24.04.16

CSE-2103 : Data Structures

chap-1

A data structure can be defined informally as an organized collection of values and a set of operations on them.

* Data structures have three components : (describe)

i) A set of functions — function

ii) A storage structure

iii) A set of algorithms — set of instruction

[← assign করার
K ← n , n এর value
K তে assign করা]

K ← n

while ~~while~~ K ≠ 0 do

t ← 0

for j = 1 to K−1 do

if $x_j > x_{j+1}$ then

$x_j \longleftrightarrow x_{j+1}$

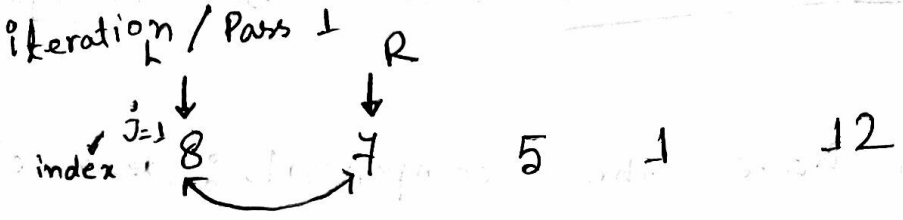
t ← j

K ← t

chapter-8 - Sorting

1. Bubble sort / Transposition sorts :

iteration / Pass 1



t = 1 : 7 8 5 1 12

t = 2 : 7 5 8 1 12

t = 3 : 7 5 1 8 12

t = 3

iteration / Pass 2 :

7 5 1 8 12

t = 1 : 5 7 1 8 12

t = 2 : 5 1 7 8 12

iteration / Pass 3 :

5 1 7 8 12

t = 1 : 1 5 7 8 12

26.04.16

$k \leftarrow n$

while $k \neq 0$ do

$t \leftarrow 0$ ↗ comparison number

for $j = 1$ to $k-1$ do

if $x_j > x_{j+1}$ then ⟶ $x[j] > x[j+1]$

$x_j \leftrightarrow x_{j+1}$ → swaping

$t \leftarrow j$

~~$k \Leftarrow t$~~

$k \leftarrow t$

2. Insertion Sort :

$x_0 \leftarrow -\infty$

for $j = 2$ to n do

$i \leftarrow j - 1$

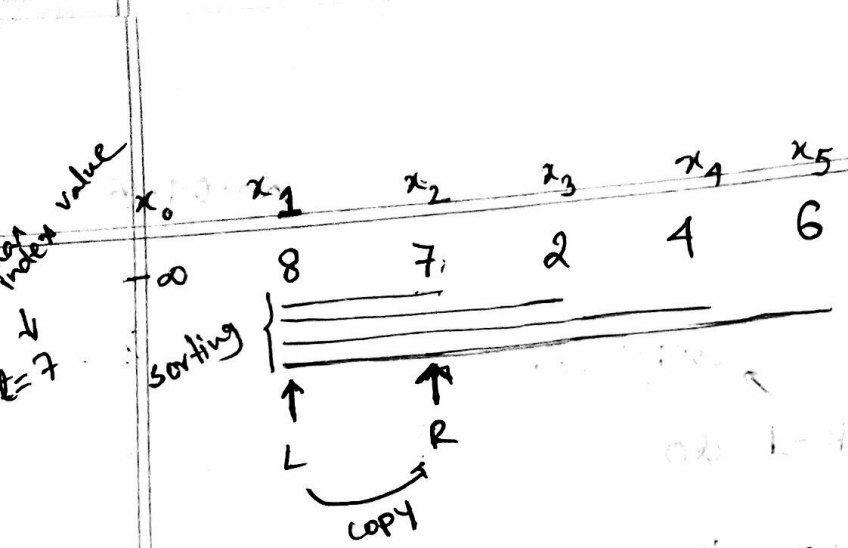
$t \leftarrow x_j$

while $t < x_i$ do

$x_{i+1} \leftarrow x_i$

$i \leftarrow i - 1$

$x_{i+1} \leftarrow t$



$-\infty$ 8 8 2 4 6

$-\infty$ 7 8 2 4 6

$-\infty$ 7 8 8 4 6

$-\infty$ 7 7 8 4 6

$-\infty$ 2 7 8 4 6

$-\infty$ 2 7 8 8 6

$-\infty$ 2 7 7 8 6

$-\infty$ 2 4 7 8 6

$-\infty$ 2 4 7 8 8

$-\infty$ 2 4 7 7 8

$-\infty$ 2 4 6 7 8

bubble / insertion sort pass count:

27.04.16

3. Selection Sort : (Simple selection Sort)

77 33 44 11 88 22 66 55

Algorithm

for $j = n$ to 2 by -1 do (decrement)

$t \leftarrow 1$

for $k = 2$ to j do

if $x_t < x_k$ then

$t \leftarrow k$

$x_j \leftrightarrow x_t$

1st

77 33 44 11 88 22 66 55

77 33 44 11 55 22 66 88

2nd.

66 33 44 11 55 22 77 88

3rd

22 33 44 11 55 66 77 88

4th

22 33 44 11 55 66 77 88

5th

22 33 11 44 55 66 77 88

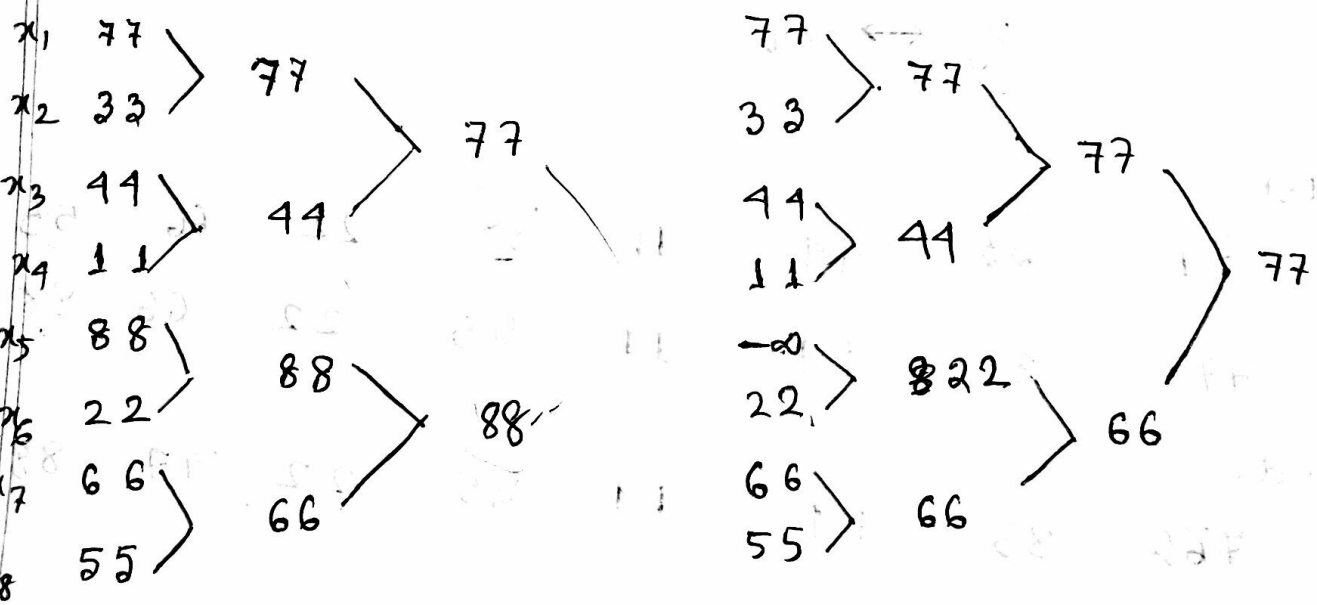
6th

22 11 33 44 55 66 77 88

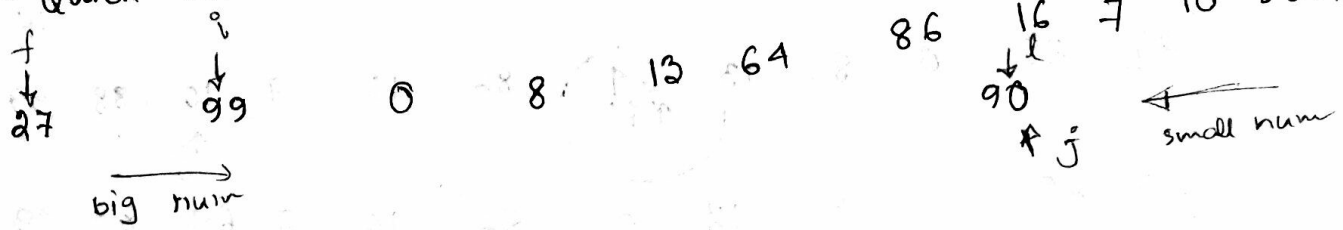
7th

11 22 33 44 55 66 77 88

4. Knockout tournament :



4. Quick Sort



Algorithm (divide and conquer)

QUICK SORT (f, l) | f = 1, l = n

if f < l then

i ← f+1

while $x_i < x_f$ do

i ← i+1

j ← l

while $x_j > x_f$ do

j ← j−1

while i < j do

$x_i \leftrightarrow x_j$

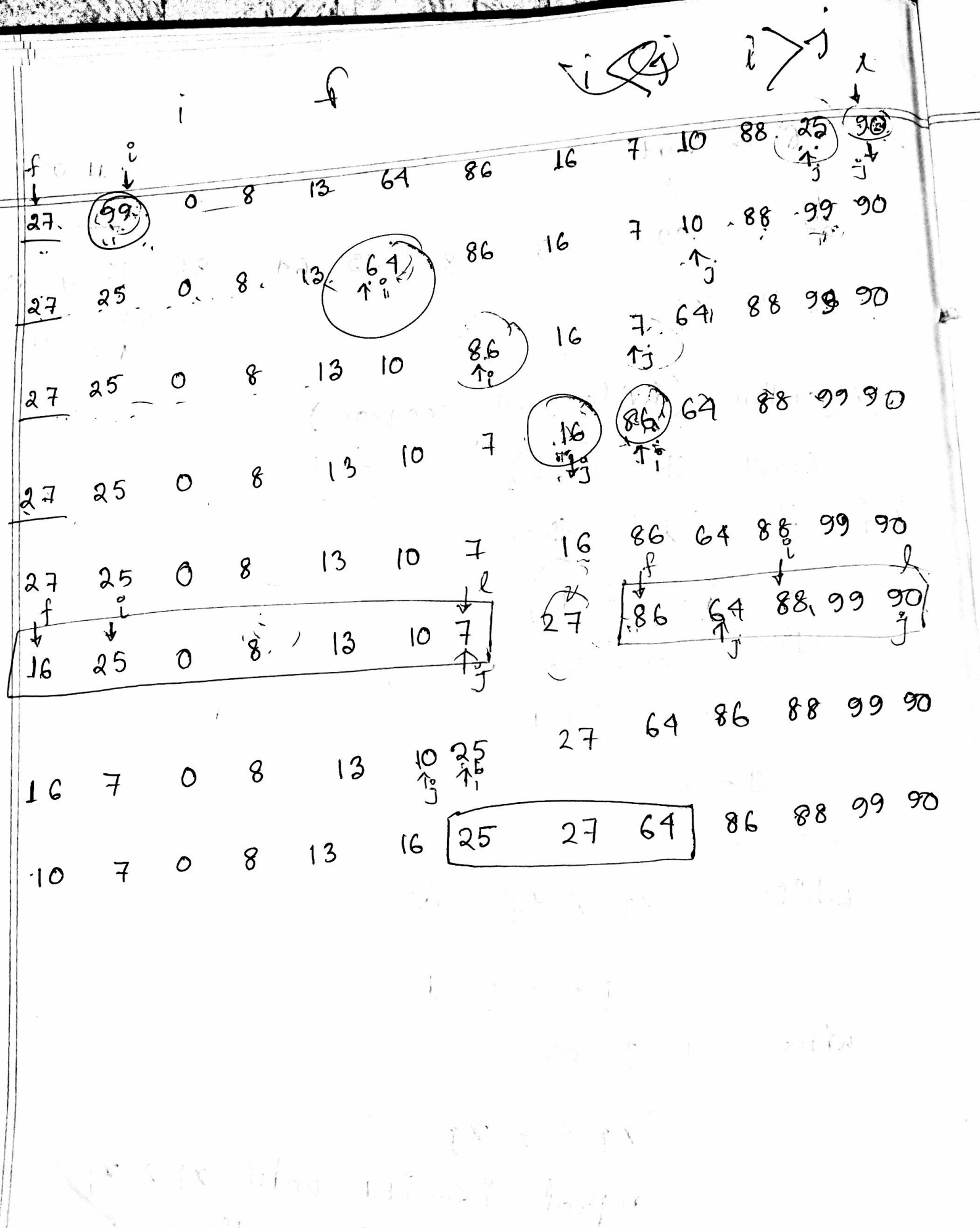
repeat i ← i+1 until $x_i \geq x_f$

repeat j ← j−1 until $x_j \leq x_f$

$x_j \leftrightarrow x_f$

QUICK_SORT (f, j−1)

QUICK SORT (j+1, l)



Merge Sort

15.05.16

Algorithm

```
Merge-Sort (A, P, r)
    if  p < r  then
        q ← floor ( (P+r)/2 )
        Merge-Sort (A, P, q)
        Merge_Sort (A, q+1, r)
        Merge (A, P, q, r)
```

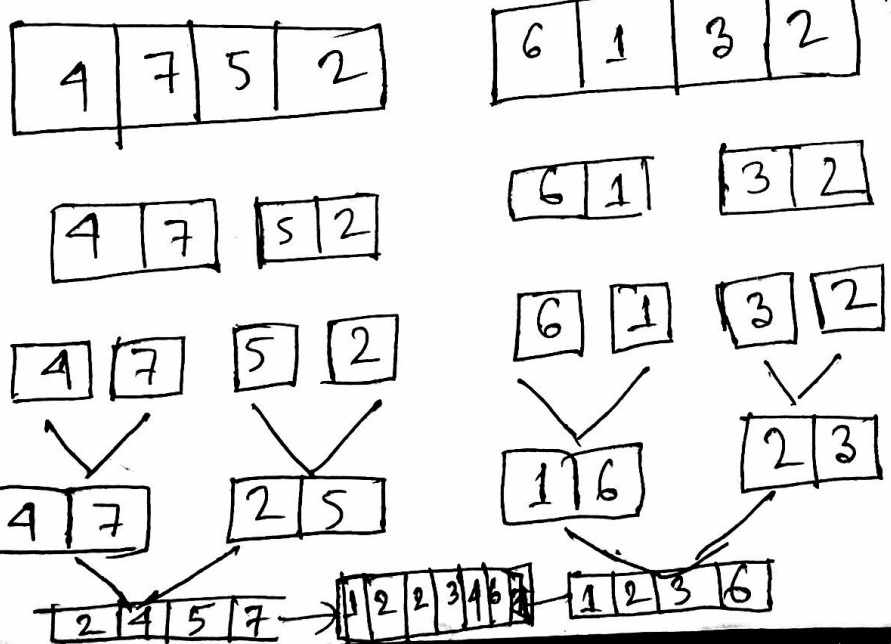
~~Merge Sort~~

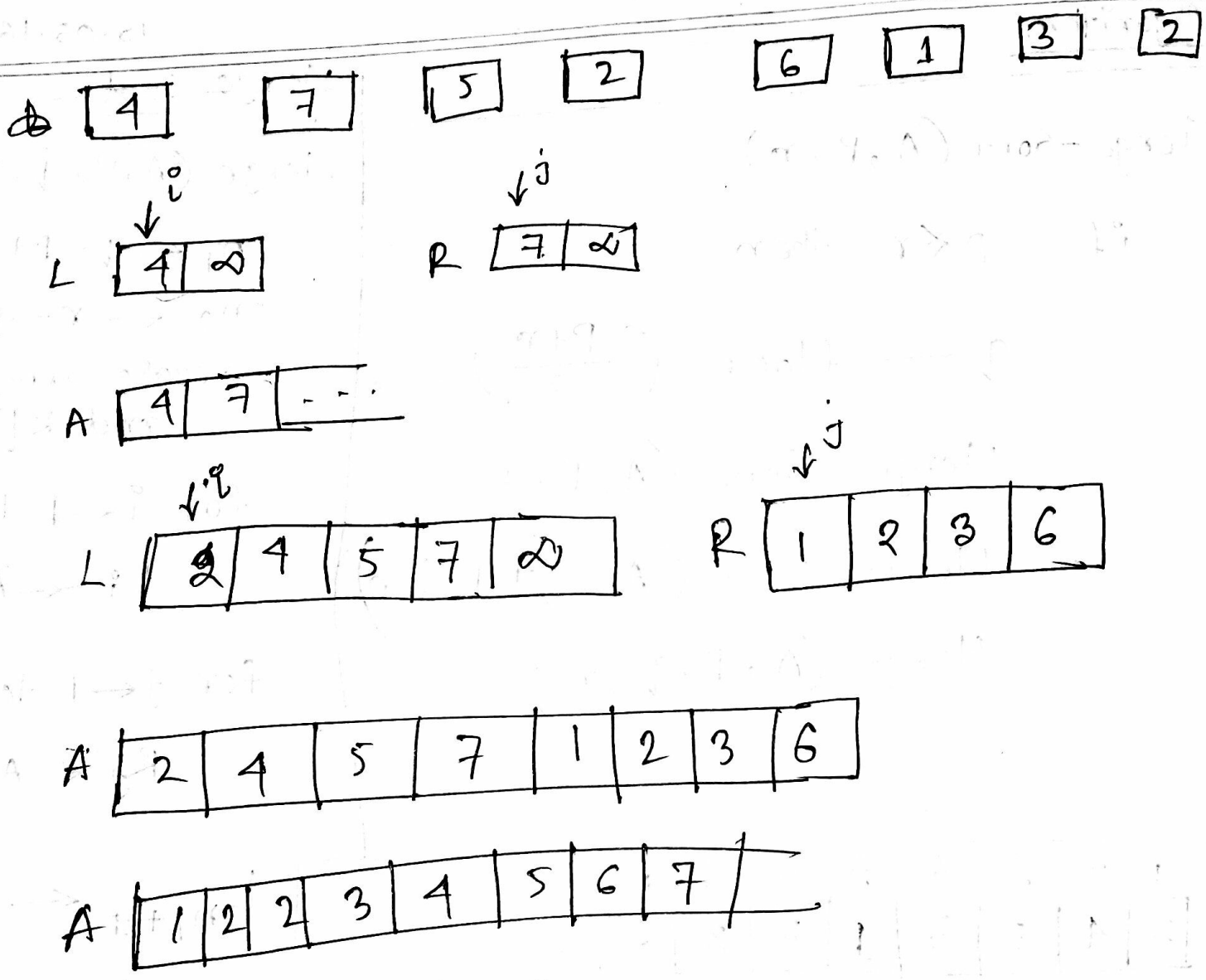
```
Merge (A, P, q, r)
    n1 ← q - P + 1
    n2 ← r - q
    create arrays L[1 .. n1+1]
        and R[1 .. n2+1]
    for i ← 1 to n1 do
        Li ← A(P+i-1)
    for j ← 1 to n2 do
        Rj ← A(q+j)
    Ln1+1 ← ∞
    Rn2+1 ← ∞
    i ← 1
    j ← 1
    for k ← P to r do
        if Li ≤ Rj then
            Ak ← Li
            i ← i + 1
        else Ak ← Rj
            j ← j + 1
```

	1	2	3	4	5	6	7	8
A	2	4	5	7	1	2	3	6

	1	2	3	4	5	6	7	8
A	4	7	5	2	6	1	3	2
	P			q				r

A → index (indicate mtd)




4 7 5 2 6 1 3 2
i
j
L 4 ∞
R 7 ∞
A 4 7 . . .
i
j
L 2 4 5 7 ∞
R 1 2 3 6
A 2 4 5 7 1 2 3 6
A 1 2 2 3 4 5 6 7

Quiz-1
↳ Next week Wednesday : 25.06.2016

19.05.16

Radix Sort

* positive number এর জন্য করতে হবে

use the fields link$_1$, link$_2$, ... link$_n$ to form $x_1, x_2, \ldots, x_n$ into an input Queue, Q → 1 টা queue

⇒ linked list

for j = 1 to P do → number of pass ≡ total number of digit

initialize each of the queues $Q_0 \ldots Q_{n-1}$ to be empty → another queue

while Q not empty do

$X \Leftarrow Q$

let $X = (x[P], x[P-1], \ldots, x[1])$ $x[x[1], x[2], x[3]]$

$Q_{x[j]} \Leftarrow X$ ⟶ $Q_{x[j]}[Q_0 \text{—} Q_9]$

concatenate ~~Concatenate~~ queues $Q_0 \ldots Q_{r-1}$ together to form the new Queue, Q

↓

pop করে value concatenate করে Q তে রাখা

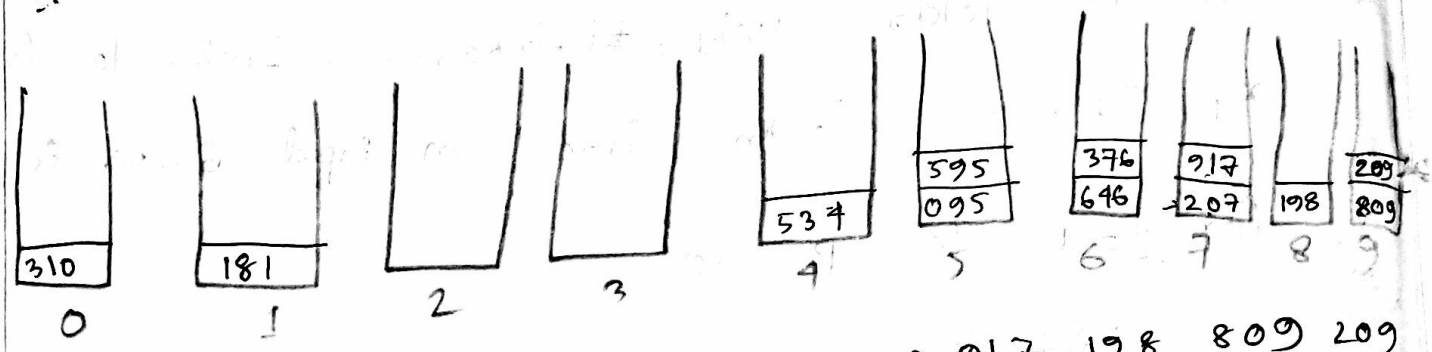
* negative, positive both এর জন্য Algorithm.

Right-most side 의 digit check 하기

total number of digit 이 2개, pass 도 2번!

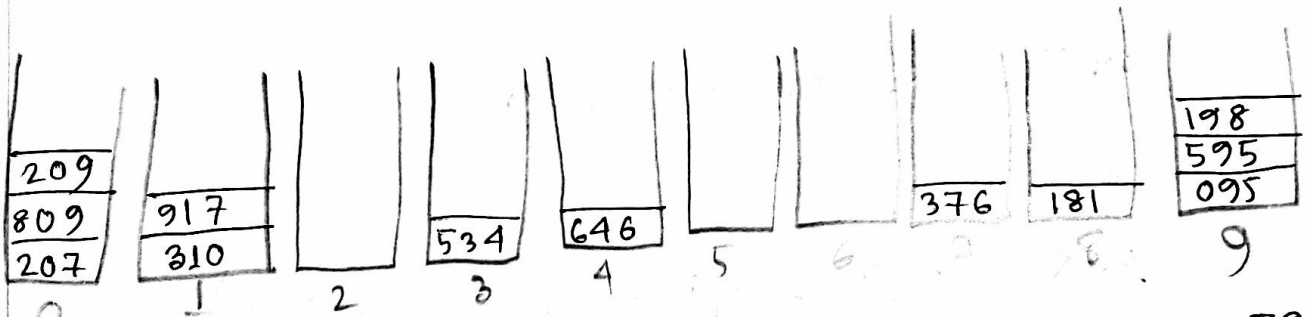
207, 095, 646, 198, 809, 376, 917, 534, 310, 209, 181, 595

Pass 1°:



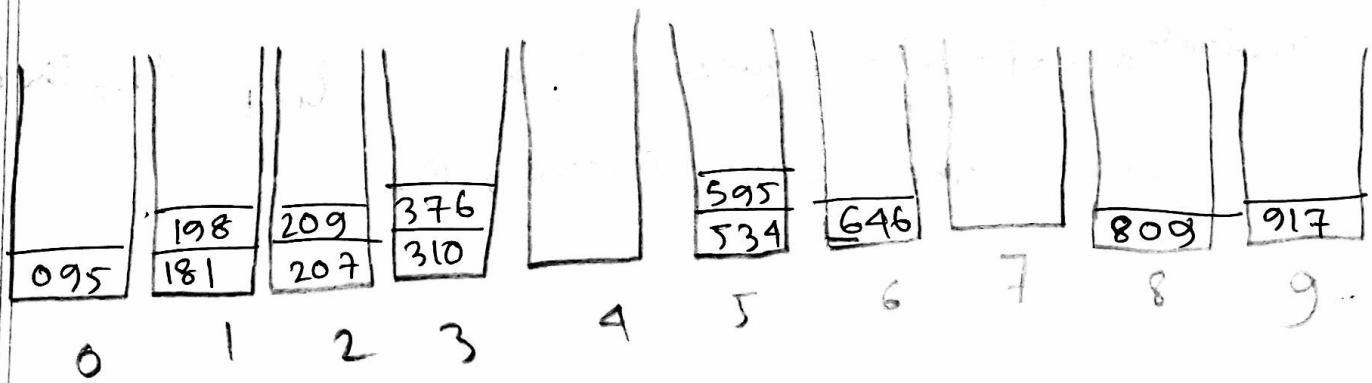
310 181 534 095 595 646 376 207 917 198 809 209

Pass 2°:



207 809 209 310 917 534 646 376 181 095 595 198

Pass 3°:



095 ~~198~~ 181 198 207 209 310 376 534 595 646 809 917

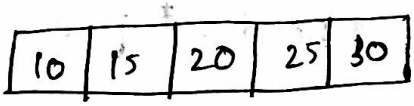
***** (imp) → 3, 4 set Ques

18.05.16

Chapter-4

Lists

A data structure list is a finite sequence of elements



→ link এর মাধ্যমে connected

next element এর address থাকবে

→ connection বুঝাতে use করা হয়।

* list < data portion / তথ্য info
  Link Portion

→ general format

* last list-এ Link = Null থাকবে।

* first node এর address কে ~~per~~ head pointer এ point করতে হবে।

* List → dynamically memory allocate হয়।

* disadvantage → traverse problem

$P_1 = (struct\ node*)malloc(sizeof(struct\ node));$ → ঠিকানা return করবে।
→ ঠিকানা memory তে occupy করে।

type casting

$P_2 =$

$P_3 =$

P_4

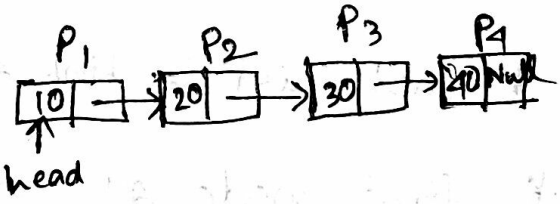
P_1 [info | link] , P_2 [info | link]

```
struct node
{
    int info;
    node *link; / struct node * link;

} *P1, *P2, *P3, *P4; *head, *P;

int main()
{ P1 = (Struct node*) malloc (sizeof(struct node));
  P2 =
  P3 =
  P4 =

head = P1;
  P1 -> info = 10;
  P1 -> link = P2;
  P2 -> info = 20;
  P2 -> link = P3;
  P3 -> info = 30;
  P3 -> link = P4;
  P4 -> info = 40;
  P4 -> link = NULL;
* P = head;
```




```
*
while (P != NULL)
{
    printf ("%d", P->info);
    P = P -> link;
}
return 0;
}
```

Algorithm: Insertion of a new record in a sorted linked list

29.05.2016

if L = nil or number(new) < number(L) then

Link(new) ← L

L ← new

* instruction

else

pred ← L

p ← link(L)

while p ≠ nil and number(p) < number(new) do

pred ← p

p ← link(p)

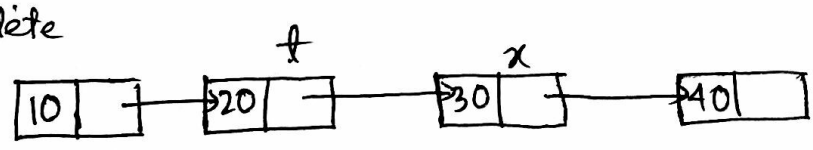
link(new) ← link(pred)/p

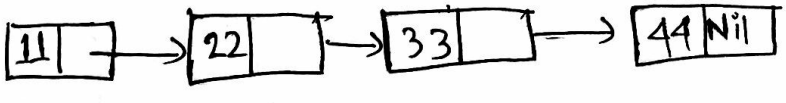
link(pred) ← new

```
Pre = head ;
curr = head → link
if(curr→ info == 30)
{
   pre→ link = curr→ link;
}
pre ← curr
curr ← curr→ link
```

Single linked list

Delete





t→link = link(t)

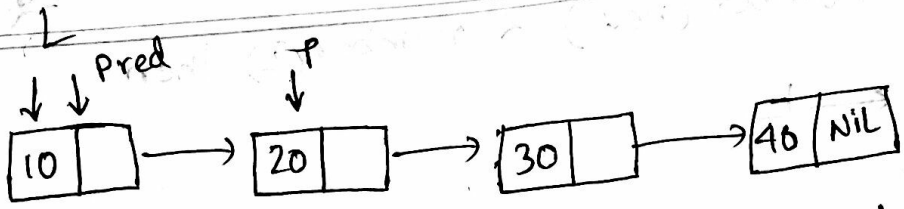
link(t), ← lin(x) → instru

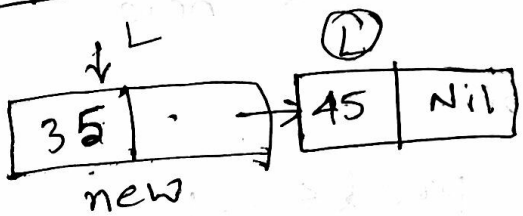
delete operation

move

link(t) ← link(y)

link(y) ← link(x)

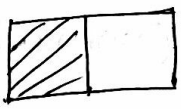


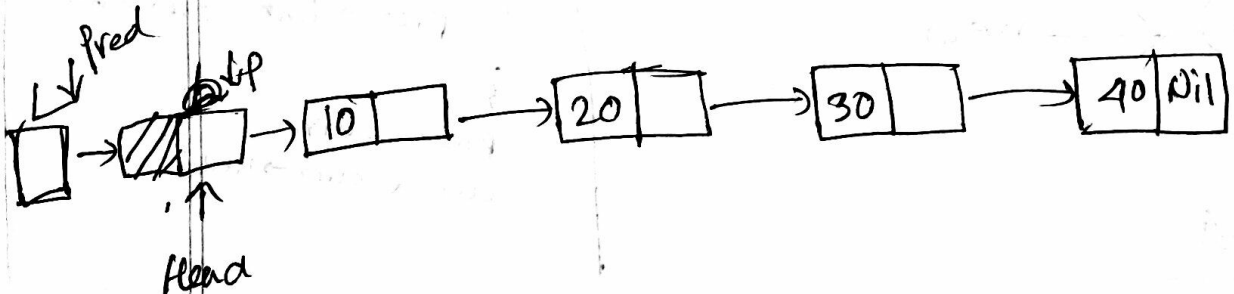


Dummy Element (define + advantage)

H.W

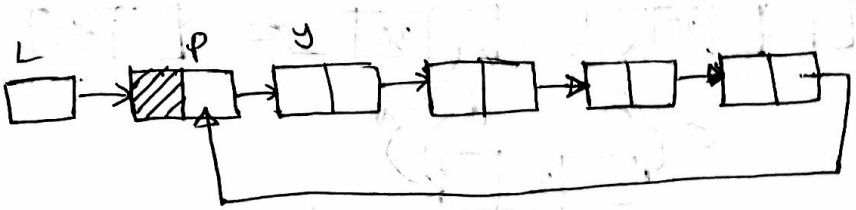
Dummy → just link
⇒ info

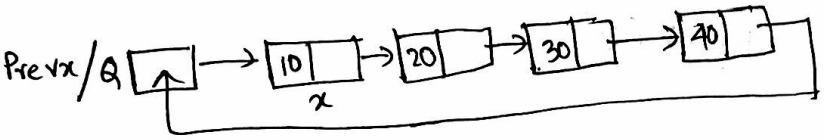




24.05.2016

Circular linked list

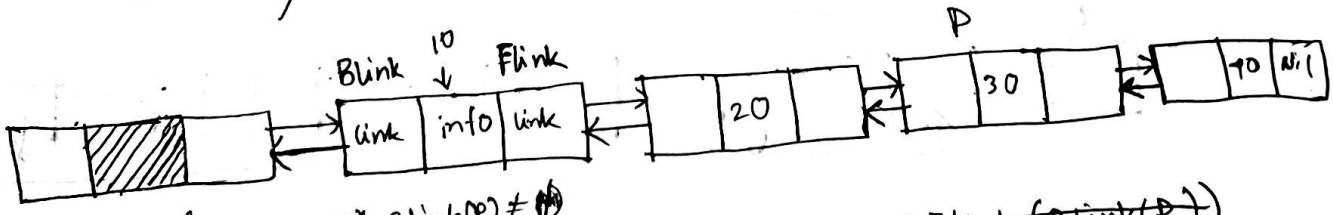




malloc

```
Q ← getcell
y ← link(P)
Prevx ← Q
while y ≠ P do
    x ← getcell
    info(x) ← info(y)
    link(Prevx) ← x
    y ← link(y)
    prevx ← x
link(Prevx) ← Q
```

Dubly / Duble linked list



flink – Forward link

Blink – Backward link

~~Flink(Blink(P))~~

~~if (Blink ≠ Nil)~~ if (Blink(P) ≠ Nil)

~~Flink(Blink(P)) ⇔ Flink(P)~~

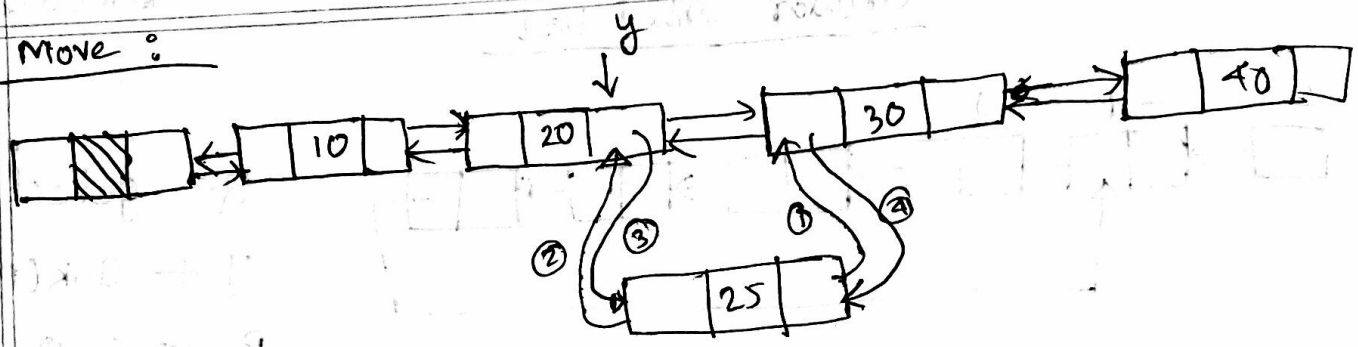
if (Flink(P) ≠ Nil)

Blink(Flink(P)) ⇐ Blink(P)

Delete:

```
if (Blink(P) ≠ Nil) then
    Flink(Blink(P)) ← Flink(P)
if (Flink(P) ≠ Nil) then
    Blink(Flink(P)) ← Blink(P)
```

Move :



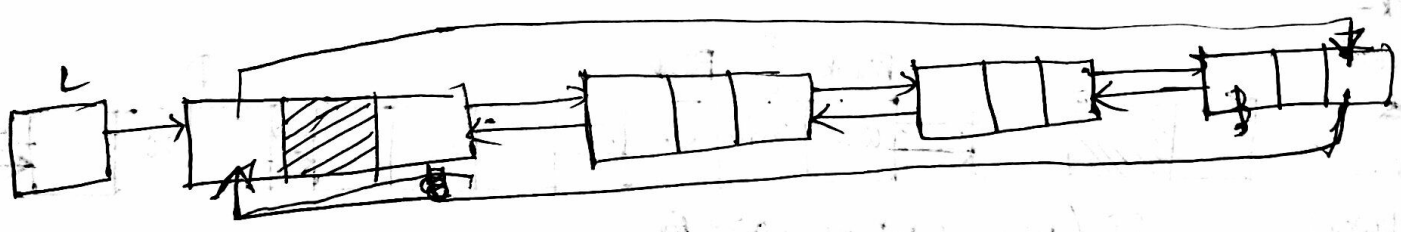
① Flink(new) ← Flink(y)

② Blink(new) ← y

③ Flink(y) ← new

④ Blink (Flink(y)) ← new

Dubly circular linked list



29:05.16

Stacks & Queues

H.W : sequential implementation

Linked Implementation

Stack

```
s ⇐ x :   l ← getcell
          info(l) ← x
          link(l) ← t
          t ← l
```

```
x ⇐ s :  if t = nil then
         underflow
              else
                   x ← info(t)
                   t ← link(t)
```

Queue

```
Q ⇐ x :  l ← getcell
         info(l) ← x
         link(l) ← nil
         link(r) ← l
         r ← l
```

```
x ⇐ Q :  t ← link(f)        (1st node address)
         if t = nil then
         underflow
              else
                   x ← info(f)
                   link(f) ← link(t)
                   if link(f) = nil
                   then r ← f
```

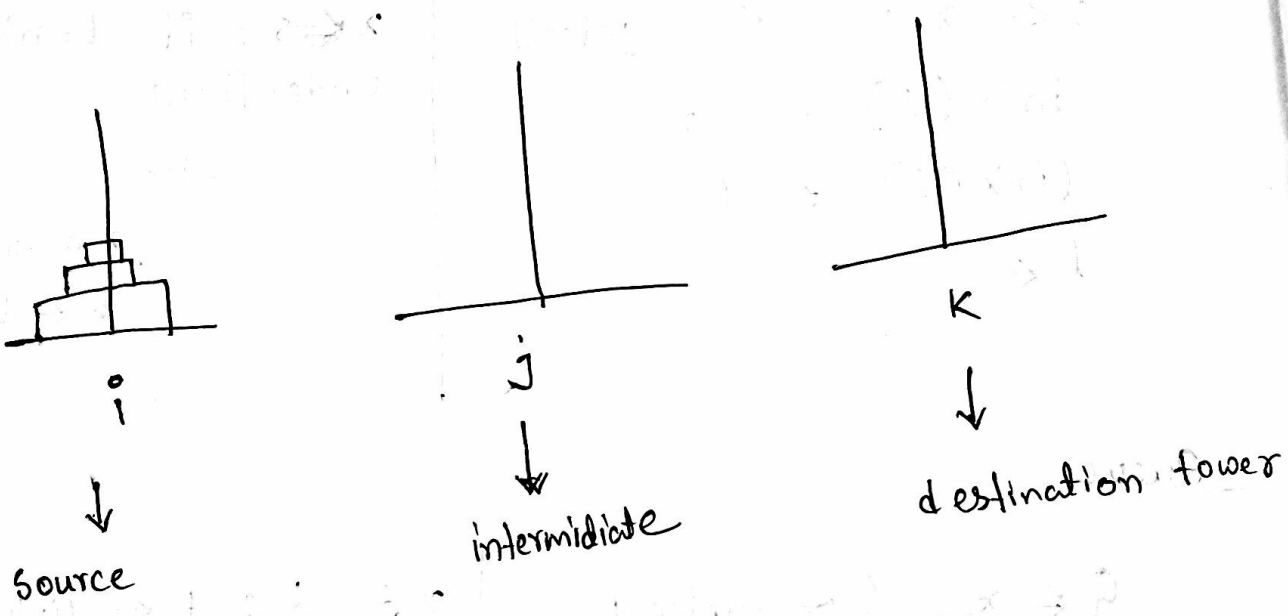
31-05-16

Application of Stack and Queues

Stacks & Recursion

Towers of Hanoi

i → k
i → j
k → j
i → k
j → i
j → k
i → k



1st : $n-1 \rightarrow j$

$H_n = 2H_{n-1} + 1$

$H_1 = 2H_0 + 1$

$= 0 + 1 = 1$

$H_2 = 2H_1 + 1$

$= 2 + 1 = 3$

$H_3 = 2H_2 + 1 = 7$

HANOI (n, i, j, k)

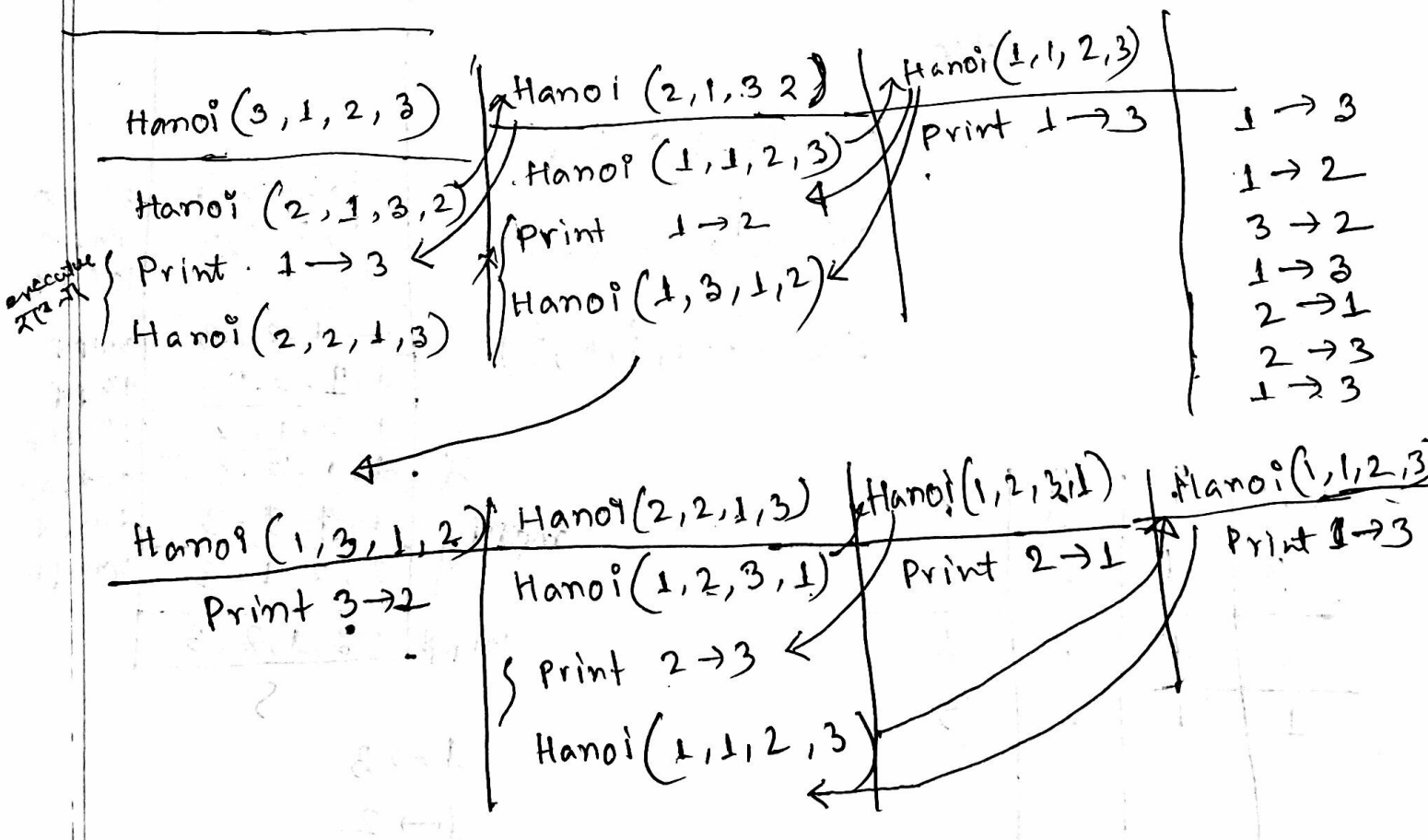
if n=1 then move the top disk from pole i to k

else

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

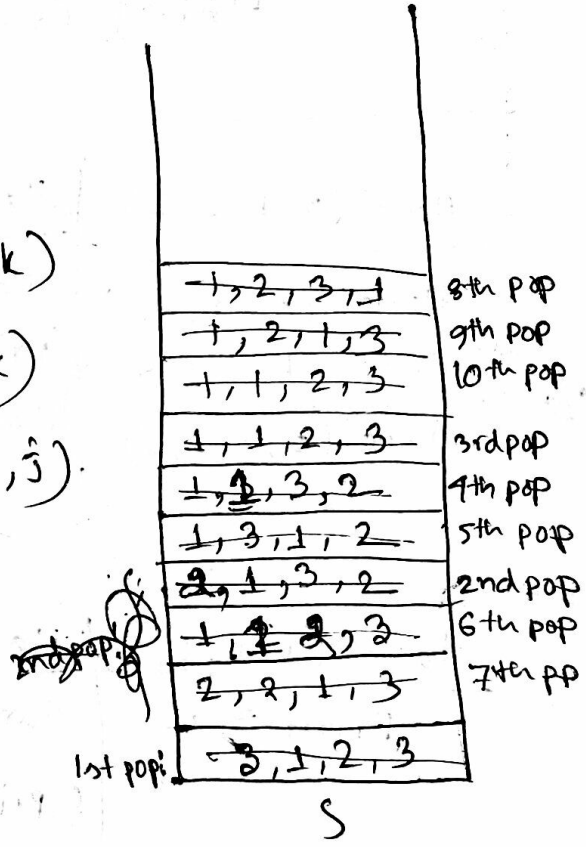
move the top disk from pole i to pole k (Print)

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



* Stack algorithm for the towers of Hanoi problem

```
S ⟵ empty stack
S ⟸ (n, 1, 2, 3)
while S ≠ empty do
        (n, i, j, k) ⟸ S        ← Pop
        if n = 1 then move the top disk from Pole i
                to Pole k
        else
                S ⟸ (n-1, j, i, k)
                S ⟸ (1, i, j, k)
                S ⟸ (n-1, i, k, j)
```



n	i	j	k
3	1	2	3
2	1	3	2
1	1	2	3

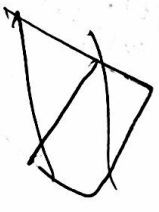
1 → 3
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3

* Stacks & Arithmetic Expressions

Algorithm

Conversion an infix expression to post-fix Expression

```
S ← Empty stack
S ⇐ "#"
j ← 0
i ← 0
while top(S) ≠ "$" do
        i ← i + 1
        case
        ① E[i] is an operand
                j ← j + 1
                P[j] ← E[i]
        ② E[i] = "(" : S ⇐ E[i]
        ③ E[j] = ")" :
                x ⇐ S
                while x ≠ "(" do
                        j ← j + 1
                        P[j] ← x
                        x ⇐ S
```



* infix to prefix → H.W

④ E[i] is an operator :

while Prec(E[i]) ≤ Prec(top(s)) do

$j \leftarrow j+1$

$P[j] \Leftarrow S$

$S \Leftarrow E[i]$

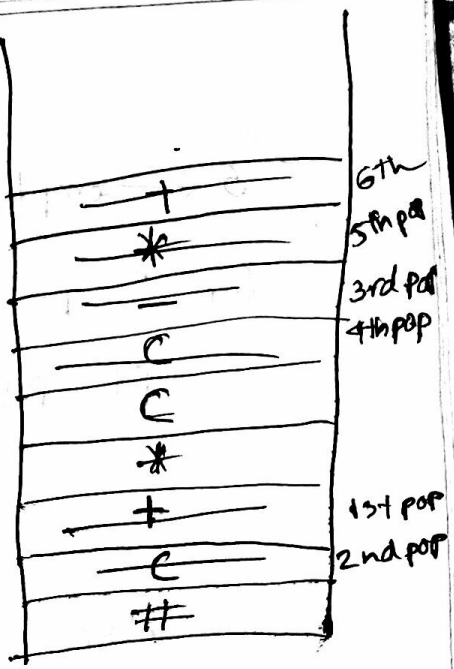
Character	Precedence
#	0
$	1
C	2
+, −	3
*, /	4

stack এ 1st # insert করতে হবে

$ Expression শেষ বুঝাতে

* Infix to Postfix:

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



$(A+B) = AB+ \;/\; +AB$

$((C-D)*E+F) = (CD-)*E+F \;/\; (-CD)*E+F$

$= (CD-E*)+F \quad = (*-CDE)+F$

$= CD-E*F+ \quad = +*-CDEF$

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

$= (AB+)*(CD-E*F+) \;/\; (+AB)*(+*-CDEF)$

$= AB+CD-E*F+* \;/\; *+AB+*-CDEF$

E

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

i=1

P

A	B	+	C	D	-	E	*	F	+	*

01.06.16

Algorithm: Evaluation of Postfix Expression

S ← empty stack

for i = 1 to n do

if P[i] is an operand then

S ⇐ P[i]

else

y ⇐ S

x ⇐ S

S ⇐ Value of the operator P[i] applied to x and y

P	A	B	+	C	D	−	E	*	F	+	*

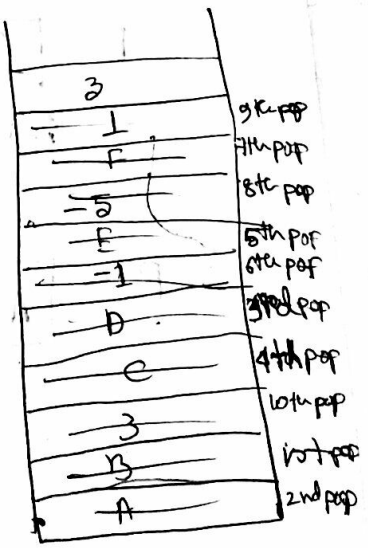
(A+B) * ((C−D) * E + F))

= 3 * (1)

= 3

y = B

x = A



Priority Queue

- It is an abstract data typ which is like a regular queue or stack data structure, but where additionally each element has a "Priority" associated with it.

- An element with high priority is served before an element with low priority

- If two elements have the same priority, they are served according to their order in the queue.

- Application: An array, linked list, Binary search tree

a

1	2	3	4
10	20	30	40
1	4	2	0

08.06.16

DEQUE (Double Ended QuEue)

↳ in point 2ই exit point 2ই 2 side থেকে insert করতে পারবো, 2side থেকে pop করতে পারবো.

Abstract data type

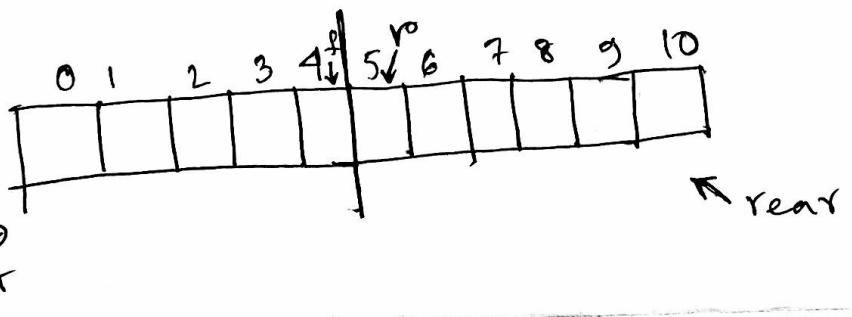
- Queue() creates a new ~~queq~~ queue i.e. empty. It needs no parameter and returns an empty queue.

— enqueue(item) adds a new item to the rear of the queue. It needs the item & returns nothing.

— dequeue() removes the front item. It ~~read~~ needs no parameters & returns the item.

— isEmpty() tests to see whether the queue is Empty. It needs no parameter & returns a boolean value.

— size() returns the number of items in the queue. It needs no parameter & returns ~~an~~ an integer.



Application:

- Bank (a queue of waiting patrons)
- Printer

Four Operations of DEQUE

Algorithm insert front:

1. Start
2. check the queue is full or not as if $(r = max-1)$ && $(f == 0)$ ~~$(r = max-1)$~~
3. If false update the pointer f as $f = f-1$
4. Insert the element at pointer f as $Q[f]$ = element.
5. stop.

Algorithm insert back:

1. start
2. check the queue is full or not as if $(r = max-1)$ && $(f == 0)$
3. Insert the element at pointer r as $Q[r]$ = element
4. If false update the pointer r as $r = r+1$
5. stop.

Algorithm Remove front :

1. start
2. check the queue is empty or not as if (f=r)
3. If false update the pointer f as f=f+1 and delete element position f as element = a[f]
4. If (f=r) reset pointer f and r as f=r=-1
5. stop.

12.06.16

4. Algorithm Remove Back of DEQUE

1. Start
2. Check the queue is empty or not as if (f==r). If yes, Queue is empty.
3. If false update pointer at position r as element = Q[r]
4. Update pointer r as r=r-1
5. If (f=r) reset pointer f and r as f=r-1
6. Stop

(Breadth first Search)

BFS (V, E, S)

for each u in V - {S} do

color [u] ← white

d[u] = ∞

π[u] = Nil

color [S] ← Gray

d[s] ← 0

π[s] ← NIL

Q ← empty queue

ENQUEUE (Q, S)

while Q is non-empty do

u ← DEQUEUE (Q)

for each v adjacent to u do

if color [v] ← white then

color [v] ← Gray

d[v] ← d[u] + 1

π[v] ← u

ENQUEUE (Q, v)

~~DEQUEUE (Q)~~

~~DEQUEUE~~ (Q)

Color [u] ← Black

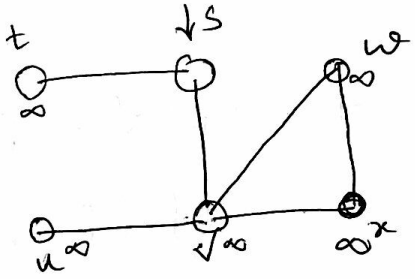
Graphs

H.W:

→ definition of graphs, edges, vertices

→ diff types of Graphs

→ Adjacent Matrix



white — not visited

Gray — Entry into queue

Black — visited

Wednesday — Quiz #2
linked list → priority queue

15.06.16

BFS (Print_Path)

Print_Path (G, s, v) ← starting node, → ending node (যে দুইটার মধ্যে node এর path বের করতে)

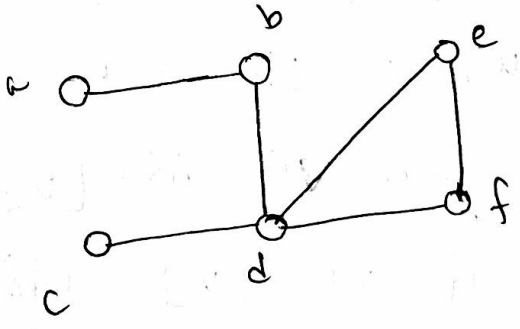
b-d-f

If v = s then
    print s

else if π[v] = NIL then
    print "no path exists" from "s" to "v"

else
    print_Path (G, s, π[v])
    print v ;

b-a-d-e-e-f



DFS (Depth - first - search)

DFS (G)

for each vertex $u \in V[G]$ do (vertex, vertex set)

color [u] ← white

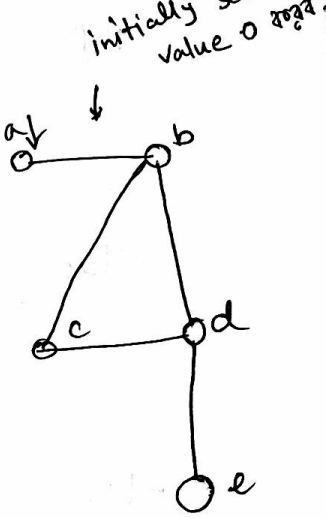
π[u] ← NIL

time ← 0

for each vertex $u \in V[G]$ do

if color [u] = white then

DFS_VISIT (u)



DFS_VISIT (u)

color [u] ← Gray

d [u] ← time + 1 (discover time)

for each $v \in$ Adj [u] do

if color [v] = white then

π [v] ← u

DFS_VISIT (v)

color [u] ← Black

f [u] ← time + 1 (finishing time)

19.06.16

Trees

— A tree is a collection of elements, of which one is the root and rest are partitioned into trees, called the subtrees of the root

— A Forest is a set of trees.

— The level of a node p in a tree T is defined recursively as 0 if p is the root of the tree T, otherwise the level of p is,

$$p = 1 + \text{level}(\text{Father}(P))$$

— The height $h(T)$ of a tree T is defined by

$$h(T) = \max_{\substack{\text{nodes} \\ p \text{ in } T}} \text{level}(P)$$

Binary Tree

Types :

— A Rooted Binary tree is a tree with a root node in which every node has at most two children

— A Full Binary tree is a tree in which every node other than the leaves has two children.

— A perfect Binary tree / complete Binary tree is a binary tree in which every level, except possibly the last, is completely filled and all nodes are as far left as possible.

— An infinite complete binary tree is a tree with levels, where for each level d the number of existing nodes at level d is equal to 2^d.

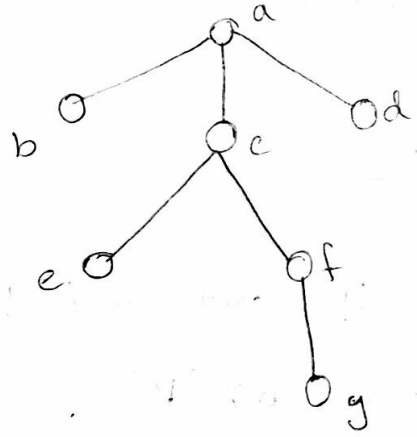
Advantages

— Trees reflect structural Relationships in the data.

— Trees are used to present hierarchies.

— Trees provide an efficient insertion & searching

— Trees are very flexible data, allowing to move subtrees around with minimum effort.



level of root = 0

level of f = 1 + level(father(f))

⇒ f = 1 + level(c)

= 1 + 1 + level(father(c))

= ~~+ + + +~~ = 2 + level(a)

= 2 + 0 = 2

leaves — b, d, e, g

* Binary search tree :

(i) node এরমধ্যে → maximum 2টি vertics থাকবে

(ii) parent node এর যেই value left এ, বড় value right এ

(iii) duplicate value থাকবে না

(iv) left node + right node

12.07.16

Searching

- Linear/sequential search techniques
- Binary search Techniques

Sequential Search Techniques

z = 16

1				
8	0	2	26	4

(a) Sequential_list_search

```
i ← 1
while i≤n and z ≠ xi do
        i ← i+1
if i≤n then found
else NOT FOUND
```

(b) linked-list search

info/data

```
P ← L
while P ≠ nil and z ≠ key(P) do
        P ← link(P)
if P ≠ nil then FOUND
else NOT FOUND
```

Lists in Natural Order

(a) Sequential list search in an order list

```
i ← 1
while  z > x_i  do
        i ← i+1
if  z = x_i  then  FOUND
else  NOT FOUND
```

(b) Linked-list search in an order list

```
P ← L
while  z > key(P)  do
        P ← link(P)
if  z < key(P)  then  FOUND
else  NOT FOUND
```

H.W – 7.4 (Interpolation Search)

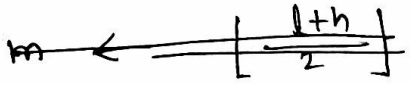
7.3 Binary Search

$l \leftarrow 1$

$h \leftarrow n$

found ← false

while $l \leq h$ and not found do



$m \leftarrow \left\lfloor \frac{l+h}{2} \right\rfloor$

case

$z < x_m$: $h \leftarrow m-1$

$z > x_m$: $l \leftarrow m+1$

$z = x_m$: found ← true

if found then FOUND

else NOT FOUND

→ input → sort → binary search

$z = -6$

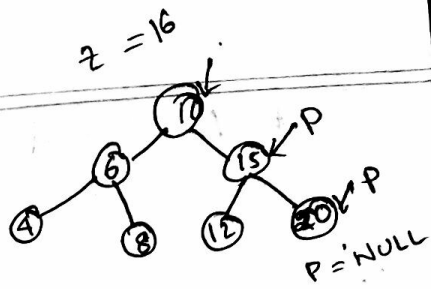
64	0	8	9	-1
-1	0	8	9	64

iter-1: $m = \left\lfloor \frac{l+h}{2} \right\rfloor = \left\lfloor \frac{1+5}{2} \right\rfloor = 3$

$z < x_m$ & $h = m-1$

iter-2: $m = \left\lfloor \frac{l+h}{2} \right\rfloor = \left\lfloor \frac{1+2}{2} \right\rfloor = 1$

Binary Search Tree



P ← T

found ← False

while P ≠ nil and not found do

case

z < key (P) : P ← Left (P)

z > key (P) : P ← Right (P)

z = key (P) : found ← true

if found then FOUND

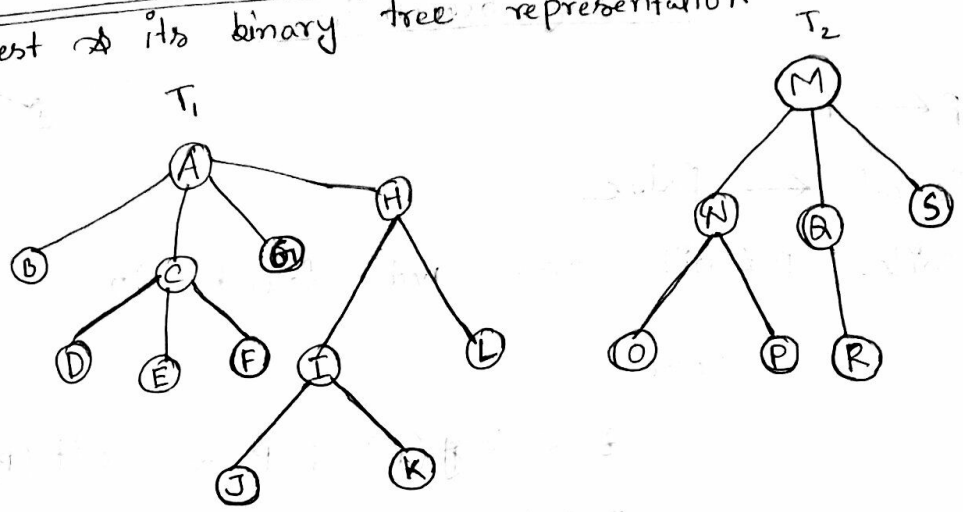
else NOT FOUND

* forest কে binary tree তে convert করার process টা
  খুব natural correspondence

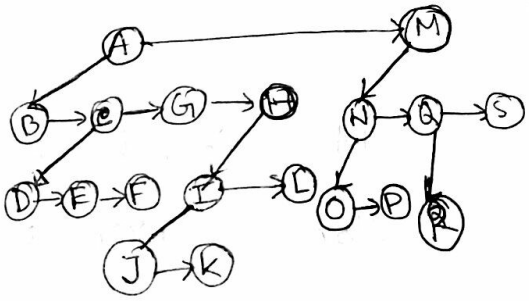
Quiz – 3
Next wednesday

13.07.16

* A forest ও its binary tree representation



Algorithm : Preorder tree traversal of a binary tree



Binary tree

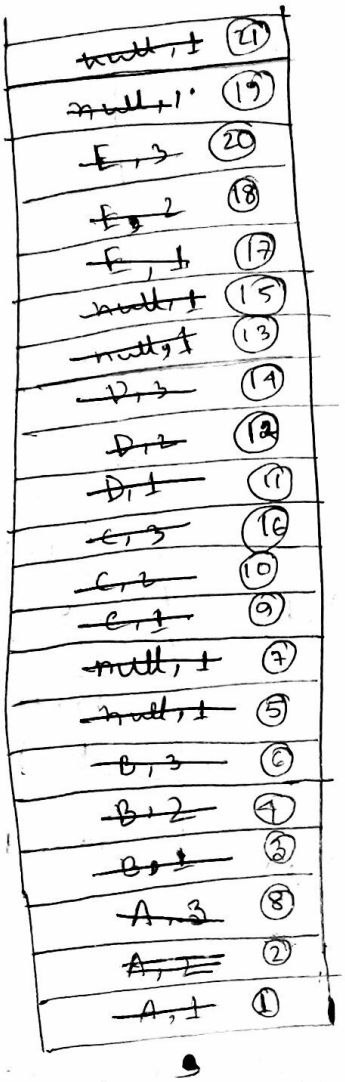
Preorder
1. root (1)
2. left node (2)
3. Right node (3)

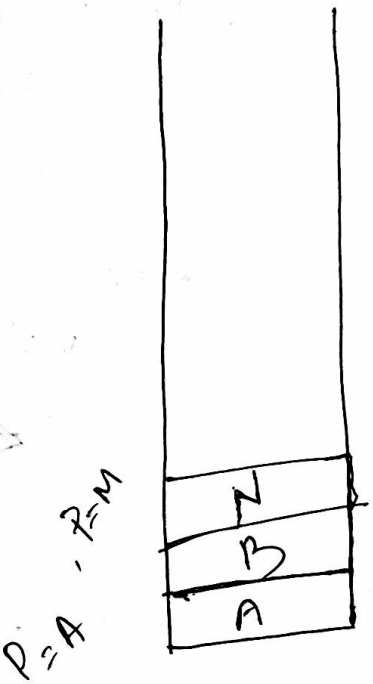
```
s ← empty stack
s ⇐ (root, 1)  → root এর corresponding num
while s ≠ empty do
    (p, i) ⇐ s
    if p ≠ nil then
        case
            i = 1 :
                s ⇐ (p, 2)
                visit node p → Print
            i = 2 :
                s ⇐ (p, 3)
                s ⇐ (left(p), 1)
            i = 3 :
                s ⇐ (Right(p), 1)
```

* inorder, postorder

Stack update:

A → B → C → D → E





5.9 * ~~A forest~~

level order traversal of a forest representing as a binary tree via natural correspondence

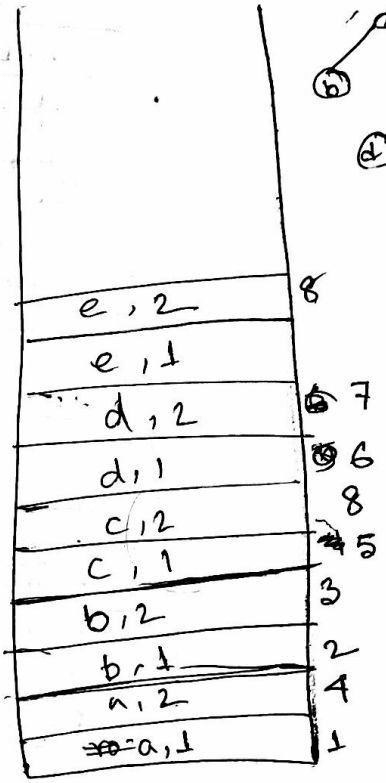
```
Q ← empty
Q ⇐ root
while Q ≠ empty do
    P ⇐ Q
    while P ≠ nil do
        visit node P
        if Left(P) ≠ nil then Q ⇐ Left(P)
        P ← Right(P)
```

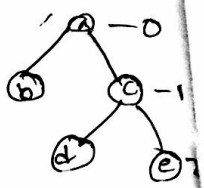
* H:W = Algorithm 5.7 → Postorder binary tree traversal.

17.07.16

Alogrithm: Inorder binary tree traversal b a d c e

```
S ← empty stack
S ⇐ (root, 1)
while S ≠ empty
    (P, i) ⇐ S
    if P ≠ nil then
        if i = 1 then
            S ⇐ (P, 2)
            S ⇐ (left(P), 1)
        else
            visit node P
            S ⇐ (Right(P), 1)
```

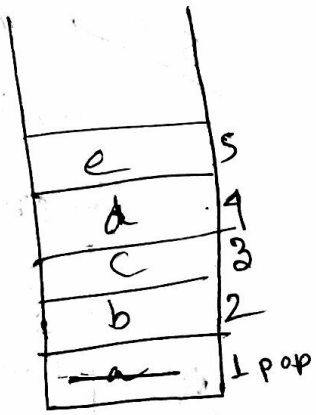




Algorithm: level order traversal of a binary tree.

a b c d e

```
Q ← empty Queue
Q ⇐ root
while Q ≠ empty do
    P ⇐ Q
    if P ≠ nil do
        visit node P
        Q ⇐ left(P)
        Q ⇐ Right(P)
```



19.07.16

Threaded Binary Tree

- A binary tree is threaded by making all right child pointers that would normally be ~~with~~ null point in the in-order successor of the node and all left child pointers that would normally be null point to the in-order predecessor of the node.

Advantages

- It is possible to discover the parent ~~of the parent~~ of the node from a threaded binary tree, without explicit use of parent pointers or stack.
- This can be useful where stack space is limited or where a stack of parent pointers in unavailable.

Types

1. Single Threaded: each node is threaded towards either the in-order predecessor or sucessor.
2. Double threaded: each node is threaded towards both the in-order ~~precessor~~ predecessor and successor.

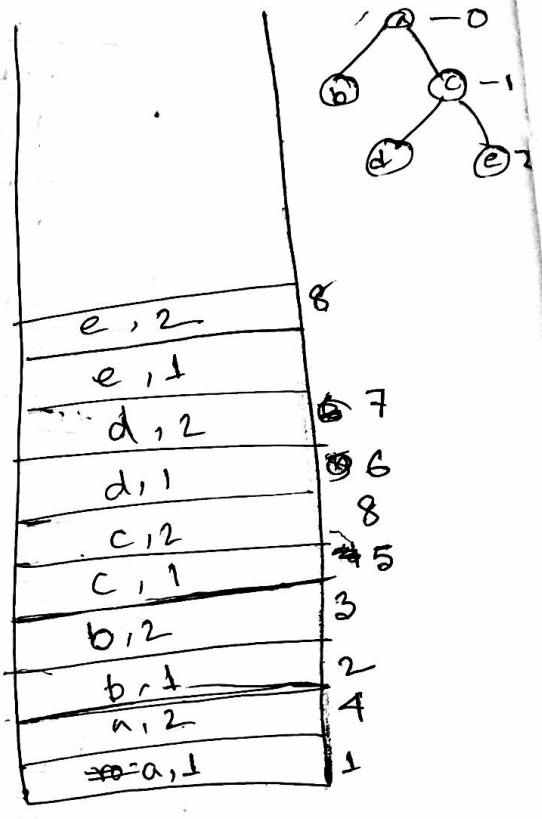
* H:W = Algorithm 5.7 → Postorder binary tree traversal.

17.07.16

Alogrithm: Inorder binary tree traversal

b a d c e

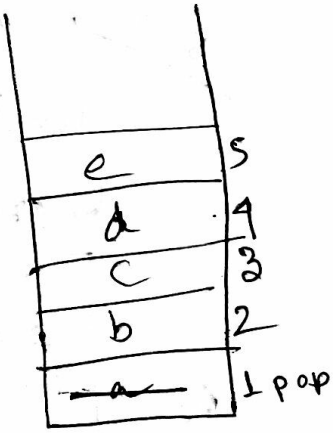
```
S ← empty stack
S ⇐ (root, 1)
while S ≠ empty
    (P, i) ⇐ S
    if P ≠ nil then
        if i = 1 then
            S ⇐ (P, 2)
            S ⇐ (left(P), 1)
        else
            visit node P
            S ⇐ (Right(P), 1)
```



Algorithm: level order traversal of a binary tree.

a b c de

```
Q ← empty Queue
Q ⇐ root
while Q ≠ empty do
    P ⇐ Q
    if P ≠ nil do
        visit node P
        Q ⇐ left(P)
        Q ⇐ Right(P)
```



19.07.16

Threaded Binary Tree

- A binary tree is threaded by making all right child pointers that would normally be ~~with~~ null point in the in-order successor of the node and all left child pointers that would normally be null point to the in-order predecessor of the node.

Advantages

- It is possible to discover the parent ~~of the parent~~ of the node from a threaded binary tree, without explicit use of parent pointers or stack.
- This can be useful where stack space is limited or where a stack of parent pointers in unavailable.

Types

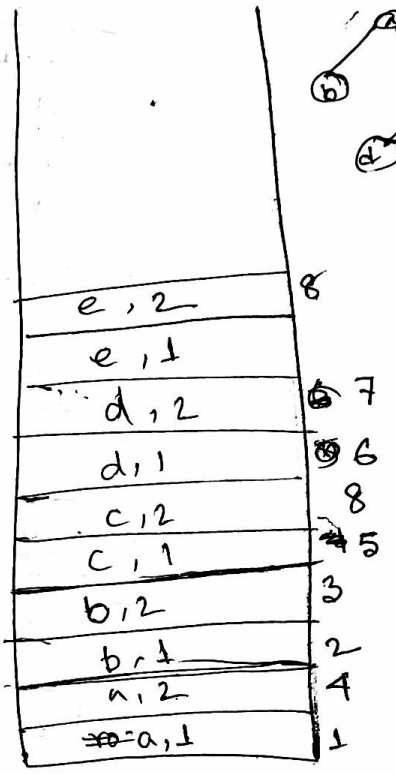
1. Single Threaded: each node is threaded towards either the in-order predecessor or sucessor.
2. Double threaded: each node is threaded towards both the in-order ~~precessor~~ predecessor and successor

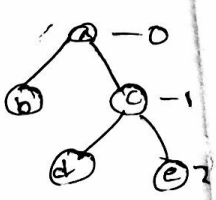
* H:W = Algorithm 5.7 → Postorder binary tree traversal.

17.07.16

Alogrithm: Inorder binary tree traversal b a d c e

```
S ← empty stack
S ⇐ (root, 1)
while S ≠ empty
    (P, i) ⇐ S
    if P ≠ nil then
        if i = 1 then
            S ⇐ (P, 2)
            S ⇐ (left(P), 1)
        else
            visit node P
            S ⇐ (Right(P), 1)
```

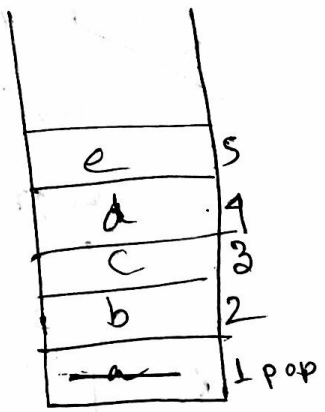




Algorithm: level order traversal of a binary tree.

a b c d e

```
Q ← empty Queue
Q ⇐ root
while Q ≠ empty do
    P ⇐ Q
    if P ≠ nil do
        visit node P
        Q ⇐ left(P)
        Q ⇐ Right(P)
```



19.07.16

Threaded Binary Tree

- A binary tree is threaded by making all right child pointers that would normally be ~~with~~ null point in the in-order successor of the node and all left child pointers that would normally be null point to the in-order predecessor of the node.

Advantages

- It is possible to discover the parent ~~of the parent~~ of the node from a threaded binary tree, without explicit use of parent pointers or stack.
- This can be useful where stack space is limited or where a stack of parent pointers in unavailable.

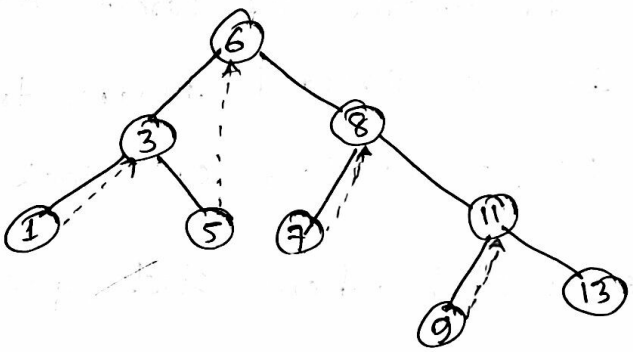
Types

1. Single Threaded: each node is threaded towards either the in-order predecessor or sucessor.
2. Double threaded: each node is threaded towards both the in-order ~~precessor~~ predecessor and sucessor.

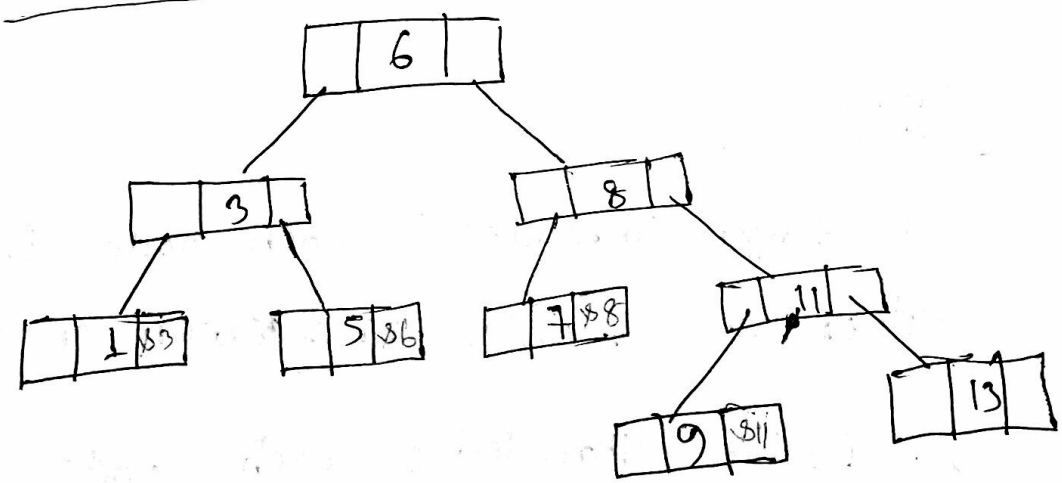
* tree convert threaded binary tree inorder

Threaded tree traversal

- We start at the leftmost node in the tree, print it and follow its right thread.
- If we follow a thread to the right, we output the node and continue to its right.
- If we follow a link to the right, we go to the leftmost node, print it and continue.



threaded binary tree



24.07.16

Chapter-7 (Hashing)

— suppose, we have an array of m table locations T[0], T[1], ---- T[m-1], say; and given an element z to be inserted we transform it to a location h(z);

$0 \leq h(z) < m$. h is called a hash function.

— the hash function takes an element to be stored in a table and transform it into a location in the table.

Principles of hash function

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

— A has function should break up naturally occuring clusters of elements.

0.74061

– A has function should be very quick and easy to compute.

XOR 7.2

word	Binary form	h(z) = Third bit, last two bits	$h_1(z)$	$h_2(z)$	$h_3(z)$
THE	10100 01000 00101 = 20741	$(101)_2 = 5$	25	2	13
OF	01111 00110	$(110)_2 = 6$	9	21	21
AND	00001 01110 00100	$(000)_2 = 0$	11	19	4
TO	10100 01111	7	27	4	7
A	00001	1	1	1	19
IN	01001 01110	2	7	23	30
THAT	10100 01000000 10100	4	9	18	17
IS	01001 10011	3	26	28	2

location variable

search

$h_1(THE)$ =
10100
01000
11100
00101
11001 = 25

$h_2(z) = z \bmod m$

$h_2(THE) = 20741 \% 31$

$h_3(z) = 1 + 30$ [fractional part of .612542337 1 * value of word]
$= 20741 * 0.612542337 1 = 12704.940$
$= 1 + 30 * 0.74061$

26.07.16

Algorithm: Search & insertion into a hash table with collision Resolved by coaleseed chaining;

```
loc ← h(z)                          ← Location variable
found ← false
if T[loc] is not empty then
        i ← loc
        repeat
            if T[i] = z then found ← ~~false~~ true
            else prev i ← i
                 i ← link(i)                ← Search
        Until found or i = -1
if not found then
        if T[loc] is empty then T[loc] ← z
        else
            repeat free ← free - 1 until T[free] is empty
            if free = -1 then overflow
            else
                link(previ) ← free
                T(free) ← z
                link(free) ← -1
```

Collision Resolution

* . Coalesced chaining

Fig: 7.25

27.07.16

~~Alog~~ Algorithm : Linear Probing

```
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 table overflow
        else
                n ← n+1
                T[i] ← z
```

2. Seperate chaining
3. Double hasing

Lab final
7A04
Time : 3:30-4:20

31.07.16

Chapter - 2(Elementray Data objects)

H.W Number Conversion

1. Binary — 0 1 = 2
2. integer / Decimal — ~~10~~ 0 - 9 = 10
3. logical / Hexadecimal = 0 — F = 16
4. Octal — 0 - 7 = 8

5. Character String Representation
6. Floating Point Representation

* Example - 3 → solution

$$\sum_{i=0}^{K-1} a_i b^i$$

$$(10011)_{10} = 1\times10^4 + 0\times10^3 + 0\times10^2 + 1\times10^1 + 1\times10^0$$

$$= 1\times(1010)^{100} + 0\times(1010)^{11} + 0\times(1010)^{10} + 1\times(1010)^1 + 1\times(1010)^0$$

$$= 100111000100000 + 1010 + 1$$

$$= (100111000110 11)_2$$

any base to decimal convert করতে।

$(10011)_2 = (((1)2+0)2+0)2+1)2+1$
$= (19)_{10}$

Horner's Method

$$\sum_{i=0}^{k-1} a_i b^0$$

$$= a_{k-1}b^{k-1} + a_{k-2}b^{k-2} + \cdots + a_3 b^3 + a_2 b^2 + a_1 b^1 + a_0 b^0$$

$$= a_{k-1}b^{k-1} + a_{k-2}b^{k-2} + \cdots + a_3 b^3 + a_2 b^2 + a_1 b + a_0$$

$$= (a_{k-1}b^{k-2} + a_{k-2}b^{k-3} + \cdots + a_3 b^2 + a_2 b + a_1) b + a_0$$

$$= (a_{k-1}b^{k-3} + a_{k-2}b^{k-4} + \cdots + a_3 b + a_2) b + a_1) b + a_0$$

$$= (a_{k-1}b^{k-4} + a_{k-2}b^{k-5} + \cdots + a_3) b + a_2) b + a_1) b + a_0$$

H.W

Signed Integers

1. 1's complement
2. 2's complement ⟶ if n is positive integer, $-n = 2^{(k)} - n$, (k) → no of bits
3. Signed Magnitude
4. offset Representation

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

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

if n is negative,

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

$-5 = 2^3 + (-5) = 3$

$= (0011)$

02.08.16

Floating Point Representation

16 bit

sign	Exponent	Manlissa
← 1bit →	← 5bit →	← 10 bit →

2.5 → 16 bit standard

2^{10} → exponent, binary 01010

sign bit — 0 → positive value
— 1 → negative value

Exponent $\frac{2.5}{2^v} \times 2^v = 0.625 \times 2^v$ — Ex

$\frac{2.5}{2} = 1.25$

$\frac{1.25}{2} = 0.625$

Exponent

-347.625

$= -0.6789550\,78 \times 2^9$ → Exponent

$= -0.679 \times 2^9$

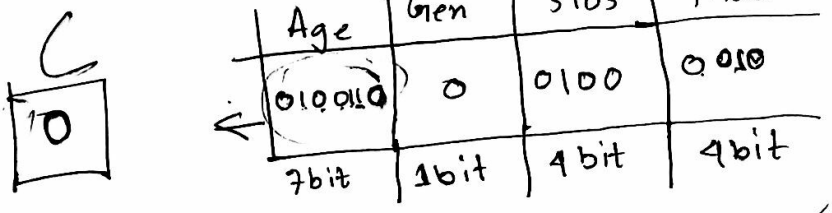
1	01001	1010110111

.625 × 2 = 1.358 — 1
.358 × 2 = 0.716 — 0
.716 × 2 = 1.432 — 1
.432 × 2 = 0.864 — 0
.864 × 2 = 1.728 — 1
.728 × 2 = 1.456 — 1
.456 × 2 = 0.912 — 0
.912 × 2 = 1.824 — 1
.824 × 2 = 1.648 — 1
.648 × 2 = 1.296 — 1

Manlissa

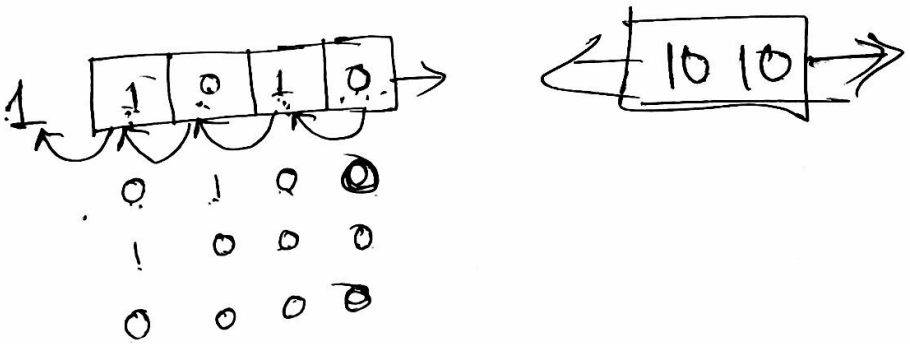
1byte = 8bit
1word = 2byte

Paked words



· Insert → Shift_Left (Person_data and 16 ~~bits~~ times)

Left shift :




```
0100110   0 0100 0010
1111111   0 0000 0000
-------------------- (AND)
0100110   0 0000 0000
```

for ↖ Shift Right ((2C42 AND FE00), 9) → Ans (Hexa)

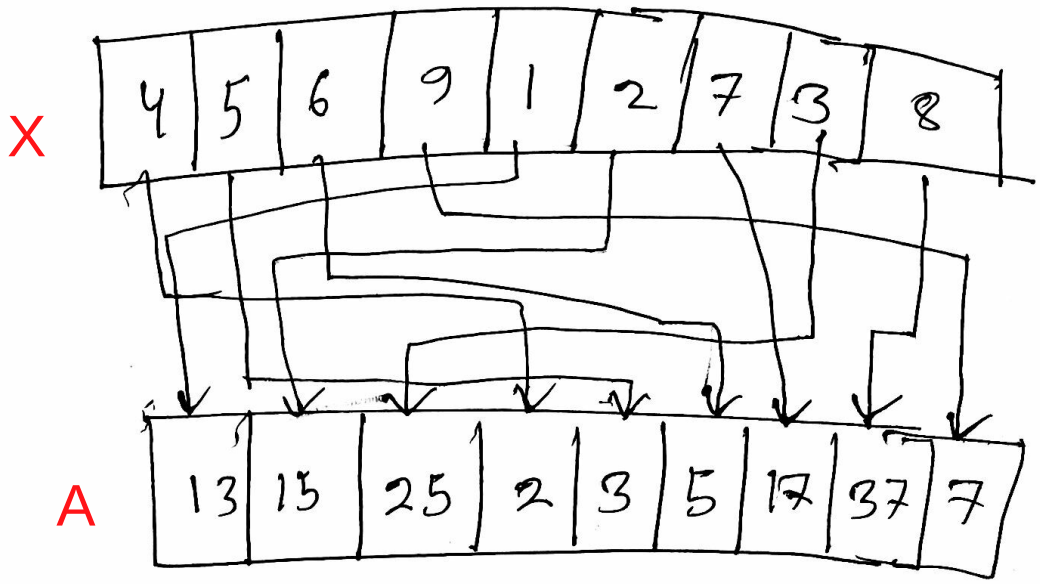
Shift Right ((2C42 AND 00F0), 4) → sibs

update → (2C42 OR 0003)

Alternate → (2C42 XOR 0420)

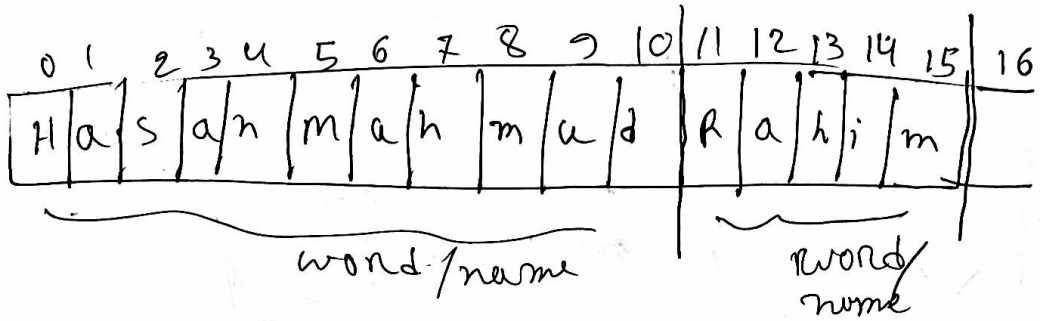
Chapter - 4

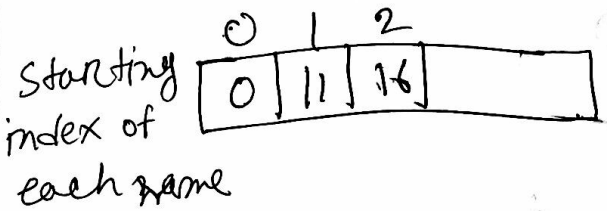
07.09.16



Algorithm: simple indirect selection sort

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





this is to indicate the ~~first~~ starting of a word or name in a character array.

$i=1$
$k=2$
$A[4] < A[5]$
$i=2$

$i=2$
$k=3$
$A[5] < A[6]$
$i=3$

$i=3$
$k=4$
$A[6] < A[7]$
$i=4$

$i=4$
$k=5$
$A[9] < A[13]$
$i=5$

$i=5$
$k=6$
$A[13] < A[15]$
$i=6$

$i=6$
$k=7$
$A[15] < A[17]$
$i=7$

i = 7

k = 8

17 25

A[7] < A[3]

i = 8

i = 8

k = 9

25 37

A[8] < A[8]

i = 9