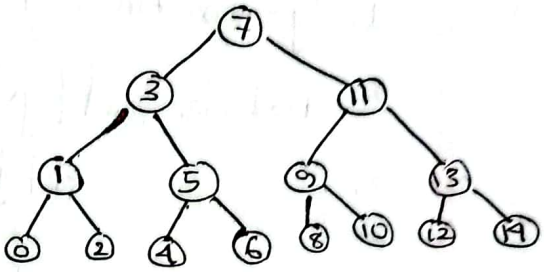
Sub: Binary SEARCH. (sorted array) ①

Day
Time : Date : / /

```
(1)  BinSearch (A,K)
(2)  শুরু = 0
(3)  শেষ = size(A) - 1
(4)  while শুরু ≤ শেষ
(5)      মাঝে = (শুরু + শেষ) / 2
(6)    যদি A[মাঝে] == K.
(7)      return 'Found'
(8)    অথবা যদি A[মাঝে] < K.
(9)      শুরু = মাঝে + 1
(10)   অথবা
(11)     শেষ = মাঝে - 1
(12) return 'not found'
```

0 1 2 3 4 5 6 7 8 9 10
11 12 13 14.



height of tree → $\log_2(n+1)$.

Selection Sort

```
1. Find_Min(A):
2.   n = size(A).
3.   min A[0]
4.   for j=1 to n
5.     if A[j] < min
6.       min = A[j]
7.     end if
8.   end for
9.   print min.
```

```
1.  Section_sort(A):
2.    n = size(A).
3.    for i=0 to n-1
        if min = i   //max = i
        for j=i+1 to n
        if A[j] < A[min]   //>
          min = j   //max = j
        end if
      end for
      swap A[j], A[min]
      end for
```

Sub: Data Structure (Definition)

Day

Time: Date: / /

Data Structure: There are 2 words Data and Structure. Data means raw facts or information that can be processed to get results. Elementary items constitute a unit and that unit may be considered as a structure. DS is a structure or unit where we organize elementary data items in different ways and there exists structured relationship among the items.

Example: Array, linked list, Structure, tree, graph, stack queue, hash, table etc.

Elementary data items: int, char, node, edge, weight value etc.

Selection Sort Pass in Ascending:

	29	72	98	13	87	66	52	51	36
1.	13	72	98	29	87	66	52	51	36
2.	13	29	98	72	87	66	52	51	36
3.	13	29	36	72	87	66	52	51	98
4.	13	29	36	51	87	66	52	72	98
5.	13	29	36	51	52	66	87	72	98
6.	13	29	36	51	52	66	87	72	98
7.	13	29	36	51	52	66	72	87	98
8.	13	29	36	51	52	66	72	87	98

Sub :

Day

Time : Date : / /

Descending:

29	72	98	13	87	66	52	51	36
1. [98]	72	(29)	13	87	66	52	51	36
2. [98]	[87]	(29)	13	(72)	66	52	51	36
3. [98]	[87]	[72]	13	(29)	66	52	51	36
4. [98]	[87]	[72]	[66]	(29)	(13)	52	51	36
5. [98]	[87]	[72]	[66]	[52]	(13)	(29)	51	36
6. [98]	[87]	[72]	[66]	[52]	[51]	(29)	(13)	36.
7. [98]	[87]	[72]	[66]	[52]	[51]	[36]	(13)	(29)
8. [98]	[87]	[72]	[66]	[52]	[51]	[36]	[29]	[13]

Sub: Insertion (Sorted array) (ascending order)

Day | Time: | Date: / /

A:

10	15	20	25	30
0	1	2	3	4

1. j = size(Arr) − 1
2. while ~~A[j] > k~~ j ≥ 0 and A[j] > k
3. A[j+1] = A[j]
4. ~~end while~~ j = j − 1
5. A[j] end while
6. A[j+1] = k

Sorting (Insertion Sorting)

0	1	2	3	4	5
5	12	4	−6	1	3

→ (unsorted)

0 moves: (5) (12) | 4 − 6 1 3 (j = i−1; key = i)

3 moves: (4) (5) (12) | −6 1 3

3 moves: (6) (4) (5) (12) | 1 3

3 moves: −6 (1) (4) (5) (12) | 3

3 moves: −6 1 (3) (4) (5) (12)

Total = 11 moves.

→ Algo (PTO)

Sub :

Day

Time : Date : / /

```
0. Insertion_sort(A):
1. n = size(A).
2. for i=1 to n.
3.     K = A[i]
4.     j = i-1
5.     while j≥0 and A[j]>K.
6.         A[j+1] = A[j]
7.         j = j-1
8.     end while.
9.     end for A[j+1] = K.
10. end for.
```

① // A[j] < K.

Ascending

K | 5 | 12 | 4 | -6 | 1 | 3

j = i-1

1. 5 | 12 | 4 | -6 | 1 | 3
2. 4 | 5 | 12 | -6 | 1 | 3
3. -6 | 4 | 5 | 12 | 1 | 3
4. -6 | 1 | 4 | 5 | 12 | 3
5. -6 | 1 | 3 | 4 | 5 | 12

11 moves – underline

key [j]-এর থেকে বড় হইলে পিছে সরাব [j]-কে [j+1]-এ রাখব

Descending

5 | 12 | 4 | -6 | 1 | 3

1. 12 | 5 | 4 | -6 | 1 | 3
2. 12 | 5 | 4 | -6 | 1 | 3
3. 12 | 5 | 4 | -6 | 1 | 3
4. 12 | 5 | 4 | 1 | -6 | 3
5. 12 | 5 | 4 | 3 | 1 | -6

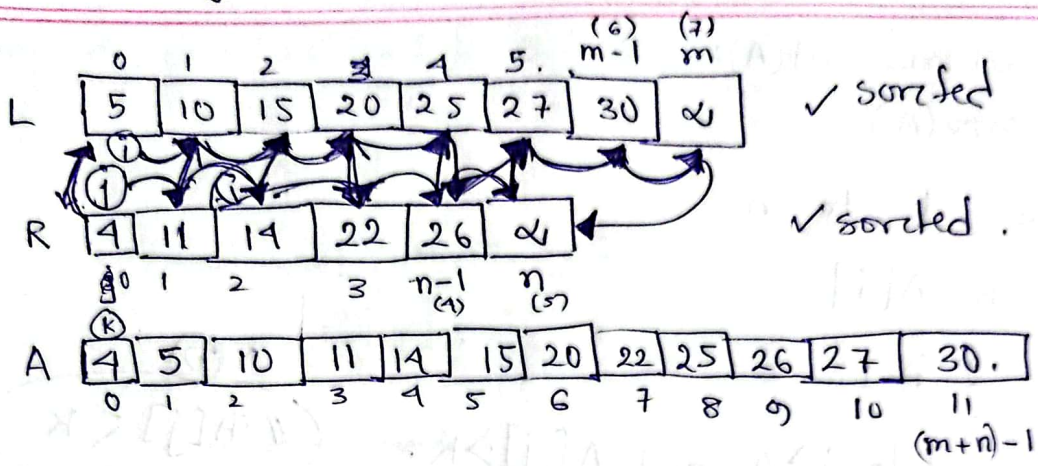
4 moves.

Sub: Merge Sort

Day

Time: Date: / /

Merging



Algo:

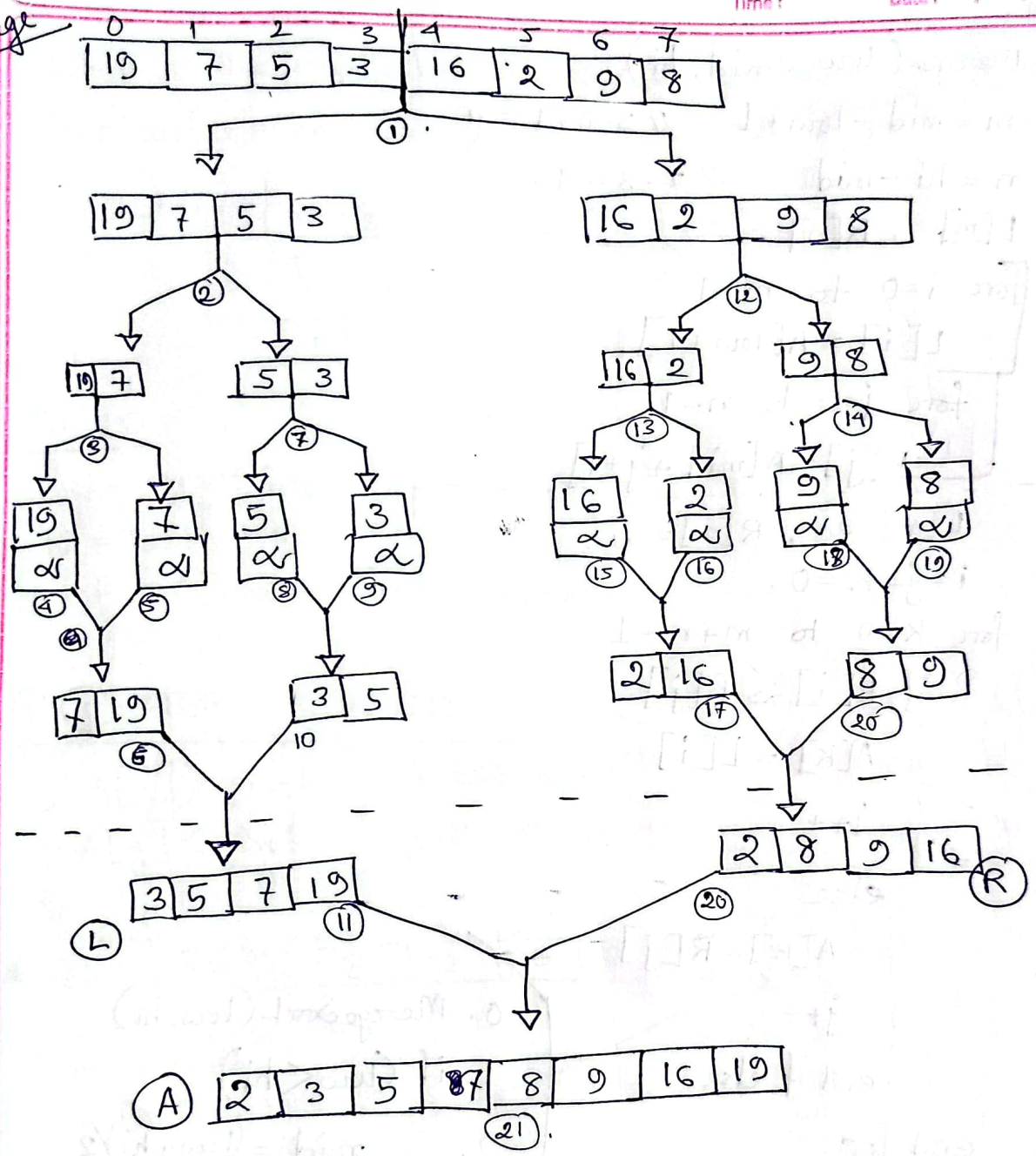
```
0. Merging (L,R)
1. m = size(L)
2. n = size(R)
3. L(m) = ∞
4. R(n) = ∞
5. i = j = k = 0
6. for k=0 to m+n-1   (or up to m+n)
7.     if L[i] ≤ R[j]
8.         A[k] = L[i]
9.         i++
10.    else
11.        A[k] = R[j]
12.        j++
13.    end if
14. end for.
```

Sub :

Day

Time : Date : / /

merge



Time Complexity = O(nlogn)

Sub :

Day

Time : Date : / /

Algo

```
0. Merge (low , mid, hi)
1. m = mid - low + 1      // 3-0+1 = 4
2. n = hi - mid           // 7-3 = 4.
3. L[m] , R[n]
4. for i=0 to m-1
5.     L[i] = A[low+i]
6.  for j=0 to n-1
7.     R[j] = A[mid+j+1]
8.   L[m]=∞ , R[n]= ∞
9.   i=j=k=0.
10. for k=0 to m+n-1
11.     if L[i] ≤ R[j]
12.         A[k] = L[i]
13.         i++
14.     else
15.         A[k]= R[j]
16.         j++
17.     end if else
18. end for
```

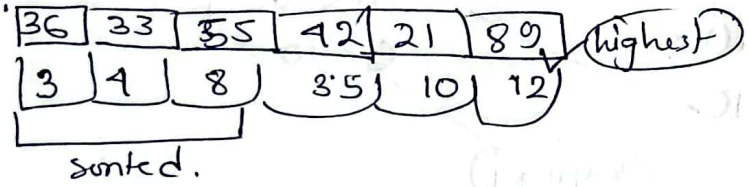
```
0. MergeSort (low, hi)
1. if (low < hi.)
2.     mid = (low+hi)/2
3.     MergeSort(low, mid) (L)
4.     MergeSort(mid+1, hi) (R)
5.     Merge(low, mid, hi)
6. end if .
```

Sub: Quick Sort . (Fast not fastest).

Day | Time : | Date : / /

Whether the smallest or greatest number will be at a corner (left or right) then it is called already sorted position.

exple:



TO DO:

Low = আমরা index.

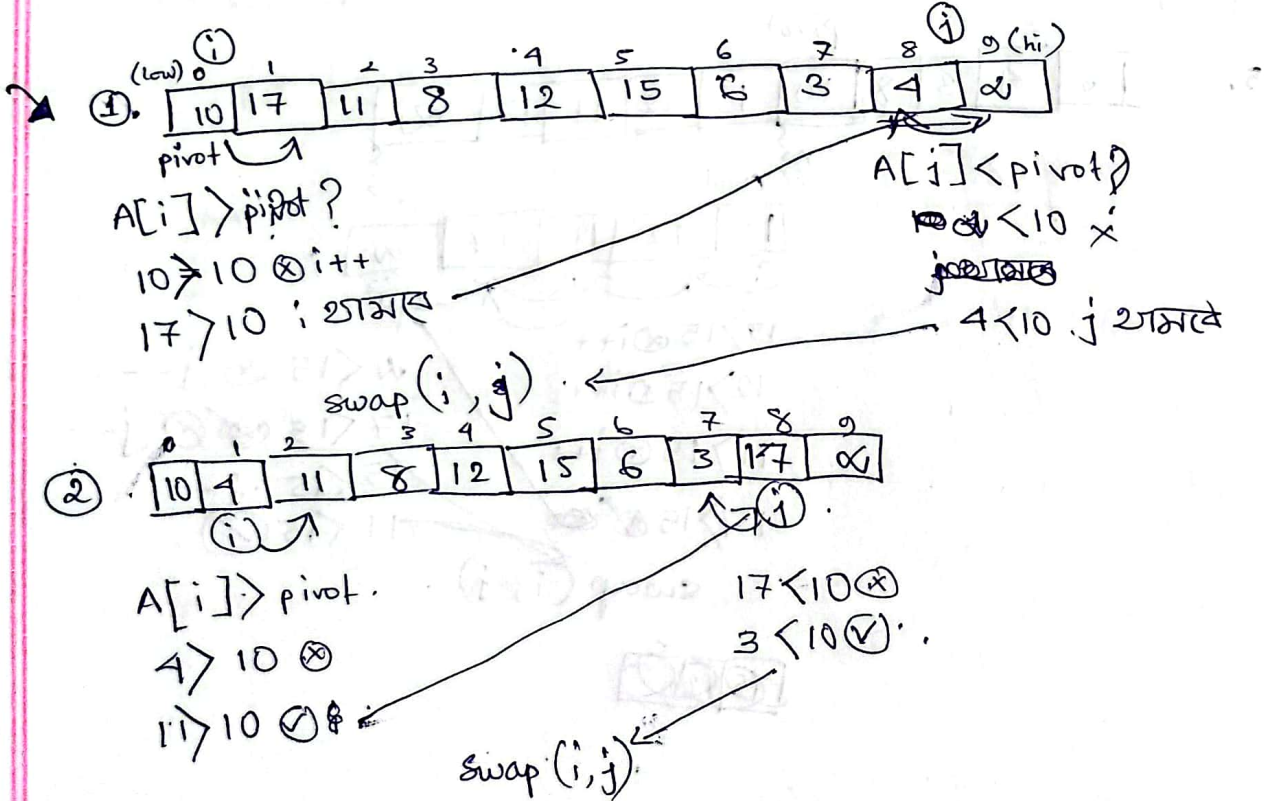
hi = last " .

pivot = A[low] = First element.

① (i) (j) কাকে কখন থামাব: (i) ->..<..(j)

A[i] > pivot. ; থামবে or i++

A[j] < pivot j থামবে or j--

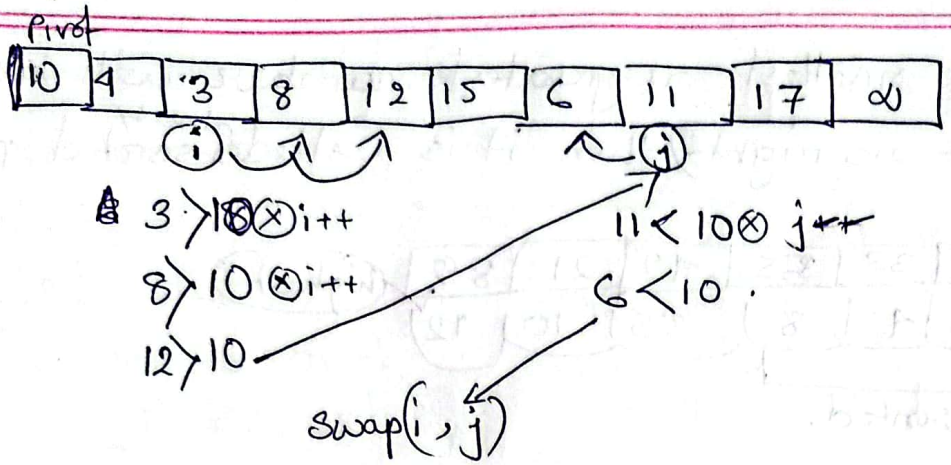


Sub :

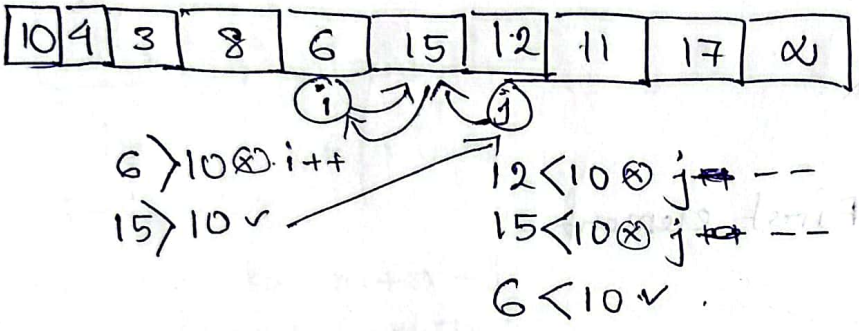
Day

Time : Date : / /

3.

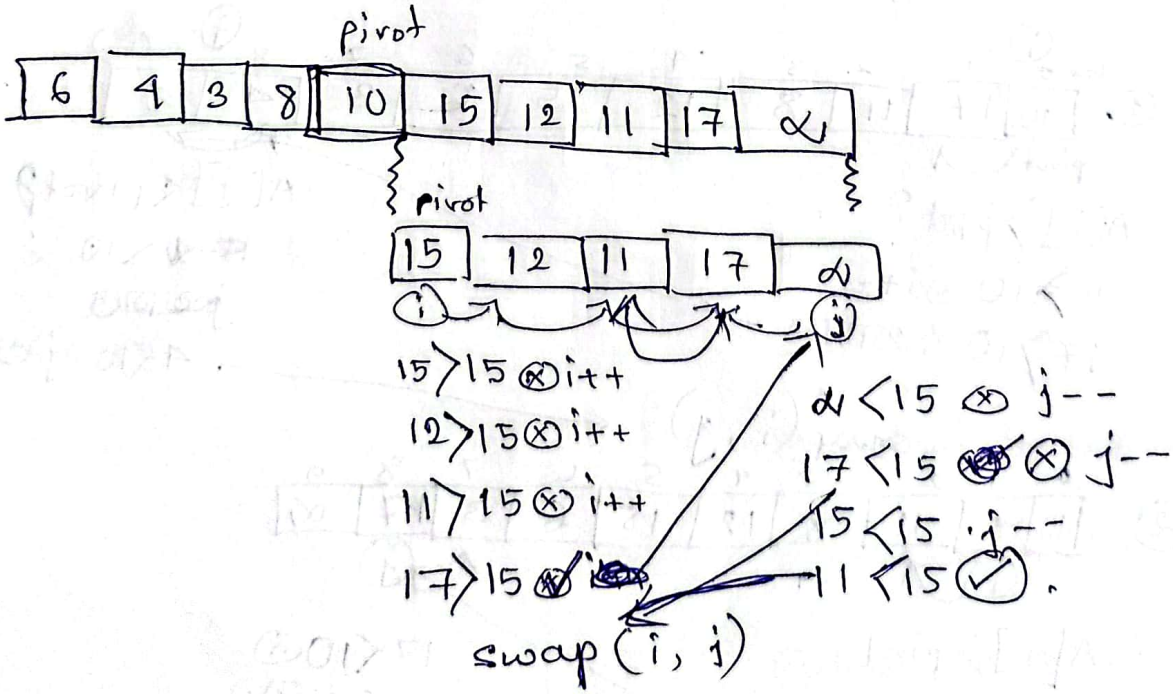


4.



এখানে j i-কে cross করেছে. তাই i < j হইছে না হয়ে j < i হইছে now swap (j, pivot)

5.



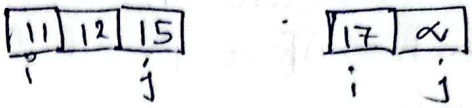
Sub :

Day

Time : Date : / /

6.

swap(j, pivot)

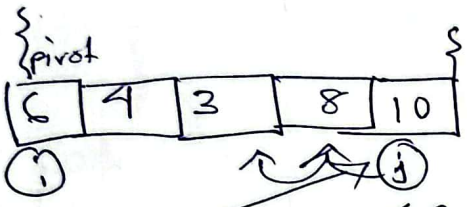


11 > 12

12 > 12

15 > 12 ✓

7



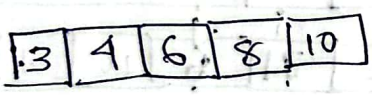
6 > 4 ✓

10 < 6 j--

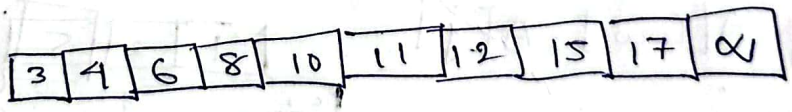
8 < 6 j--

3 < 6 ✓

swap(i, j)



***



Day

Sub :

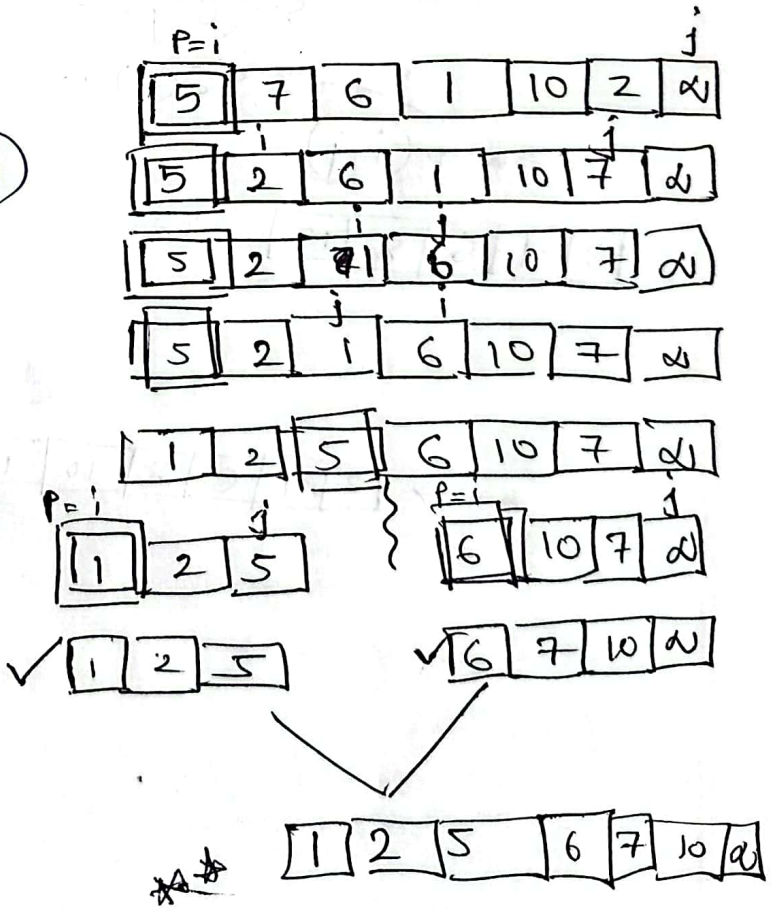
Time : Date : / /

Algo:

```
0. Partition (low, hi)
1. pivot = A[low]
2. i = low
3. j = hi
4. while i<j
5.     while A[i] <= pivot
6.         i++
7.     end while
8.     while A[j] > pivot
9.         j--
10.    end while
11.    if i<j
12.        swap(A[i], A[j])
13.    end if
14. end while
15. swap A[low], A[j]
16. return j
```

```
0. QuickSort (low, hi):
1. if low < hi
2.     j = Partition (low, hi)
3.     QuickSort (low, j)
4.     QuickSort (j+1, hi)
5. end if.
```

Another exple:



Comparison:

Name	TC (B)	TC (A)	TC (W)	SC (W)	Stability
Linear S.	O(1)	O(n)	O(n)	O(1)	
Binary S.	O(1)	O(logn)	O(logn)	O(1)	
Selection	O(n²)	O(n²)	O(n²)	O(1)	No
Quick	O(nlogn)	O(nlogn)	O(n²)	O(logn)	No
Merge	O(nlogn)	O(nlogn)	O(nlogn)	O(n)	Yes
Insertion	O(n)	O(n²)	O(n²)	O(1)	Yes

Adaptive
Insertion
Quick

non-adaptive
merge
selection

Internal sorting
Insertion
Quick

External
Merge

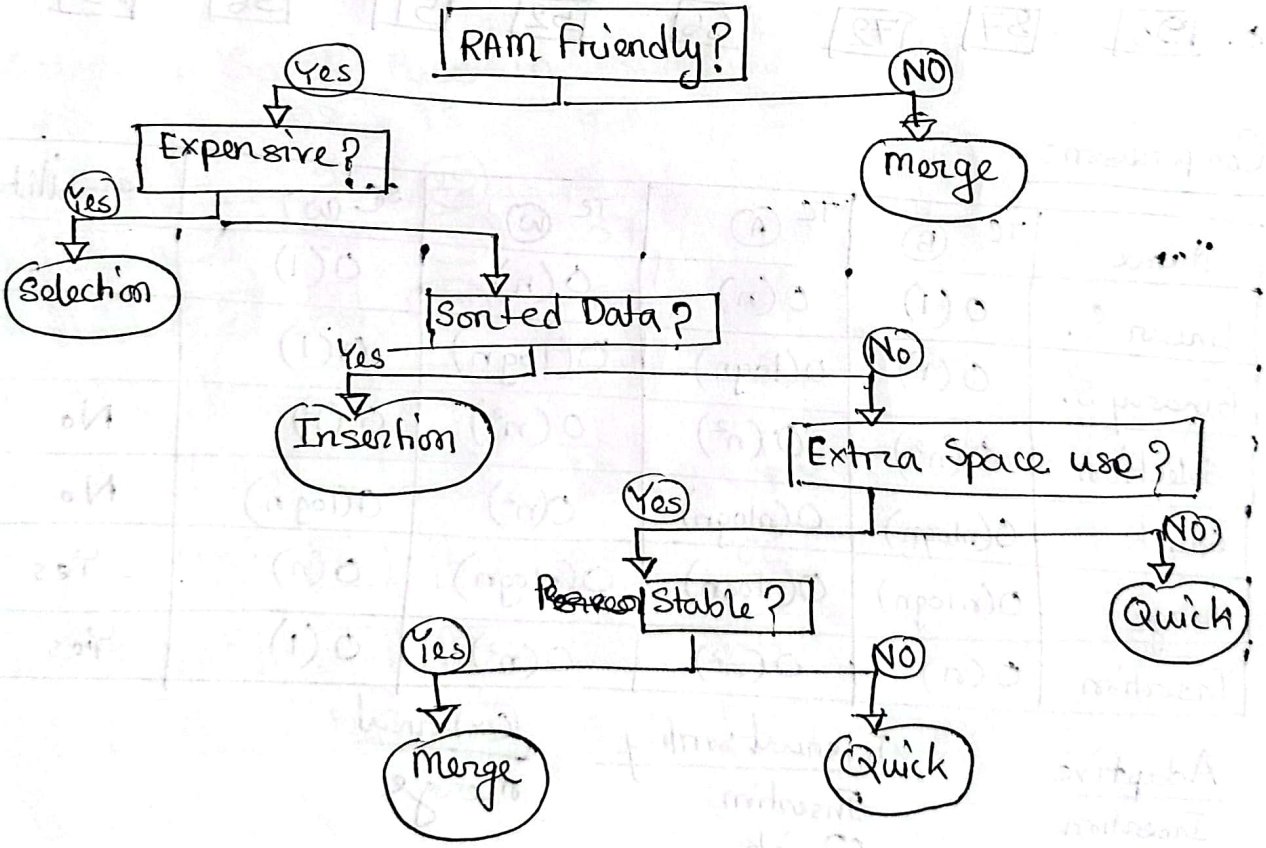
Sub :

Day

Time : Date : / /

Linear	Binary
1. Sorted / Unsorted	1. Sorted.
2. Sequential Search	2. Half interval search
3. TC (O(n))	3. TC O(logn)
4. Multidimensional array allowed	4. Only single dimensional
5. Equality comparisons	5. Ordering Comparisons
6. Less Complex	6. More Complex
7. Slow	7. Fast

Sorting Algos :



Sub: Algorithm.

Day

Time: Date: / /

USES:

- Selection: List is small, memory limited.
- Insertion: Small nearly sorted list.
- Quick: Fit in memory datasets (not large), in place sort (no extra memory used)
- Merge: Fast for linked list (stable, RAM access low)
  equal datas can be sorted.

More of applications:

- Insertion: - small list, ~~sorted~~ sort smaller sub prob. in Quick sort.

Sub: Recursion

Day
Time: Date: / /

Iterative: for (int i=n; i>=1; i--).
{ cout << i; }.

recursion: void FOR(i,n)
{ if (i<=n)
{ FOR(i+1,n)
cout << i. }
}.

① call FOR$_1$ with i=1.
② call FOR$_2$ with i=2
③ call FOR$_3$ " i=3
④ call FOR$_4$ i=4
⑤ call FOR$_5$ i=5.
i break no more call.
⑥ return to FOR$_4$
⑦ print 4.
⑧
⑨ return to FOR$_3$
print 3.
⑩ return to FOR$_2$
⑪
⑫ print 2
⑬ return to FOR$_1$
⑭ print 1
⑮ return to main.
⑯ end of recursion

Sub : Singly

Day

Time : Date : / /

```
#include <bits/stdc++.h>
using namespace std;

struct node
{
    int data;
    node *next;
};
node * root = NULL;

void printing()
{
    node *curr_node = root;
    while (curr_node != NULL)
    {
        cout << curr_node->data << " ";
        curr_node = curr_node->next;
    }
}

int search(int val)
{
    node *curr_node = root;
    while (curr_node != NULL)
    {
        if(curr_node->data == val)
        {
            return 1;
            break;
        }
        curr_node = curr_node->next
    }
    return -1;
}
```

Sub :

Day

Time : Date : / /

```
int last_node()
{
    node *curr_node = root;
    while(true)
    {
        if(curr_node->next == NULL)
        {
            break;
        }
        curr_node = curr_node->next;
    }
    return curr_node->data;
}
```

```
void prev_node(int val)
{
    node *prev_node = NULL;
    node *curr_node = root;
    while(curr_node != NULL)
    {
        if(curr_node->data == val)
            break;
        prev_node = curr_node;
        curr_node = curr_node->next;
    }
    cout << "\n Previous node Found " << " ";
}
```

Sub :

Day

Time : Date : / /

```
void insert_first(int val)
{
    node *temp = new node();
    temp->data = val;
    temp->next = NULL;

    if (root == NULL)
    {
        root = temp;
    }
    else
    {
        temp->next = root;
        root = temp;
    }
}
```

```
void insert_last(int val)
{
    node * temp = new node();
    temp->data = val;
    temp->next = NULL;
    if (root == NULL)
    {
        root = temp;
    }else
    {
        node * prev_node = NULL;
        node * curr_node = root;
        while(true)
        {
            if (curr_node->next == NULL)
            {
                curr_node->next = temp;
                break;
            }
            curr_node = curr_node->next;
        }
    }
}
```

VDS

Sub:

Day

Time: Date: / /

```
void insert_anywhere(int pos, int val)
{
    node *temp = new node();
    temp->data = val;
    temp->next = NULL;

    node *currnode = root;
    int cnt=0;
    while(curr_node != NULL)
    {
        curr_node = curr_node->next;
        cnt++;
    }

    curr_node = root;
    if (pos == 1)
    {
        insert_first(val);
        return;
    }

    if (cnt == pos)
    {
        insert_last(val);
        return;
    }
    for(int i=1 ; i<pos-1 ; i++)
    {
        curr_node = curr_node->next;
    }
    temp->next = curr_node->next;
    curr_node->next = temp;
}
```

Sub :

Day

Time : Date : / /

```
void delete_first()
{
    root = root->next;
}

void delete_last()
{
    node *prev_node = NULL;
    node *curr_node = root;

    while(curr_node != NULL)
    {
        if(curr_node->next == NULL){
            prev_node->next = NULL;
        }
        prev_node = curr_node;
        curr_node = curr_node->next;
    }
}

void delete_anywhere(int val)
{
    if(root->data == val)
    {
        delete_first();
        return;
    }

    if(last_node() == val)
    {
        delete_last();
        return;
    }
```

Sub :

Day

Time : Date : / /

```
node *prev_node = NULL;
node *curr_node = root;
while (curr_node != NULL)
{ if (curr_node -> data == val)
    break;
  prev_node = curr_node;
  curr_node = curr_node -> next;
}
prev_node -> next = curr_node -> next;
}

int main()
{
  ..... // call;
}
```

```
void removeduplicate()
{ node *curr_node = root;
  while (curr_node != NULL)
  { node* prev_node = NULL;
    if (curr_node -> data == curr->next
                          -> next -> data)
    { prev_node = curr_node -> n -> n;
      delete_anynode (curr -> next)
      curr_node -> next = prev_node;
    } else
    { curr = curr -> next;
    }
  }
}
```

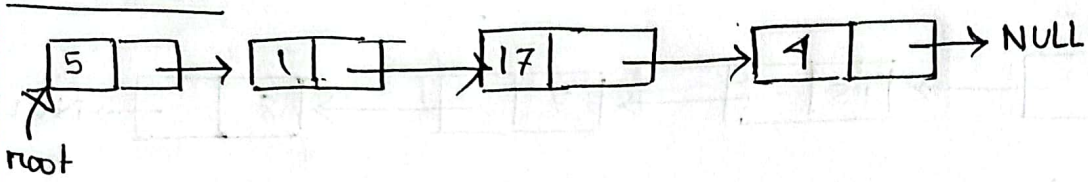
```
void printReverse ()
{ if (root == NULL)
    return;
  printReverse (root -> next);
  cout << root -> data << " ";
}
```

```
void duplicate ()
{ node *curr_node = root;
  while (true)
  { node *finder = curr_node -> next;
    while (finder != NULL)
    { if (finder -> data == curr_node -> data)
      { cout << finder -> data << " ";
        finder = finder -> next;
      }
    }
  }
}
```

Sub: SINGLY

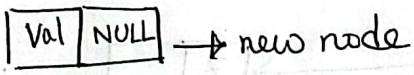
Day | Time : | Date : / /

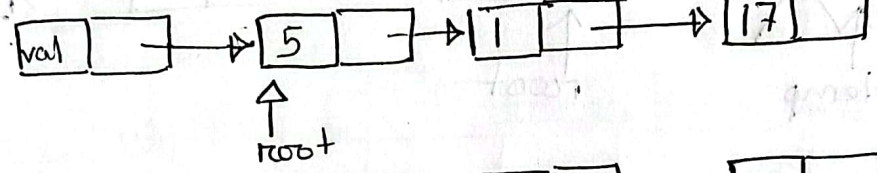
✱ Traverse :

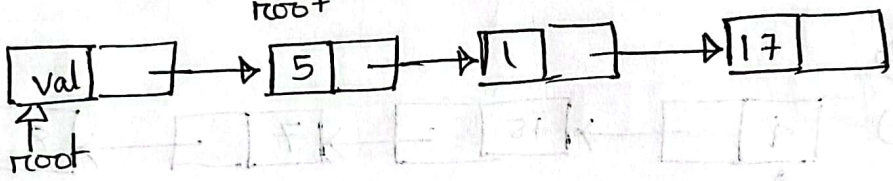


✱ Insert :

Front :

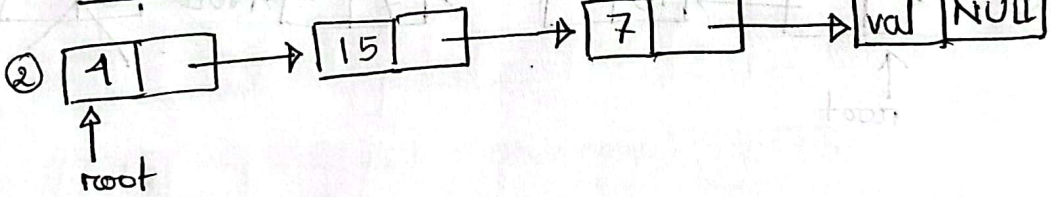
① 

② 

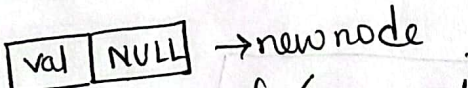
③ 

End :

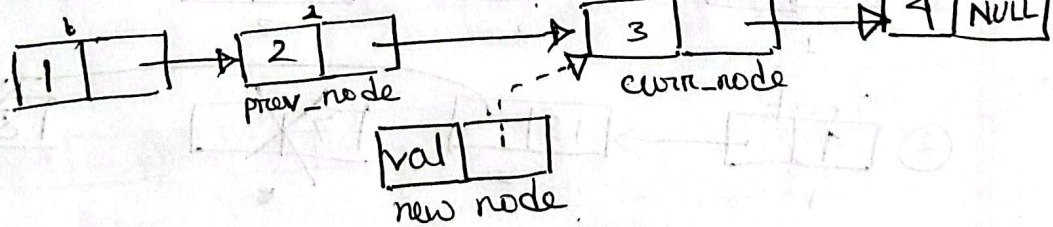
① 

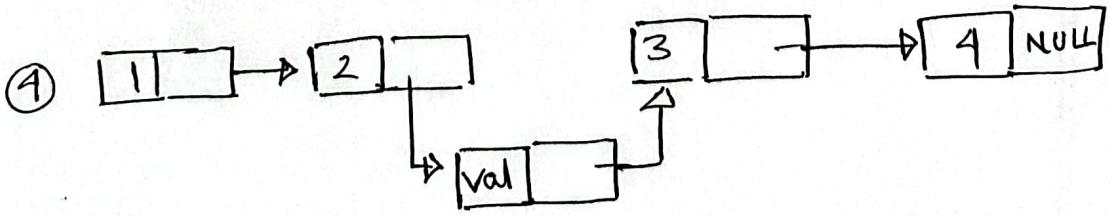
② 

RANDOM :

① 

② Call Front : if (pos == 1)

[if pos==3] ③ 

④ 

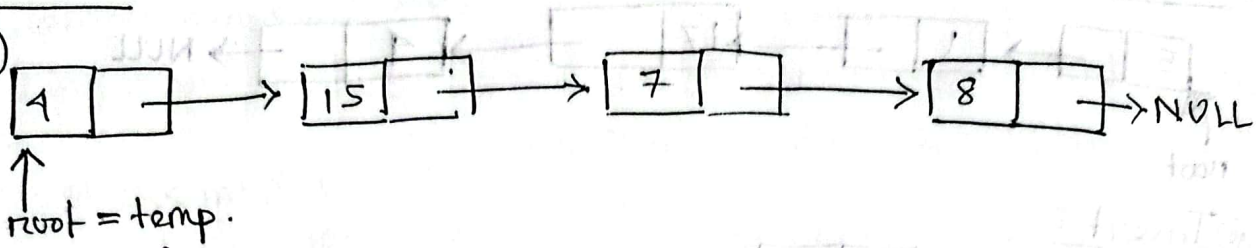
Sub :

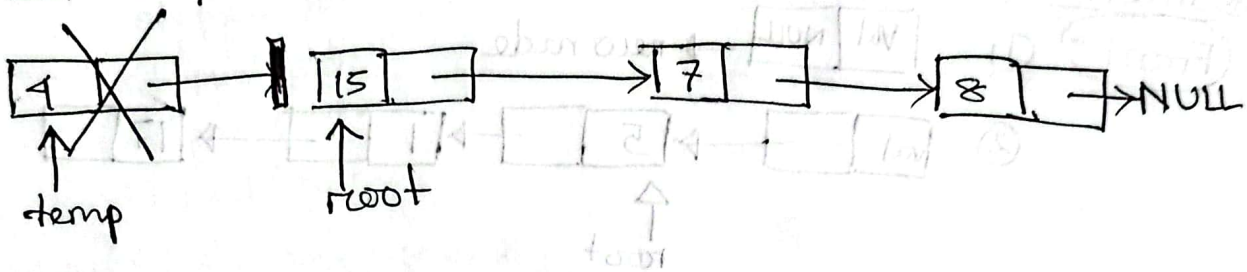
Day

Time : Date : / /

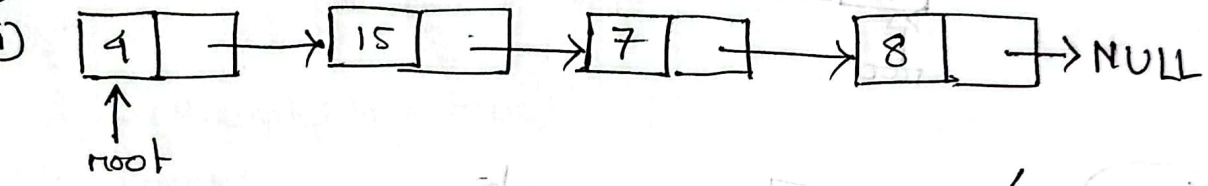
Delete :

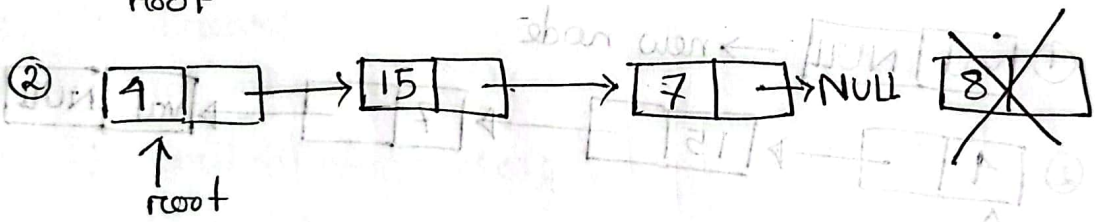
FRONT



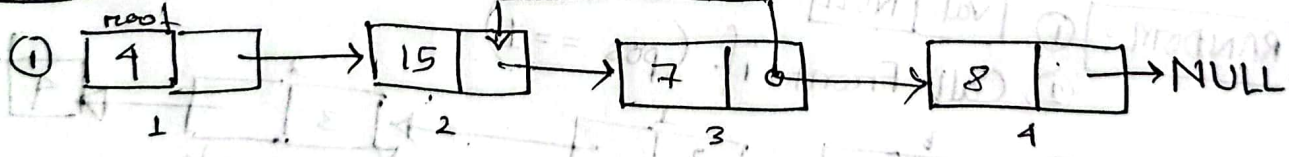


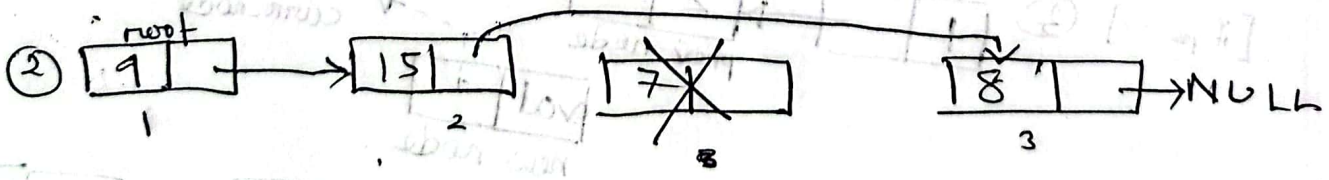
END





MIDDLE pos = 3



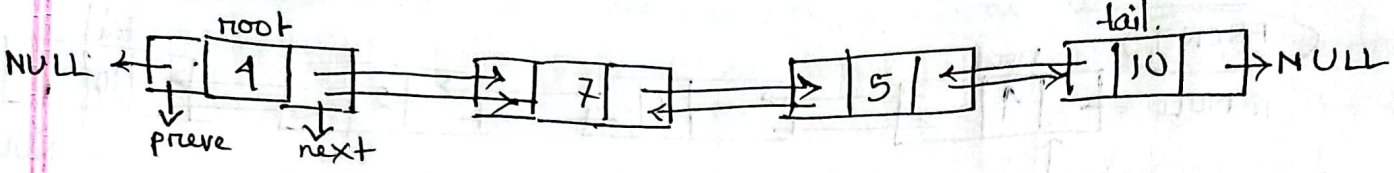


Sub: Doubly

Day

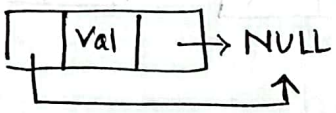
Time : Date : / /

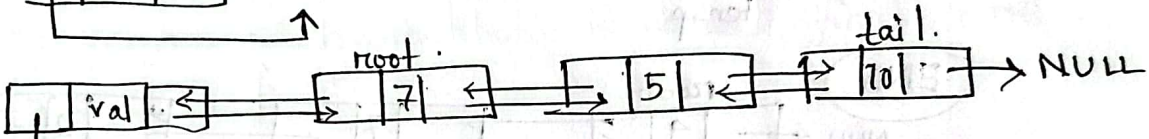
✱ Traverse :

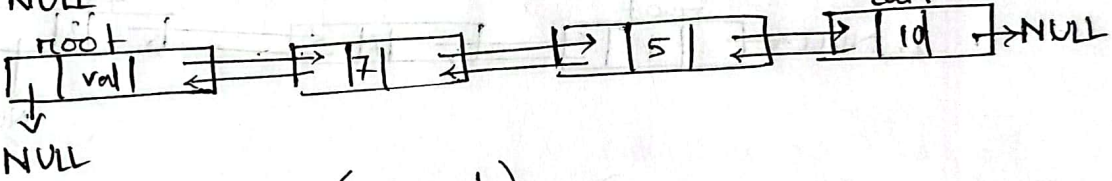


✱ INSERT :

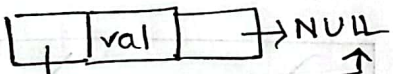
FRONT

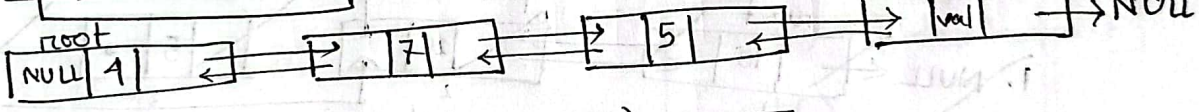
1. 

2. 

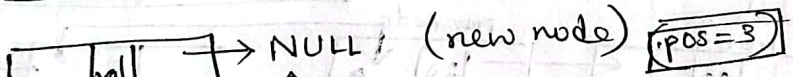
3. 

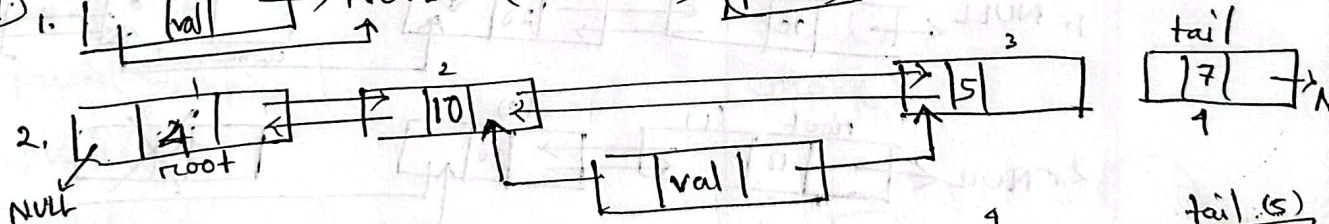
END

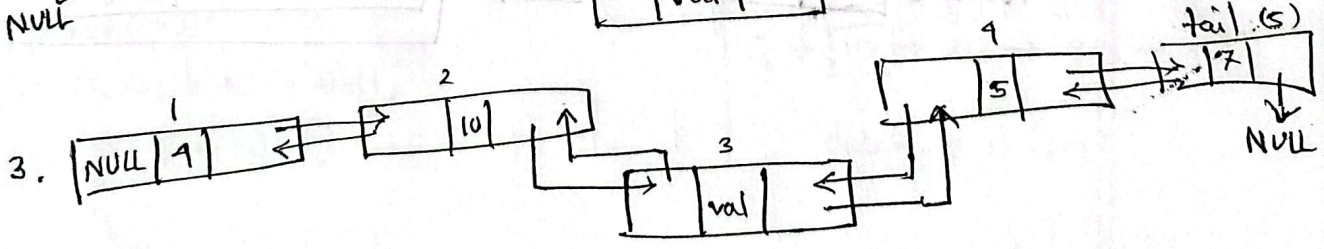
1. 

2. 

RANDOM

1. 

2. 

3. 

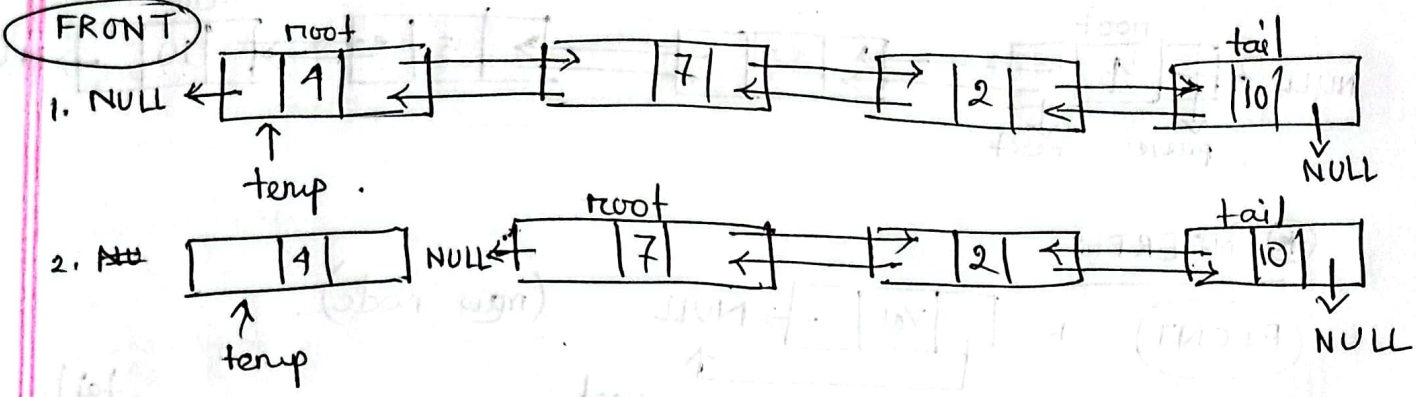
Sub :

Day

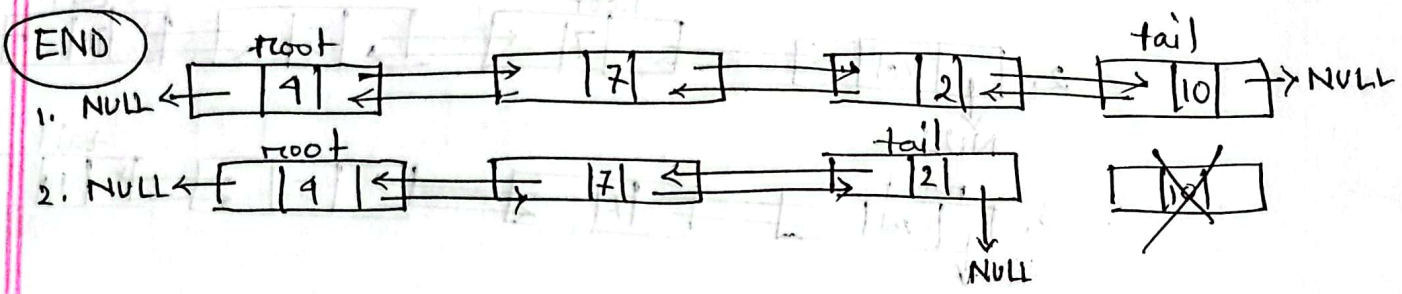
Time : Date : / /

⑤ Delete :

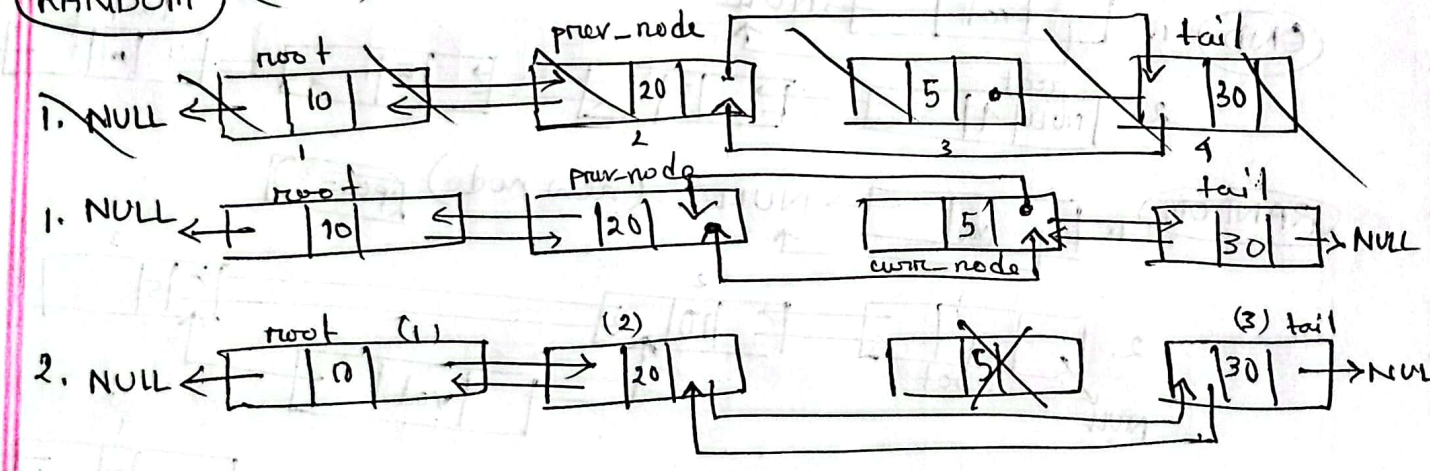
FRONT



END



RANDOM (pos=3)



Sub :

Day

Time :

Date : / /

Singly	Doubly
① 2 fields → data and address of next node.	① 3 fields → add prev_node, data, add next node
② One way traverse	② Both way
③ less memory	③ More memory
④ TC of insertion + delete at a known pos O(n)	④ TC of known pos O(1)
⑤ One list-pointer root	⑤ 2 list-pointer root and tail.

ARRAY	LINKED LIST
① Stored in contigious location.	① No
② Fixed size	② Dynamic
③ Memory allocation at compile time.	③ at run time
④ less memory	④ More memory
⑤ easy access	⑤ access needs traversing
⑥ Insert + delete takes time	⑥ Faster.

Linked list features:

* Consecutive elements connected by pointer.
* Size not fixed.
* Last node points to null.
* Memory (extra) used but not wasted
* Entry point is root.

Tree

Sub :

Day

Time : Date : / /

Uses : ① Insertion + deletion effective
② less TC
③ Can be implemented for stack, queue., abstract ds
④ Represent trees + graphs
⑤ DNA.

Uses in REAL world :

① Music player
② Web browser
③ Image viewer
④ alt + tab.
⑤
⑥ ctrl + z

STACK :

Stack is a linear list where any element is added at the top of the list and any element is deleted (accessed) from the top of the list. Add operation for a stack is called 'push' operation and deletion operation is called 'pop' operation. Stack is LIFO (Last In First Out) ds. That means the element which was added last will be deleted or accessed first.

push(val); ⟶ insert_first
pop(); ⟶ delete_first
top(); ⟶ last_node
size(); ⟶ cnt.
empty(); ⟶ null ahiye check.

Sub: QUEUE

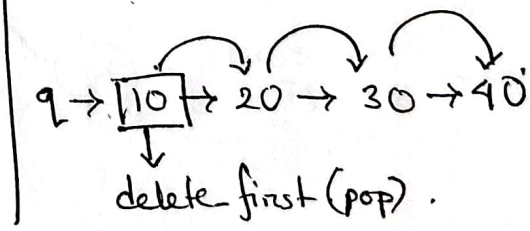
Day
Time: Date: / /

Queue is a linear list where all additions are made at one end, called 'rear' and all deletions (accesses) are made from another end called 'front' of the list. So, in a queue there are generally two pointers. Rear pointer is used to add elements and front pointer is used to delete (access) elements. Queue is a FIFO (First In First Out) data structure that means element that is added first will be deleted (accessed) first.

C++

```
1. #include<...>
2. using .....
3. int main()
4. {
5.   queue<int>q;
6.   q.push(10);
7.   q.push(20);
8.   q.push(30);
9.   q.push(40);
10.  cout<<q.size<<endl;
11.  while(q.empty())
12.  {
13.    int val = q.front();
14.    cout<<val<<endl;
15.    q.pop();
16.  }
17. }
```

push(val) [insert_last]
pop() [delete first]
front()
size()
empty()



Sub :

Day

Time :

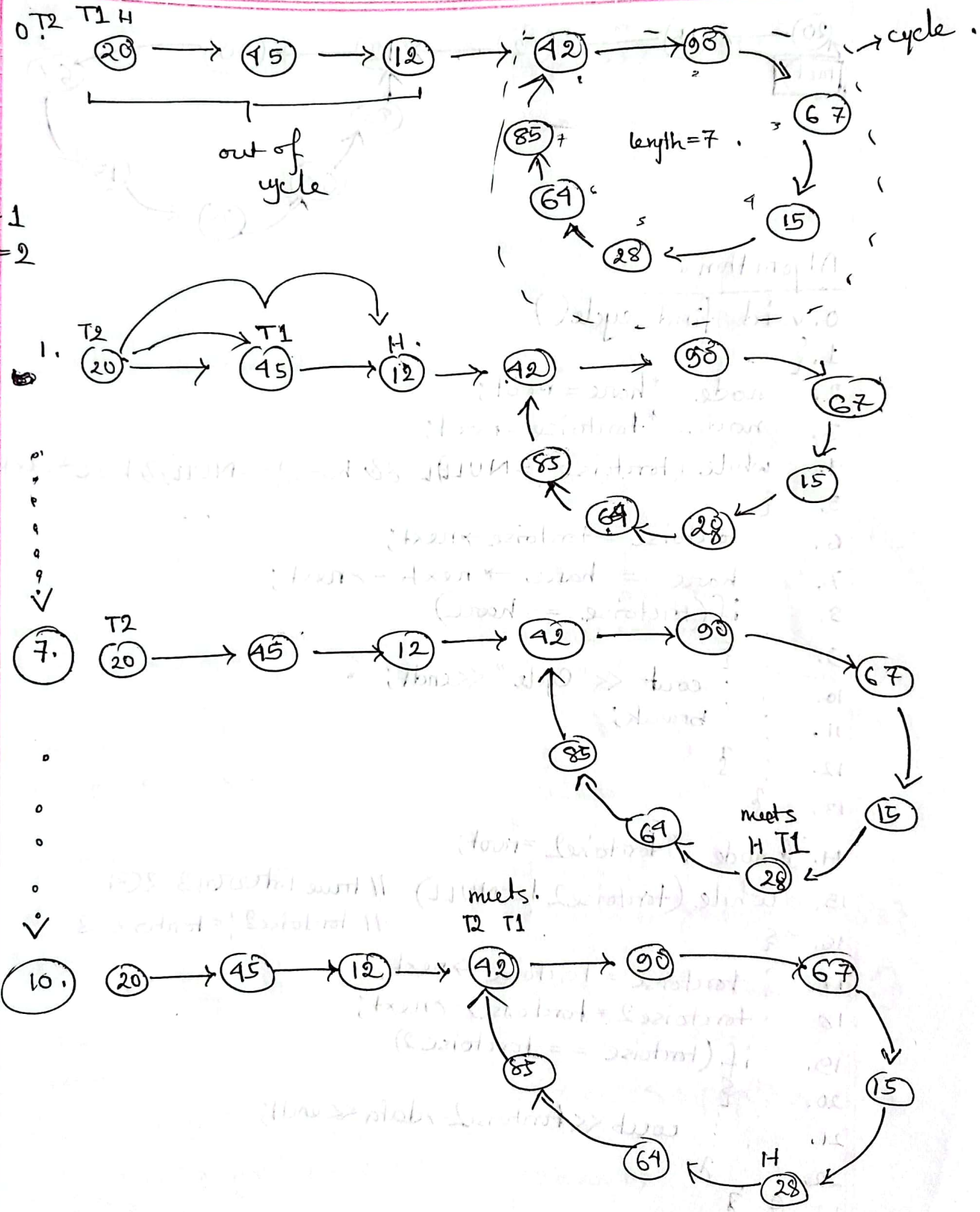
Date : / /

Stack	Queue
1. LIFO ; FILO.	1. FIFO ; LILO.
2. Insert + delete – same end	2. Insert + delete – diff. end.
3. One pointer → top.	3. two pointers → front, rear.
4. Singly	4. Doubly.
5. last inserted to come first	5. first inserted to come first
6. Push Pop. insert_f delete_f	6. Enqueue Dequeue insert_Last delete_f.
(Visualize) 7. Vertical collection	7. Horizontal collection
8. dinner plates one afteranoth (jar).	⑧ People in a bus boarding line.

Sub: Floyd's Cycle (Hare and tortoise).

Day

Time: Date: / /

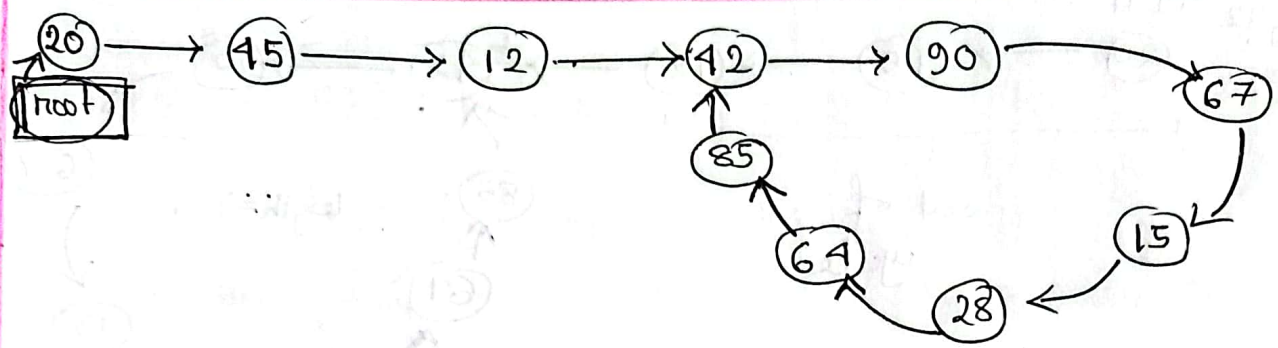


If H and T meets — cycle আছে — T1 T2 meets cycle - তে start.
start node = 0 indegree

Sub :

Day

Time : Date : / /



Algorithm:

```
0. void find_cycle()
1. {
2.    node *hare=root;
3.    node *tortoise=root;
4.    while (tortoise!=NULL && hare!==NULL && hare->next!=NULL)
5.    {
6.       tortoise = tortoise->next;
7.       hare = hare -> next -> next;
8.       if(tortoise == hare)
9.       {
10.         cout << "Cycle" <<endl;
11.         break;
12.      }
13.   }
14.   node *tortoise2 =root;
15.   while (tortoise2 !=NULL)  // true থাকলেও হবে
16.   {                         // tortoise2 != tortoise 3.
17.      tortoise = tortoise -> next;
18.      tortoise2 = tortoise2 -> next;
19.      if (tortoise == tortoise 2)
20.      {
21.         cout << tortoise2 -> data << endl;
22.      }
23.   }
24. }
```

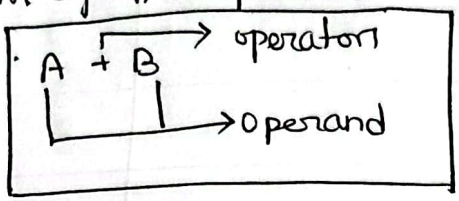
**
* (VVI)

Sub: Infix, Prefix, Postfix.

Day
Time: Date: / /

An arithmetic expression can be represented in various forms such as prefix, infix or postfix. 'Pre', 'In', 'Post' refer to the relative position of the operation with respect to its operands.

For example:



Operators: +, −, *, /
Operands: Others.

Infix: <operand><operator><operand>
Prefix: <operator><operand><operand>
Postfix: <operand><operand><operator>

Precedence/Priority	Associativity
1. () {} []	
2. ^	R-L
3. * /	L-R
4. + -	L-R

Example:

Pre

6 + 2 * 7
6 + * 2 7
6 + a
+ 6 a
+ 6 * 2 7

5 + 10 * 90 / 2
5 + * 10 90 / 2
5 + a / 2
5 + / a 2
5 + b
+ 5 b
+ 5 / a 2
+ 5 / * 10 90 2

Post

6 + 2 * 7
6 + 2 7 *
6 + a
6 a +
6 2 7 * +

5 + 10 * 90 / 2
5 + 10 90 * / 2
5 + a / 2
5 + a 2 /
5 + b
5 b +
5 a 2 / +
5 10 90 * 2 / +

A * B + C / D.
* AB + / C D
+ * AB / CD

A * B + C / D.
A B * + C D /
A B * C D / +

2 ^ 2 ^ 3
2 ^ 8
~~256~~
2^8.

Sub : Infix to Postfix Conversion.

Day
Time : Date : / /

Given Expression, $5 * (6+2) - (12/4)$

Symbol scanned	Stacks	Postfix expression
5		5
*	*	5
(	*(	5
6	*(	56
+	*(+	56
2	*(+	562
)	*(+)	562+
-	-	562+*
(	-(	562+*
12	-(	562+*12
/	-(/	562+*12
4	-(/	562+*124
)	-(/)	562+*124/
		562+*124/-

↳ step == character (or char+1)

Sub :

Day

Time : Date : / /

RULES:

① যদি symbol = operand হয়, তাহলে add to postfix.

② যদি symbol = opening bracket হয়, তাহলে push to stack

③ যদি symbol = operator হয়, তাহলে check top of the stack

(i) priority (symbol) > priority (top of stack) হয়,
push to stack.

(ii) Otherwise, pop one by one until (i) is false, add to postfix, push symbol to stack

④ যদি symbol = closing bracket হয়, pop one by one "opening bracket" পাওয়া পর্যন্ত, add to postfix.

⑤ Stack যদি empty না হয়, pop one by one, add to postfix.

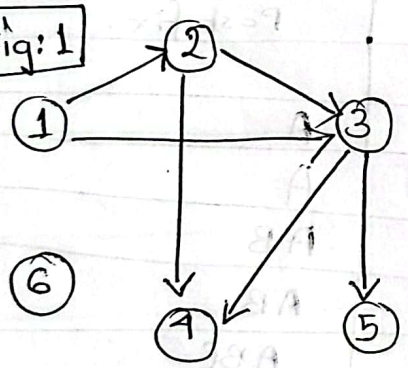
Sub : GRAGH.

Day
Time : Date : / /

A graph is a set of nodes (vertices) and edges. The node that holds data is called vertex and the line connecting two vertices is called edge. If G denotes a graph, G=(V, E) where V denotes set of vertices and E denotes set of edges.

① Directed / Undirected.
② Undirected.
③ Bidirected
④ Weighted / unweighted.

Fig: 1



↓ Disconnected.

unweighted
unidirected

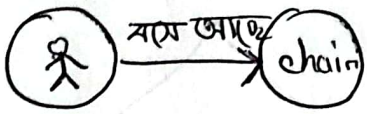
* connection নাই graph

n > 0
e = 0
possible.

* PATH :- Sequence of vertices where each pair of successive vertices is connected by an edge.

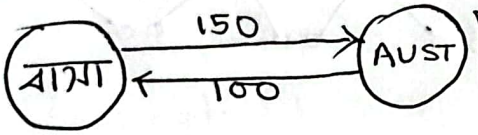
* Connected Graph :- A graph is called connected, if there is a path between each pair of vertices.

* CYCLE :- A path where the first and last vertices are the same.



[Directed – unidirected]

[bothway] — undirected.



[Bidirected] ভিন্ন weight আছে v_2

Possible	not possible
$n=e$.	$e<0$
$n>e$	$n<0$.
$n<e$	
$n=e-1$	

Input : node(no.), edge(no.)
Here . to there

Fig 1:
```
5 6
1 2
2 3
1 3.
2 4
3 4
3 5.
```

Source node → indegree=0
Destination " → outdegree=0

*Shortest-path problem → 1 → 3 → 5.
1 → 2 → 3 → 5.

Adj Matrix :

1. determine if edge exists in a path
2. $O(|V|^2)$ space
3. too many edges.
4. adding a vertex is $O(|V|^2)$ + remove
5. " " edge is $O(1)$ + remove
6. Querying → $O(1)$.

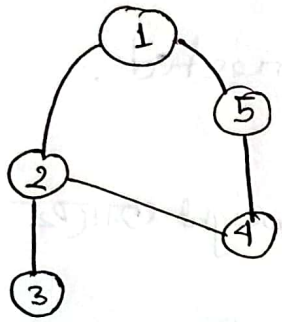
Adj list:

1. Find adjacent vertices
2. Space C. $O(|V| + |E|)$
3. add new vertex/edge $O(1)$
   [2 pointers used for rear & front]
4. Remove vertex $O(|V| + |E|)$ worst
   $O(|V|)$ best
5. Remove edge, traverse all edges $O(|E|)$
6. Querying $O(|V|)$

Sub : BFS(Breadth First Search)

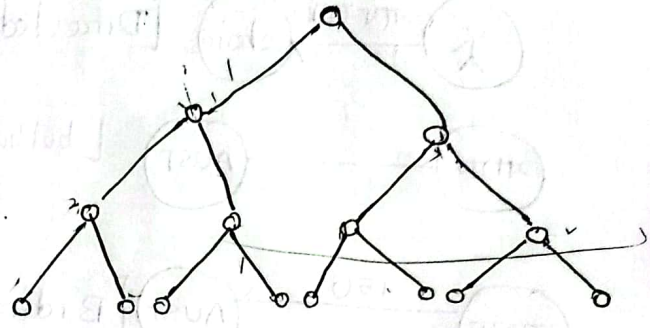
Day
Time : Date : / /

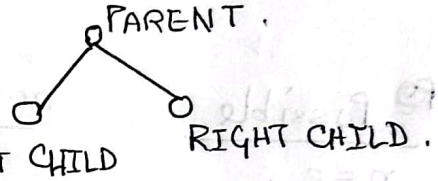
Adjacency List :



1→2,5
2→1,3,4
3→2
4→2,5
5→1,4.

{undirected. unweighted}





Graph traversal algo.

0. BFS (source).
1. Q = queue().
2. level[] = -1 or ∞
3. Q.push(source).
4. level[source] = 0.
5. while Q not empty:
6. u = Q.front()
7. Q.pop()
8. For all adjacent edges from u to v.
9. v = adjacent edge node.
10. if level[v] = -1 or ∞
11. Q.push(v).
12. level[v] = level[u] + 1.
13. end if
14. end for
15. end while.

graph-এ use
shortest path detection

** v child
u parent

Sub: GRAGH TRAVERSAL ALGORITHM

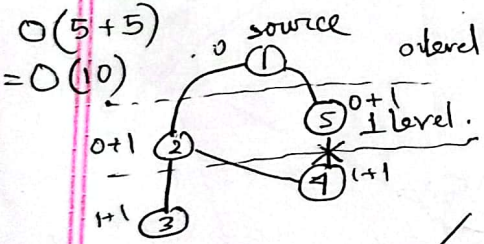
Day
Time: Date: / /

যব Graph কি tree?
" tree " Graph ?

BFS
* graph traverse
* unweighted - যে কাজ
* shortest path detection.
Time complx O(|V|+|E|)
* queue
* height- measure

DFS.
* stack.
* shortest path - (not recommended)
* Find vertexes
* cut vertices of graph.
* maze solve

BFS Mechanism:



index	level 0	1	2	3	4	5
value	-1	-1	-1	-1	-1	-1

1 বারপর value: -1 | 0 | -1 | -1 | -1 | -1

1 pop: -1 | 0 | 1 | -1 | -1 | -1

2 " : -1 | 0 | 1 | -1 | -1 | 1

pop() → 2

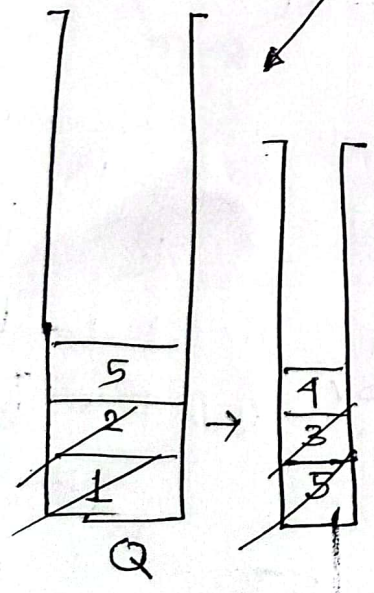
3 বারপর: -1 | 0 | 1 | 2 | -1 | 1

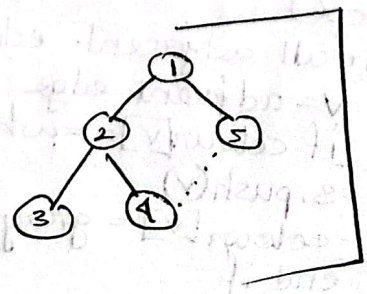
4 " : -1 | 0 | 1 | 2 | 2 | 1 → Answer.

pop → 5

pop → 3.

source





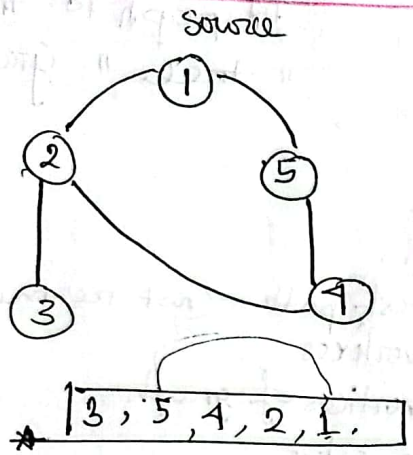
BFS tree.

Complexity → O(|V|+|E|)

Sub: DFS (Depth First Search)

Day

Time: Date: / /



Adjacency List:

1 → 2, 5
2 → 1, 3, 4
3 → 2,
4 → 2, 5
5 → 1, 4.

Colour:

white = যে নোড এখনো visit করা হয় নাই

gray = যে নোড নিয়ে কাজ চলছে

black = যে নোডের কাজ শেষ

RECURSIVE:

```
0. DFS (source)
1. colour[source]=gray.
2. For all adjacent edges from source to v.
3.     v=adjacent edge node
4.     if colour[v]=white.
5.         DFS(v)
6.     end if
7. end for
8. colour[source]=black.
```

ITERATIVE:

```
0. DFS(source)
1. S=stack()
2. colour=white
3. S.push(source)
4. colour[source]=gray
5. while S not empty
6.     v=S.top()
7.     S.pop()
8.     For all adjacent edges from v to v
9.         v=adjacent edge node
10.        if colour[v]==white
11.            S.push(v).
12.            colour[v]=gray
13.        end if
14.    end for
15. end while
```

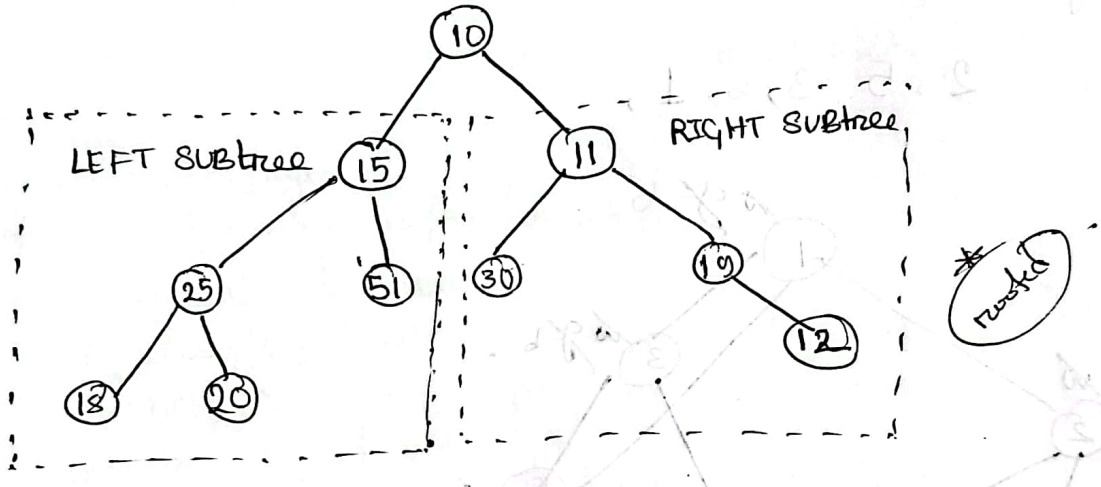
Sub : TREE

Day

Time : Date : / /

A tree is a finite collection of nodes that reflects one to many relationship among the nodes. An ordered tree has a specially designated node called ROOT node. The node that has no child node is called LEAF node. The nodes of a level are connected to the nodes of the upper level.

Fig - 1



Basic

ROOTED BINARY TREE: A tree that has a root node and every node has at most 2 children.

FULL BINARY TREE: A binary tree in which every node has either 0 or 2 children.

PERFECT BINARY TREE: A binary tree in which all interior nodes has 2 children and all leaf nodes are on the same level.

COMPLETE BINARY TREE: A binary tree in which every level except possibly the last is completely filled and all nodes in the last level are as left as possible.

Sub : _____ Day | Time : Date : / /

In a binary tree ① number of levels = k

maximum no. of nodes in the tree, $n = 2^k - 1$.

② maximum no. of nodes = n.

no. of levels, $k = \lceil \log_2 (n+1) = k \rceil$

③ If the position of a node = i.

then " " " left child = $2i$

" " " right " = $2i + 1$

" " " parent node = $\lfloor i/2 \rfloor$

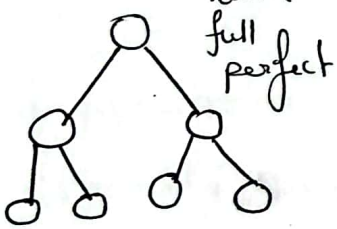
TREE TRAVERSAL TECHNIQUES :

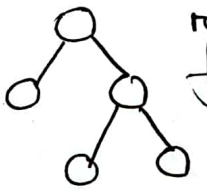
① Pre-order (root, left, right)

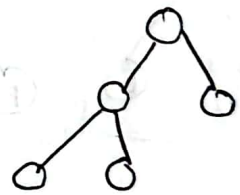
② In-order (Left, root, right)

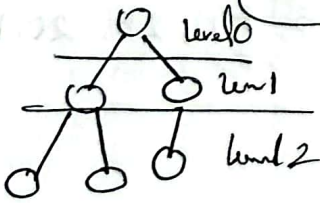
③ Post-order (Left, right, root).

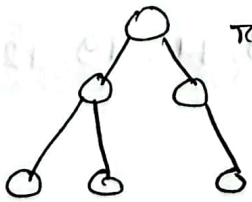
Example







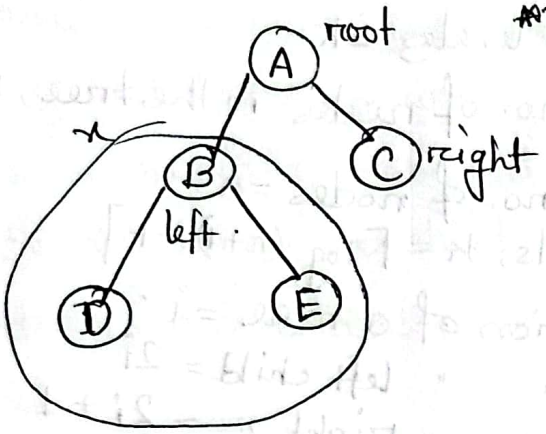




Sub :

Day

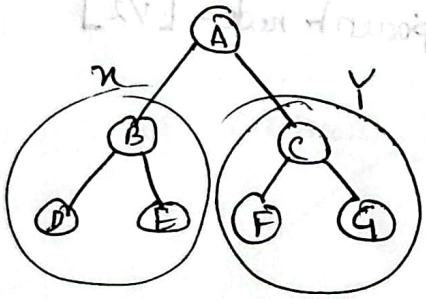
Time : Date : / /



Preorder :

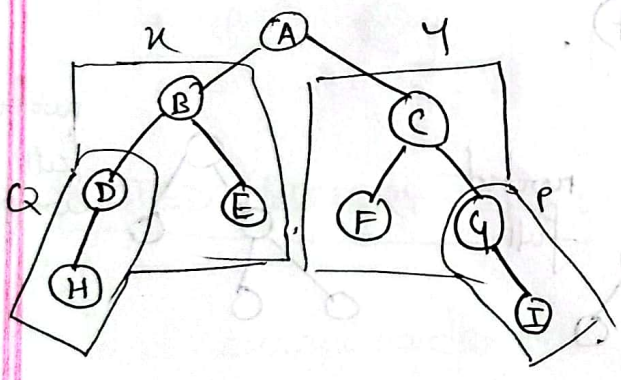
A x C

→ A B D E C.



A x Y

→ A B D E C F G



A x Y

A B Q E C F P.

A B D H E C F G I

Fig: 1 pre order = 10, 15, 25, 18, 20, 51, 11, 30, 19, 12

post order = 10, 11, 19, 12, 30, 15, 51, 25, 20, 18.

Sub: Tree codes.

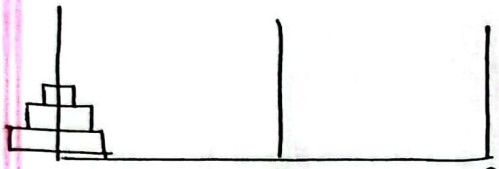
Day

Time : Date : / /

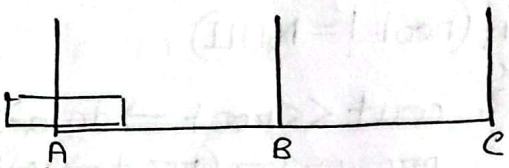
```
Void pre_order(node *root)
{
    if(root != NULL)
    {
        cout<<root->data<<endl;
        pre_order(root->left);
        pre_order(root->right);
    }
}


void in_order(node *root)
{
    if(root != NULL)
    {
        inorder(root->left);
        cout<<root->data<<endl;
        in_order(root->right);
    }
}


void post_order(node *root)
{
    if(root != NULL)
    {
        post_order(root->left);
        post_order(root->right);
        cout<<root->right data<<endl;
    }
}
```

Sub: Tower of Hanoi (Recursion). Day Time: Date: / /

More three discs

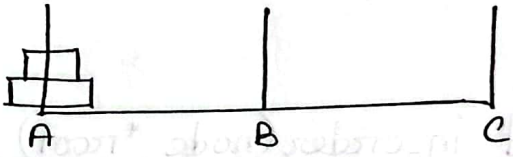


For one disc



* More A to B.

For two discs



* More 1 disc (from A) (to B) (using C)
* More A to C.
* More 1 disc (from B) (to C) (using A).

Order in Common Sense

1. A → C
2. A → B
3. C → B
4. A → C
5. B → A
6. B → C
7. A → C

Algo:

* More 2 discs (from A) (to B) (using C)
* More A to C.
* More 2 discs. (from B) (to C) (using A).

Common Algo:

* More (n-1) discs (from (A) (to B) (using C)
* More A to C.
* More (n-1) discs (from B) (to C) (using A).

Sub :

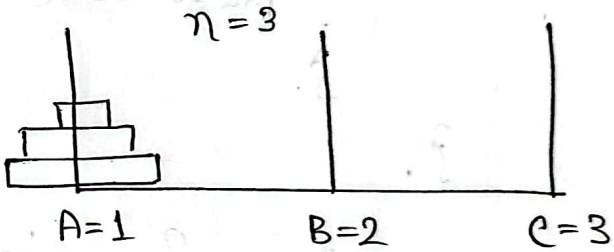
Day

Time : Date : / /

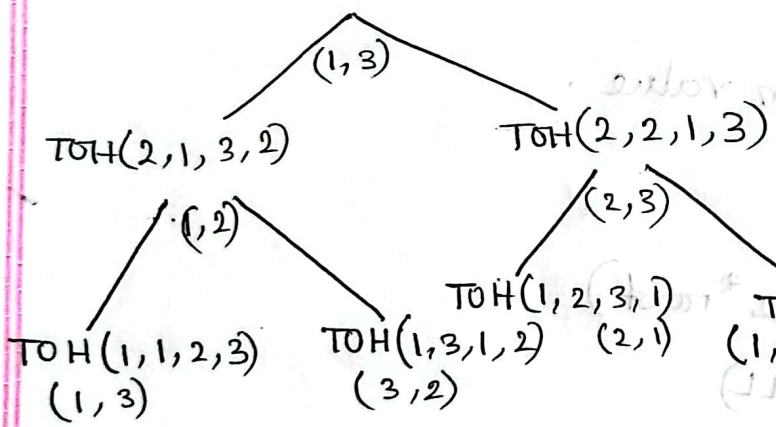
Code :

FROM USING TO

```
void TOH( int n , intA , intB , intC)
{
    if(n>0)
    { TOH(n-1, A , C , B)
      cout << A << " → " << C << endl;
      TOH (n-1, B , A , C)
    }
}
```



TOH (3, 1, 2, 3)



Order :

1. (1, 3)
2. (1, 2)
3. (3, 2)
4. (1, 3)
5. (2, 1)
6. (2, 3)
7. (1, 3)

$n=3$ # more $= 2^3 - 1 = 7$.
$= 2^n - 1$

$n=10$ # more $= 2^{10} - 1 = 1023$.

$n=1000$ # more $= 2^{1000} - 1$

Time Complexity $= O(2^n - 1)$

Space Complexity $= O(n)$

Exponential

Linear.

Sub :

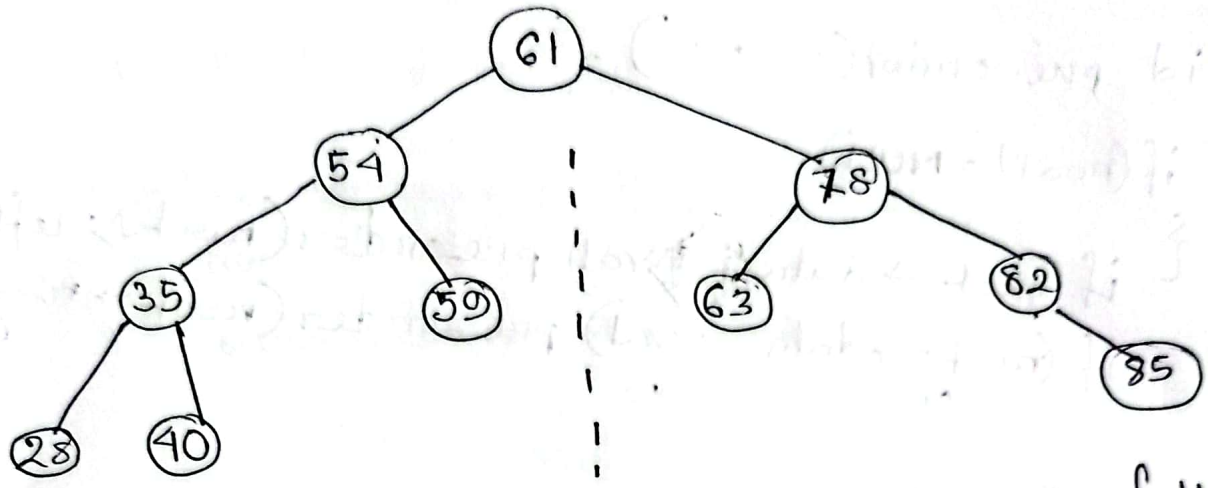
Day

Time :

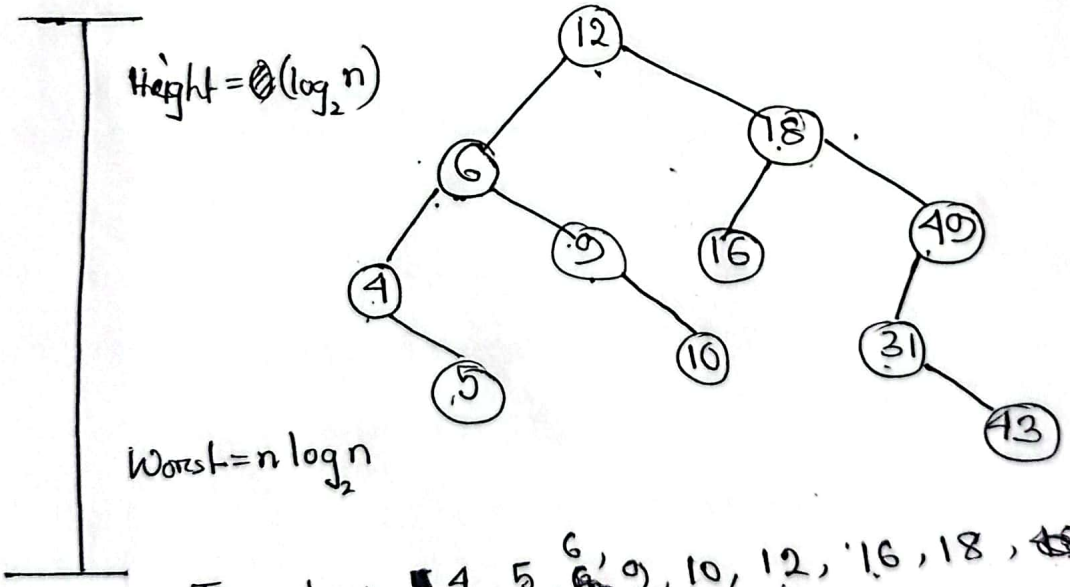
Date : / /

Binary Search Tree :

BST:



* Construct a Binary Search Tree by inserting the following numbers from left to right : 12, 6, 9, 18, 4, 10, 5, 16, 49, 31, 43.

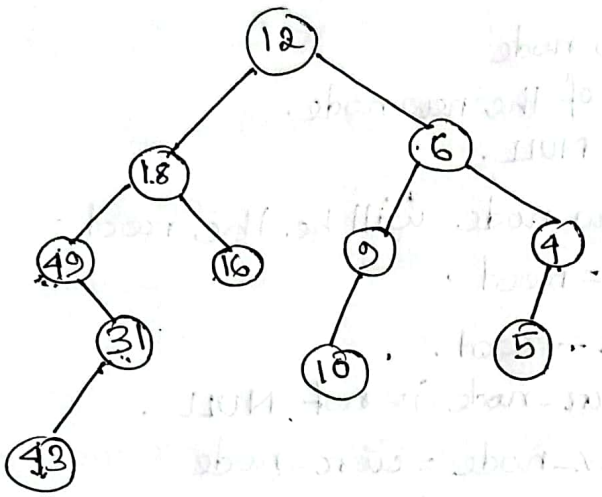


Inorder : 4, 5, 6, 9, 10, 12, 16, 18, 31, 43, 49

Sub: Practice.

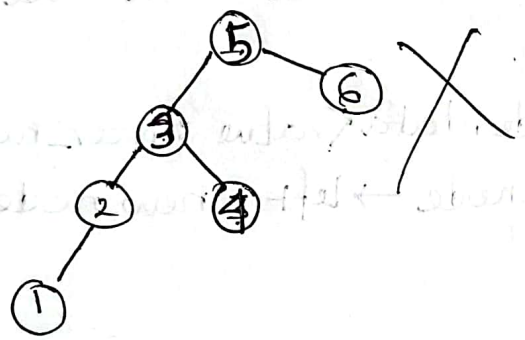
Day
Time : Date : / /

Descending



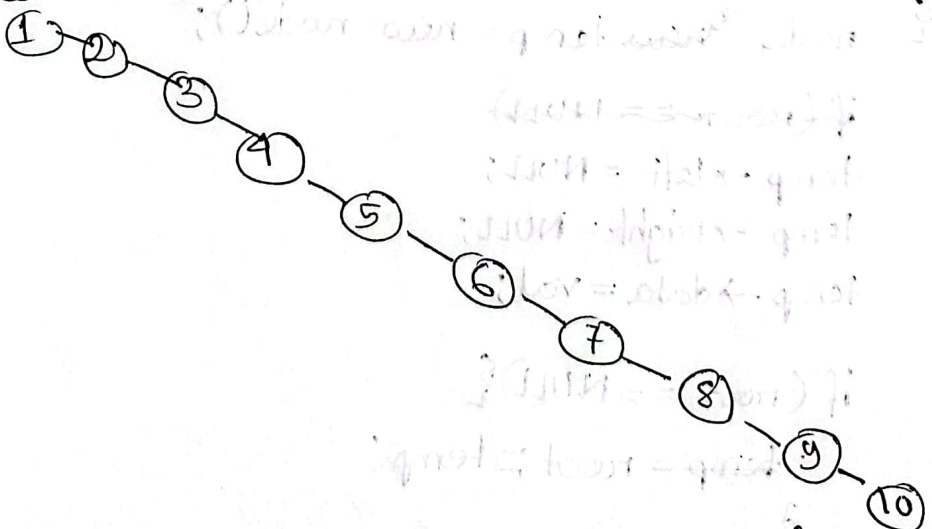
Inorder: ~~12, 18, 49,~~ 49, 43, 31, 18, 16, 12, 10, 9, 6, 5, 4

~~(*) (1-10)~~



(*) (1-10) sorted BST বানাব।

Solution:



(*) Unorder set-দেওয়া — apply BST — একবার search function লিখলেই হবে।

Sub : BINARY SEARCH TREE ALGORITHM.

Day

Time : Date : / /

0. Insert (value).
1. create new node
2. set values of the new node.
3. If root is NULL.
4. the new node will be the root.
5. curr_node = root.
6. prev_node = root.
7. while curr_node is not NULL.
8. prev_node = curr_node
9. if curr_node.data < value : curr_node = curr_node -> right.
10. else curr_node = curr_node -> left.
11. end while.
12. If prev_node.data < value : prev_node -> right = new.node
13. else prev_node -> left = new node.

Code :

```
void Insert(~~Valo~~ int val).
{  node *~~new~~ temp = new node();
   ~~if(root == NULL)~~
   temp -> left = NULL;
   temp -> right = NULL;
   temp -> data = val;

   if (root == NULL){
      ~~temp~~ = root = temp;
   }.

   root = curr_node;
   root = prev_node;
```

Sub :

Day

Time : Date : / /

```
while (curr_node != NULL)
{
    prev_node = curr_node
    if (curr_node -> data < val)
    { curr_node = curr_node -> right;
    }
    else{
    curr_node = curr_node -> left;
    }
}
if (prev_node -> data < val) prev_node -> right = temp;
else prev_node -> left = temp;
}.
```

Sub :

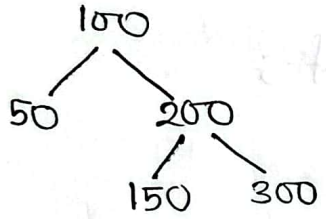
Day

Time : Date : / /

✱ Delete a node from Binary Search Tree

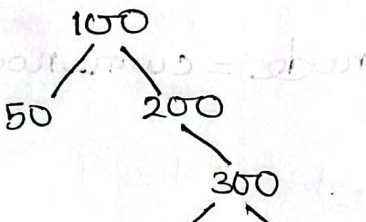
3 cases:
① node with no child
② node with one child
③ node with two child.

① NO child (leaf node)



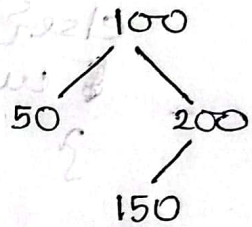
delete(300)

② right ONE child



delete(200)

2.2 ③ ~~Two~~ left child



delete(200)

```
if(curr->left==NULL || curr->right==NULL)
{  node *child;
   if(curr->left==NULL) child = curr->right;
   else child = curr_node->left;
} //(12-16) line if(prev==NULL) child = root;
  //(12-16) line
```

```
0. void deleteNode(node *root, int val).
1. {  node *curr_node = root;
2.
3.    node *prev_node = root;
4.    while (curr_node != NULL)
5.    { if (curr_node->data == val) break;
6.      prev_node = curr_node;
7.      if (curr_node->data < val) curr_node = curr_node->right;
8.      else curr_node = curr_node->left;
9.    }
10.   if (curr_node == NULL) return;
```

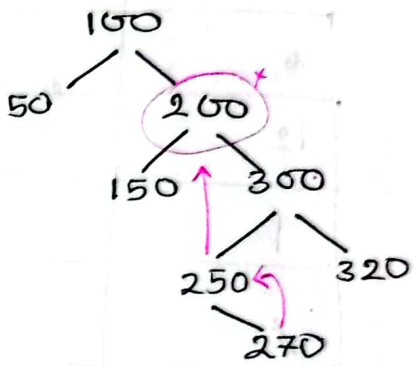
Sub:

Day

Time: Date: / /

```
11. if (curr_node -> left == NULL && curr_node -> right == NULL)
12. { if (prev_node == curr_node)
13.       prev_node -> left = NULL;
14.   else prev_node -> right = NULL;
15.   return;
16. }.
17. }.
```

Node with two child:



2Child code:

```
Void delete2Childnode (node* root, int val)
{
   while ( )
   if
   if
   if
   else {
   node *temp = curr_node;
    prev_node = curr_node;
    curr_node = curr_node -> right;
    while (curr_node -> left != NULL)
    {   prev_node = curr_node;
        curr_node = curr_node -> left;
    }
   temp -> data = curr_node -> data;
    prev_node -> left = curr_node -> right;
}
}
```

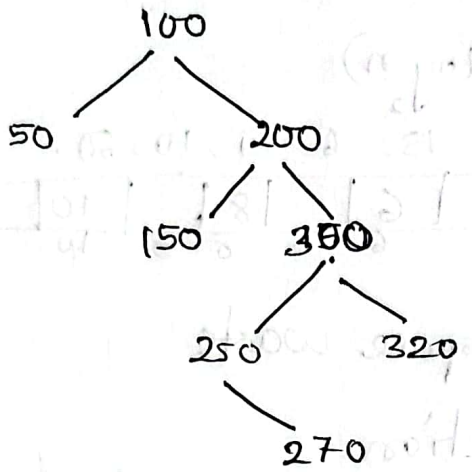
Same as before

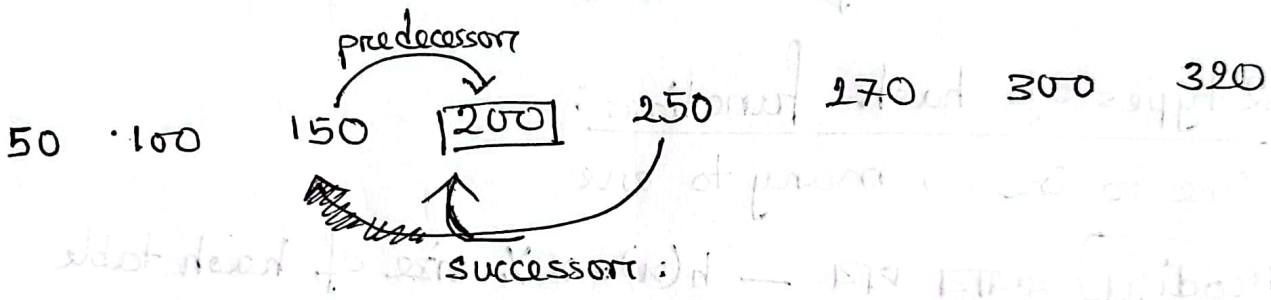
```
// for left
curr_node = curr_node -> left;
while (curr_node -> right != NULL)
{ ..... ;
curr = curr -> right.
}
prev -> right = curr -> left;
```

Sub: Tree – (successor and predecessor)

Day

Time : Date : / /





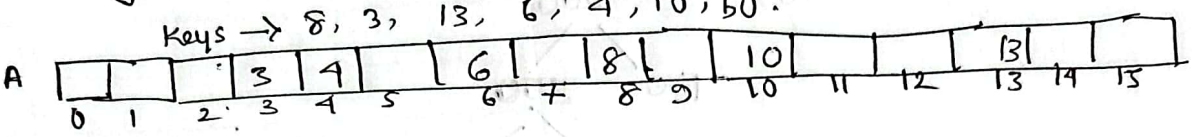
Sub: HASHING (searches) O(1)

Day

Time: Date: / /

Linear Search — $O(n)$

Binary Search — $O(\log_2 n)$



Problem — memory space waste

$h(x) = x$; hash function.

2 types of hash functions:

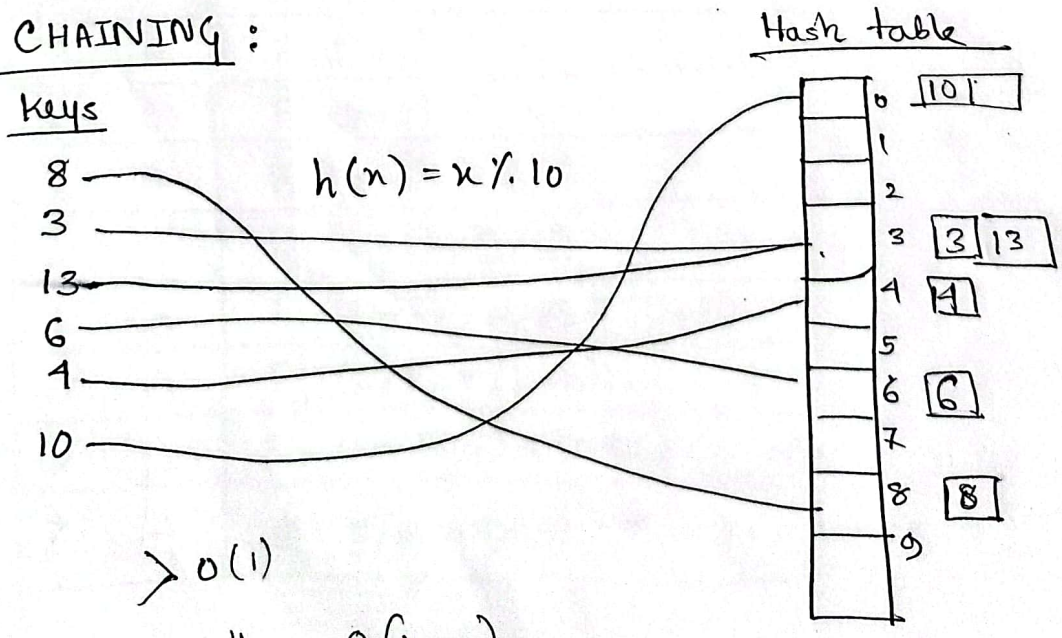
One to One, many to one

Modify করার পর — $h(x) = x \% $ size of hash table

to save memory → কম জায়গা collision হয়।

To solve collision: → Chaining
→ linear probing
→ quadratic probing

CHAINING:



> $O(1)$

less than $O(\log_2 n)$

Sub: Hash Collision.

Day
Time : Date : / /

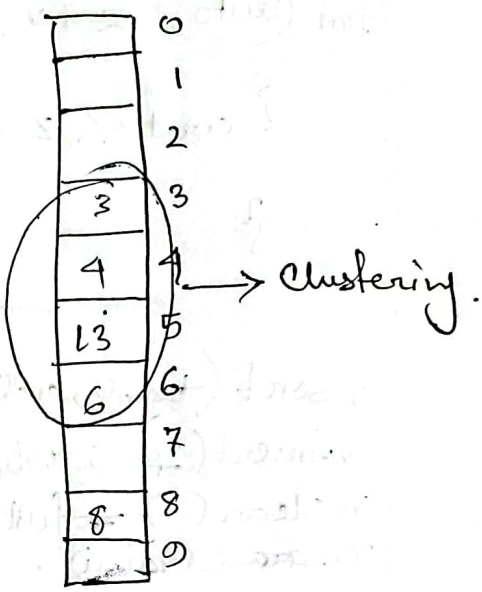
Key space

8
3
6
4
10
13

hash function

$h(x) = x \% 10$

Hash Table



Linear Probing

$h'(x) = [h(x) + f(i)] \% size.$

$f(i) = i \; ; \; i = 0, 1, 2$

$h'(13) = [h(13) + f(0)] = [3 + 0] = 3$ collision.

$h'(13) = [h(13) + f(1)] = [3 + 1] = 4$ collision

Quadratic Probing.

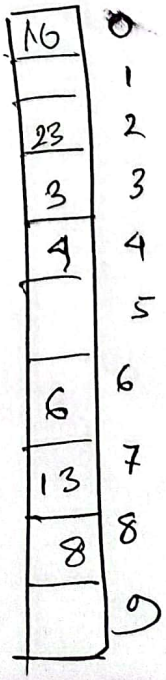
$h'(x) = [h(x) + f(i)] \% size.$

$f(i) = i^2 \; , \; i = 0, 1, 2.$

$h'(23) = [h(13) + f(0)] = [3 + 0] = 3$ collision

$h'(23) = \quad f(1) = [3 + 1] = 4$ "

$h(23) = \quad f(2) = [3 + 2] = 5$ no collision



Sub: Definitions (DS)

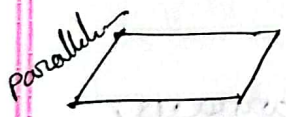
Day
Time: Date: / /

Pseudocode: Atartificial and informal language that helps programmers develop algorithms — similar to everyday English.

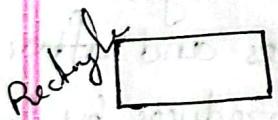
Algorithm: Set of instructions, step by step representation of pseudocode, very close to computer language.

Flowchart: A graphical representation of the sequence of operations (algorithms) in an information system or program, shows how data flows from source documents to end users.

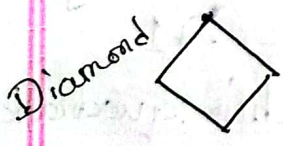
→ start or end



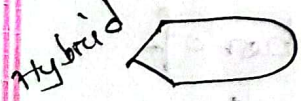
→ input operation.



→ formula



→ if, else



→ output.


flow line

Sub : Complexity Analysis (Sorting Algos)

Day

Time : Date : / /

Binary Search :

first iteration : n
2nd " : n/2
3rd " : (n/2)/2 : $n/2^2$
k^{th} " : $n/2^k$

or, $\log_2 n = \log_2 2^k$

or, $\log_2 n = k \log_2 2$

or, $k = \log_2(n)$ [$\because \log_a a = 1$]

So, time complexity is $O(\log n)$.
Best case : central index $O(1)$
Worst " : values at extremity of the list or not present.

Space complexity: iterative $O(1)$
recursive $O(\log(n))$ as all the calls will be stacked in memory.

Selection Sort :

time Complexity : If n number of arrays,
Iterates unsorted subarray (n-1) times and after every iteration the size of the subarray reduces by 1.
So, comparisons are $(n-1)+(n-2)+(n-3)+\ldots+1$.
Thus, overall complexity is $n*(n-1)/2$ which is quadratic
As, this algorithm performs same number of comparisons for all ~~the~~ given array of size n, so
best = worst = average = $O(n^2)$

Space : $O(1)$ cause no extra space needed.

Sub :

Day

Time : Date : / /

Insertion Sort: Space : O(1)

time : Best case : Already sorted ; C*(n) or O(n).

Worst " : The array is sorted in reverse, Algo picks the first element from the unsorted subarray and places it at the begining of the sorted subarray. So, comparisons is N*(N-1)/2 or O(N^2)

Average : same as worst case.

Merge sort :

Space : An extra array is needed to store merged array, so O(n).

time : regardless of input array, it performs in the same way. It divides the array recursively into two subarrays of equal size will create Logn sub arrays. Then it repeatedly merges into 2 sorted subarrays which takes linear time(n). So, O(nlogn).

best = worst = average = O(nlogn)

Quick Sort : Space :

Additional memory is needed for stack frames in recursive calls. Space complexity depends on pivot.

time) Best : If partition is done in arrays in equal halves, each time, then size of recursion tree is logN.

So, O(nlogn)

Sub :

Day

Time : Date : / /

worst : If the pivot is either largest or smallest element- then total N recursive calls there are. So recursive tree's size will be N. So, O(N).

time : best : if the pivot is media ~~in~~ each time, then. algo creates logN subarrays. So, time complexity will be O(NlogN).

worst : when array is already sorted, we select the leftmost elements as pivot, so algo recursively creates N subarrays of size N, N-1, N-2 ....1.
→ sorted arrays take linear time for partitioning, resulting in quadratic time complexity.

average : if we want we can choose median of or last elements as pivot. or randomly generating a pivot for each subarray. By these methods we can ensure equal partitioning on average. So average tc is O(nlogn).