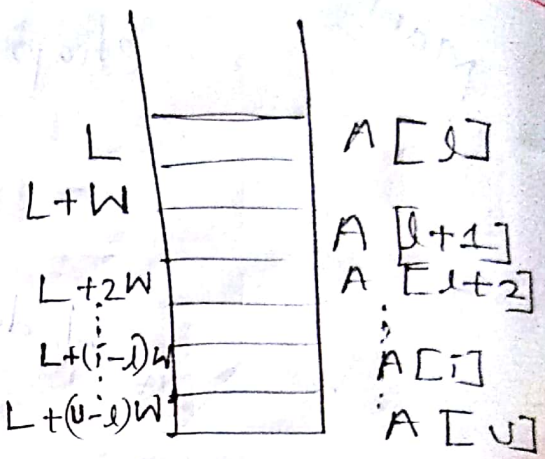
chap 1 + 3 → 1 or ques

today's lecture.

a)

b)

c)



$$loc(A[i]) = loc\ A[l] + (i-l)w \quad \text{---- (1)}$$

1 dimen. array

A[1,1]	A[1,2]	...	A[1,m]
A[2,1]	A[2,2]	...	A[2,m]
:	:	:	:
A[n,1]	...	...	A[n,m]

two dimensional array

$n \times m \times w$

$$loc(A[i]) = loc(A[l]) + (i-l)W \quad \text{---} (1)$$

$$loc(B[i]) = loc(B[1]) + (i-1)mW$$

2 dimensional ⇐ 1 → convert

$$loc(A[i,j]) = loc(B[1]) + (i-1)mW + (J-I)W$$

$$= loc(A[1,1]) + (i-1)mW + (j-1)W \quad \text{---} (11)$$

B

mw

$$\begin{bmatrix} B[1] \\ B[2] \\ \vdots \\ B[n] \end{bmatrix} \Leftarrow i$$

N dimensional

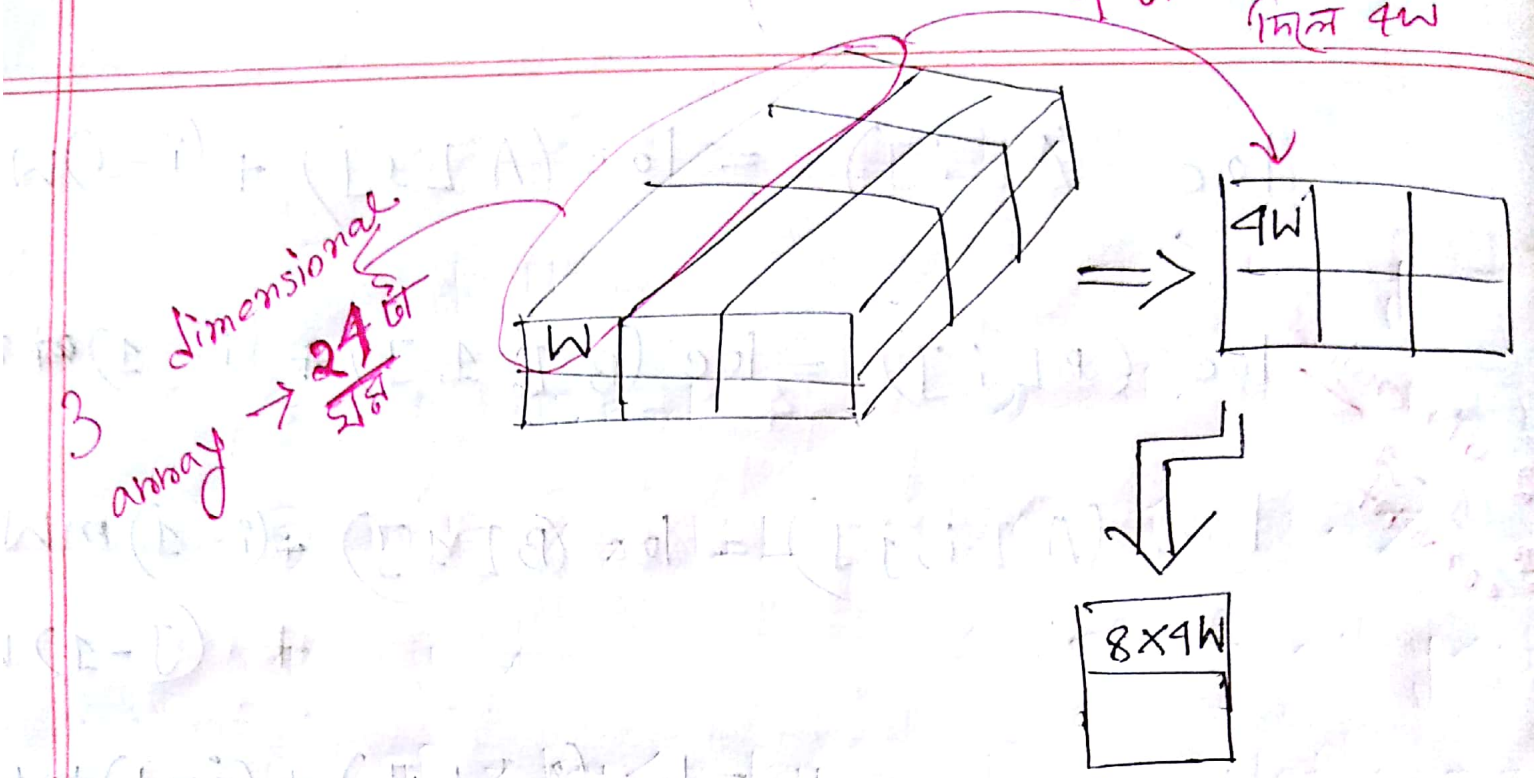
$$A[l_1 : U_1, l_2 : U_2, \ldots l_n : U_n]$$

$$loc(A[i_1, i_2, \ldots i_n]) = loc(\hat{A}[i_1, i_2, \ldots i_{n-1}]) + (i_n - l_n)W$$

$$= loc(\hat{\hat{A}}[i_1, i_2, \ldots i_{n-2}]) + (i_{n-1} - l_{n-1})(U_n - l_n + 1)W + (i_n - l_n)W$$

$$= \ldots + W\{ \quad + \quad \}$$

একটুকুই লিখতে হবে



Successor → আগেরটা
predecessor → পরেরটা

Lec - 31

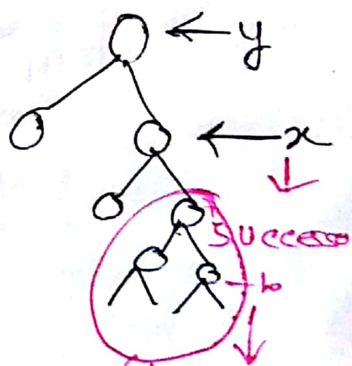
Binary Search Tree

chapter 5 এর সাথে যাবে, বাকী searching chapter
search এর সাথে যাবে।

Tree - Minimum (x)

(Max)

1) while x.~~left~~ (Right) ≠ NIL
2) x = x.~~left~~ (Right)
3) return x



immediate পরের value
ছোট থেকে r8 side-এ যেটা successor।
r8 side না থাকলে উপরে উঠে উঠে যেটা r8-এ পাবো সেটা।

Tree - Successor (x)

(Predecessor → opposite)

1) if x.right (left) ≠ NIL
2) return Tree - Minimum (Max) (x.right (left))
3) ~~else~~ y = x.p
4) while y ≠ NIL and x == y.right (left)
5) x = y
6) y = y.p
7) return y

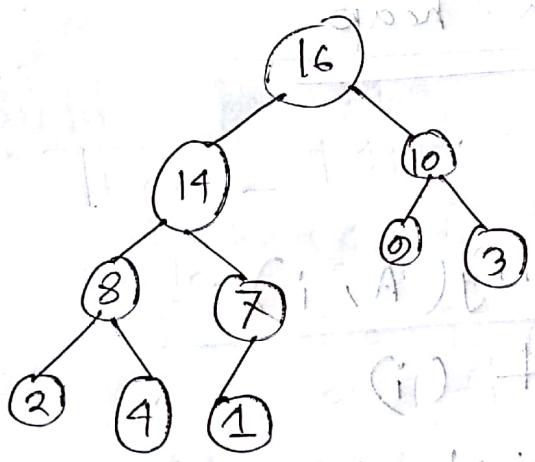
Tree_Insert (T, z)

```
1)  y = NIL
2)  x = T.root
3)  while x ≠ NIL
4)      y = x
5)      if z.key < x.key
6)          x = x.left
7)      else x = x.right
8)  z.p = y
9)  if y == NIL
10)     T.root = z
11) else if z.key < y.k
12)     y.left = z
13) else y.right = z
```

Lec – 32

Heap Sorting

16	14	10	8	7	9	3	2	4	1



Parent (i)

return $\lfloor i/2 \rfloor$

left (i)

return 2i+1

#Ques: 3. no node এর left right child কোথায় যায়?

index

#array কে Max-heap বানাও

A[i], A[2i], A[2i+1] এর মধ্যে যে বড় তাকে parent বানাবো

Max heap

যে heap এ parent এর value child এর থেকে বড় বা সমান,

Min - heap

Do it yourself

Max - heapify(A, i)

1) l = left (i)
2) r = right (i)
3) If l < A.heapsize and A[i] > A[l]
4) Largest = l
5) else largest = i
6) if r ≤ A heapsize and A[r] > A[largest]
7) largest = r

8) if $i \neq$ largest

9) exchange A[i] with A[largest]

10) Max_heapify (A, largest)

leaf নিয়ে করবে

Build Max_heap (A)

1) A.heapsize = A.length

2) for i = [A.length/2] down to 1

3) Max_Heapify (i)

Heap sort

1) Build_Max_Heap (A)

2) for i = A.length down to 2

3) exchange A[i] with A[1]

4) A.heapsize = A.heapsize - 1

5) Max_heapify (A, 1)