(1) Insertion Sort

5 17 9 4 -1 10 2

- Marker divides the list into a sort & unsorted part

iteration 1: 5 17 | 9 4 -1 10 2

check if 2nd element is in position → yes
move marker to right

iteration 2: 5 17 9 | 4 -1 10 2

move 9 to correct position

5 9 17 | 4 -1 10 2

move marker to right

iteration 3: 4 5 9 17 | -1 10 2

check if 17 is in position

move marker to right

iteration 4: 4 5 9 17 -1 | 10 2

move -1 to the correct position

-1 4 5 9 17 | 10 2

move marker to right

iteration 5: -1 4 5 9 17 10 | 2

move 10 to the right position

-1 4 5 9 10 17 | 2

move marker to the right

iteration 6: -1 4 5 9 10 17 2 |

move 2 to correct position

sorting Techniques (1)

→ insertion
→ shell Sort
→ Radix Exchange sort
→ Dijkstra's shortest Path
→ DFS
→ BFS
→ Binary Heap
→ Priority Queue
→ Expression trees
→ BST

-1 2 4 5 9 10 17 |

end

Algorithm

best $O(n)$
worst $O(n^2)$

```
n ← length(arr)
for i = 1 to n
    item ← arr[i]
    j ← i-1
    while j >= 0 and item < arr[j]
        arr[j+1] ← arr[j]        // swapping elements
        j ← j-1
    end while
    arr[j+1] ← item
end for
```

* Shell Sort

1. Take the input, find size (n)
2. Initialize the gap values as floor(n/2)
3. Compare 2 elements at the distance of the gap at every iteration.
4. For a given gap, if there are 'gap' no. of elements in the left, compare with the back too.
5. Repeat until the list is sorted.

Example

170 75 802 90 66 2 24 45 7

Pass1: gap = floor(n/2) = floor(9/2) = 4

(a) 170 75 802 90 66 2 24 45 7

66 170

(b) 66 75 802 90 170 2 24 45 7

2 75

(c) 66 2 802 90 170 75 24 45 7

24 802

(d) 66 2 24 90 170 75 802 45 7

45 90

66 2 24 45 170 75 802 90 7

7 170

66 2 24 45 75 24 90 170

*** 4 elements are present backwards also

After Pass1

∴ 7 2 24 45 66 75 802 90 170

Pass 2: Gap = floor(n/2) = 4/2 = 2

(a) 7 2 24 45 66 75 802 90 170

nochange

(b) 7 2 24 45 66 75 802 90 170

no change

(c) 7 2 24 45 66 75 802 90 170

no change

(d) 7 2 24 45 66 75 802 90 170

nochange

(e) 7 2 24 45 66 75 802 90 170

no change

(f) 7 2 24 45 66 75 802 90 170

no change

(g) 7 2 24 45 66 75 802 90 170

170 802

7 2 24 45 66 75 170 90 802

** There exists an element 2 positions before 170, compare again

⇒ leads to no change

After Pass 2:

7	2	24	45	66	75	170	90	802

Pass 3 : Gap = floor(n/2) = 2/2 = 1

(a) 2 7 2 24 45 66 75 170 90 802.

(b) 2 7 24 45 66 75 170 90 802

no change

(c) 2 7 24 45 66 75 170 90 802

no change

(d) 2 7 24 45 66 75 170 90 802

no change

(e) 2 7 24 45 66 75 170 90 802

no change

(f) 2 7 24 45 66 75 170 90 802

no change

(g) 2 7 24 45 66 75 ~~170~~ ~~90~~ 802

170 90

swap

(h) 2 7 24 45 66 75 90 170 802

** there exists an element 1 pos prior to 90, compare

→ leads to no change

(i) 2 7 24 45 66 75 90 170 802

no change

Final sorted array:

| 2 7 24 25 66 75 90 170 802 |

Algorithm

```
Shell sort ( array A[], int n)

    gap = n/2

    while gap > 0

        for i = gap+1 to n

            j = i - incr        // refers to index of item

            if A[j] > A[j+gap]

                swap(A[j], A[j+gap])

                j = j - incr

            end if

        end for

        gap = gap/2

    end while
```

* Radix Sort

1. Find maximum element
2. Find the no. of digits in the maximum element. (d)
3. Create d no. of buckets for the d no. of digits
4. for i = 0 to d, sort elements according to digits at the ith position.

Example

181 289 390 121 145 736 514 212

(i) Sort by 1's place

181 289 390 121 145 736 514 212

Pass1

390	181	121	212	514	145	736	289

(ii) Sort by 10's place

390 181 121 212 514 145 736 289

Pass 2

212	514	121	736	145	181	289	390

(iii) Sort by 100s place

212 514 121 736 145 181 289 390

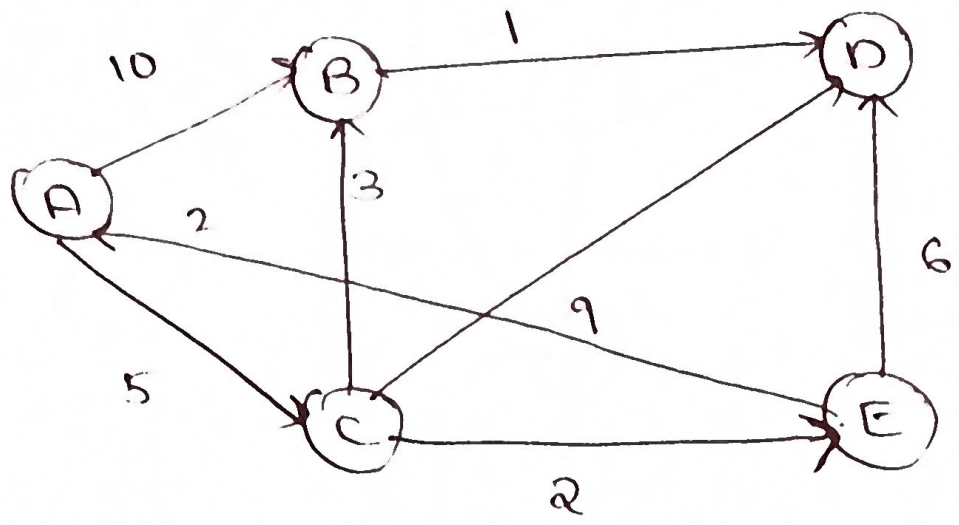
121	145	181	212	289	390	514	736

↳ Sorted Array

Algorithm

Dijkstra's Shortest Path Algorithm

Ex1

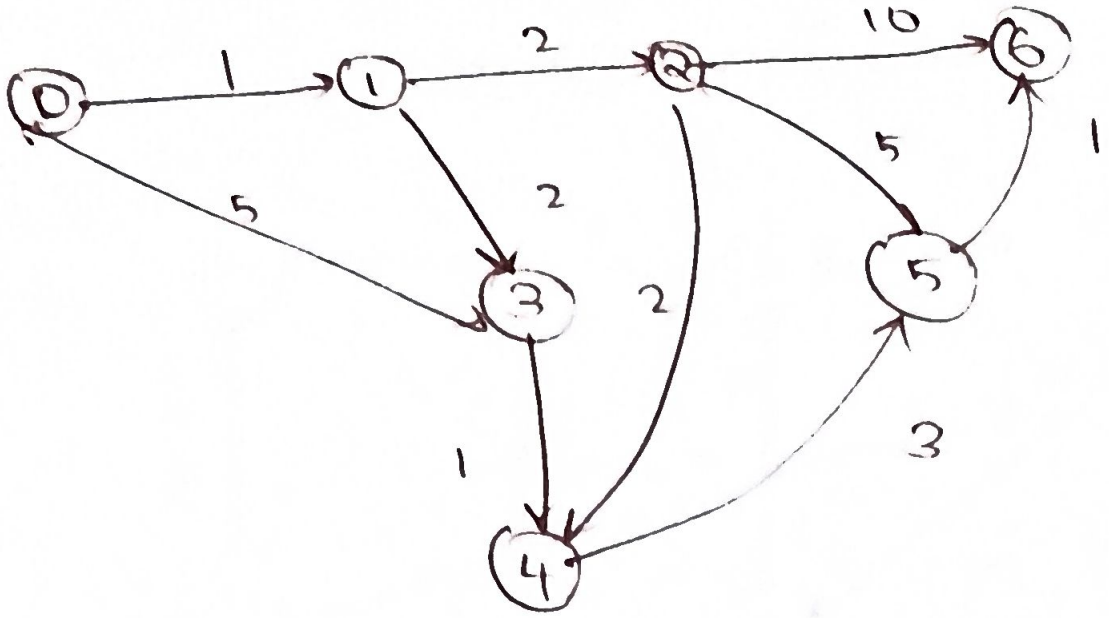


starting vertex : A.

Selected Vertex	A	B	C	D	E
A	[0]	∞	∞	∞	∞
C	[0]	10	[5]	∞	∞
E	[0]	~~8~~	[5]	~~14~~	~~8~~ [7]
B	[0]	[8] ←	[5]	13 ↰	[7]
D	[0]	[8]	[5]	[9]	[7]

To find path : D B C A

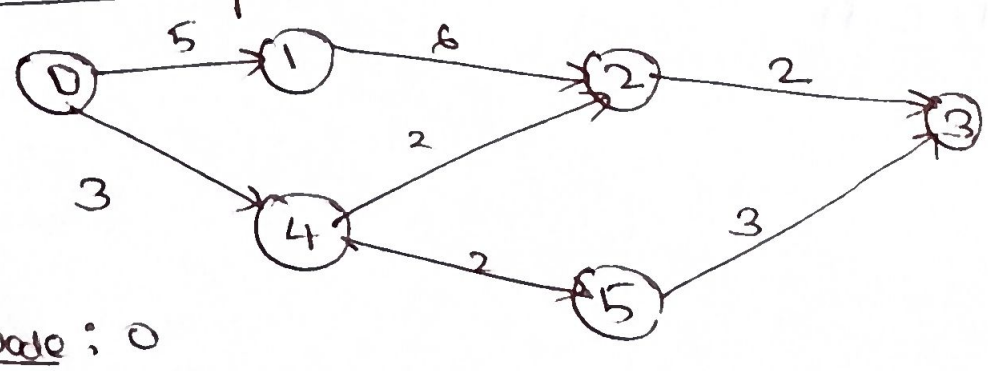
Ex 2



Starting node: 0

Selected Node	0	1	2	3	4	5	6
0	[0]	∞	∞	∞	∞	∞	∞
1		[1]	∞	5	∞	∞	∞
2			[3]	3	∞	∞	∞
3				[3]	5	8	13
4					[4]	8	13
5						[7]	13
							[8]

Ex 3

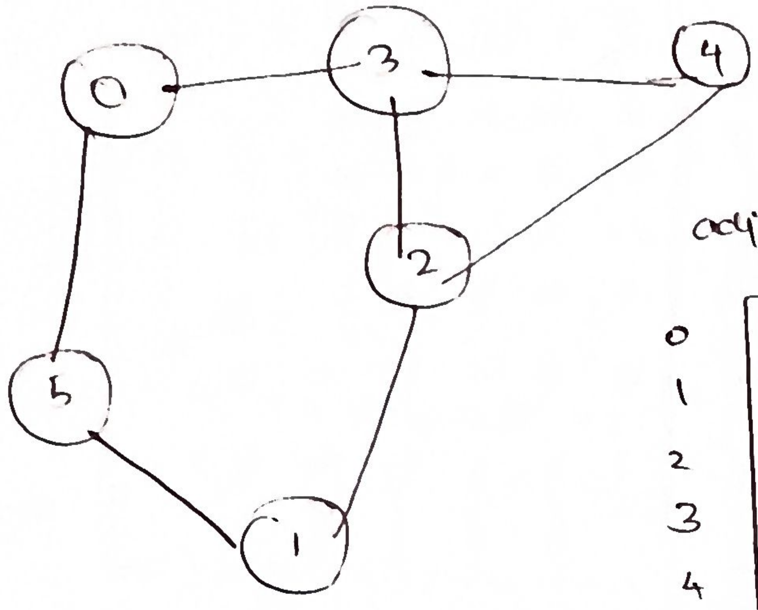


Selected Node : 0

Selected Node :	0	1	2	3	4	5
0	[0]	∞	∞	∞	∞	∞
4		5	∞	∞	[3]	∞
1		[5]	5	∞		5
2			[5]	∞		5
5				7		[5]
3				[7]		

Algorithms

* Depth First Search



adjacency matrix:

	0	1	2	3	4	5
0	0	0	0	1	0	1
1	0	0	1	0	0	1
2	0	1	0	1	1	1
3	1	0	1	0	1	0
4	0	0	1	1	0	0
5	1	1	1	0	0	0

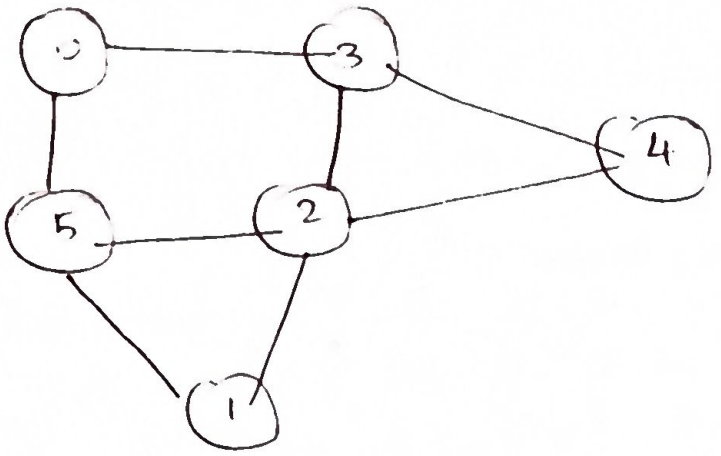
0 → 3 → 2 → 1 → 5

Program

(no. of nodes) (adj matrix) (visited array) (node)

```
void dfs ( int n , int a[ ][n], int * v, int i) {

    int j=0

    for(j=0 ; j<n; j++) {

        if (a[i][j] ==1  && v[j] ==0) {

            dfs( n, a, v, j );

        }
    }
}
```

Breadth First Search

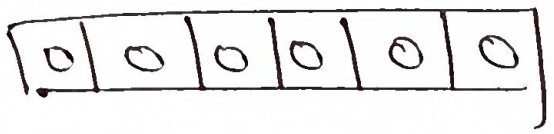


adjacency matrix

	0	1	2	3	4	5
0	0	0	0	1	0	1
1	0	0	1	0	0	1
2	0	1	0	1	1	1
3	1	0	1	0	1	0
4	0	0	1	1	0	0
5	1	1	1	0	0	0

Hand Trace

q 

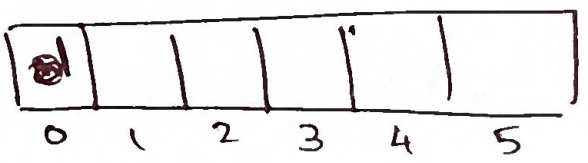
v 

Source node : 0

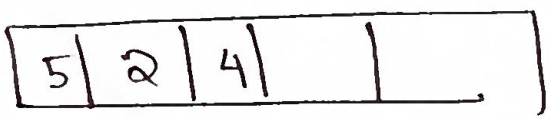
Call 1

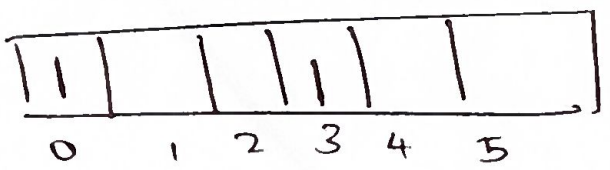
q 

output : 0

v 

Call 2 : 3 is dequeued

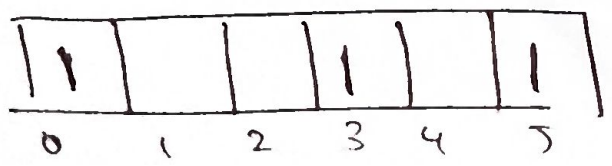
q 

v 

output 0→3

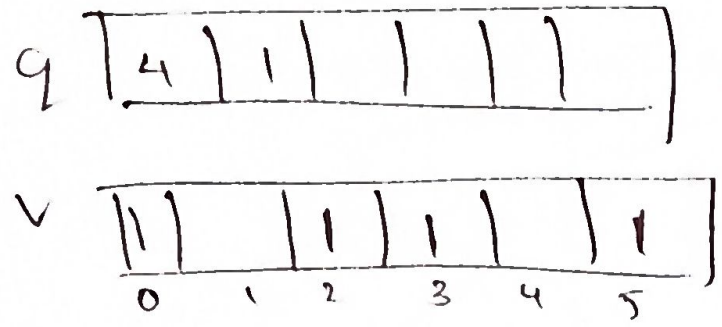
Call 3 : 5 is dequeued

q 

v 

output 0 → 3 → 5

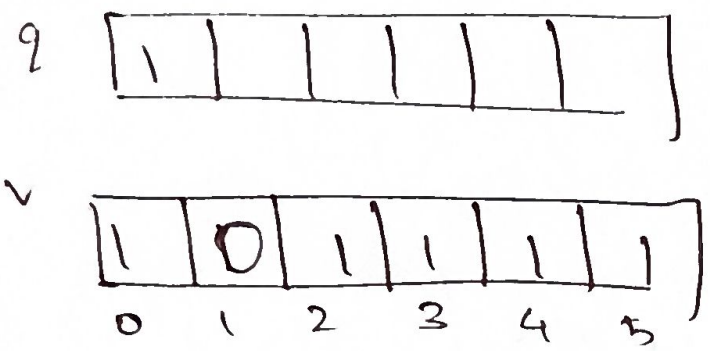
Call 4: 2 is dequed



output:

0 → 3 → 5 → 2

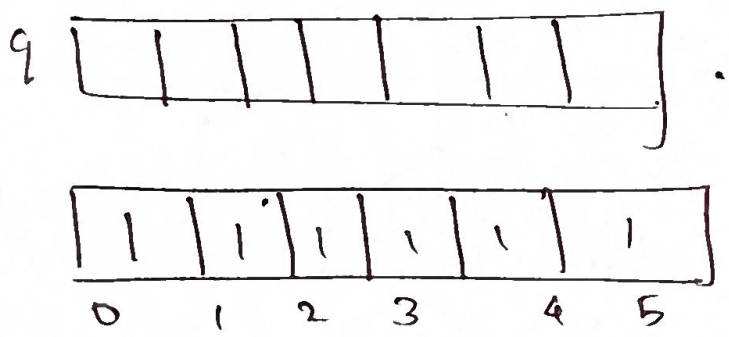
Call 5: 4 is dequeued



output

0 → 3 → 5 → 2 → 4

Call 6: 1 is dequed



Output

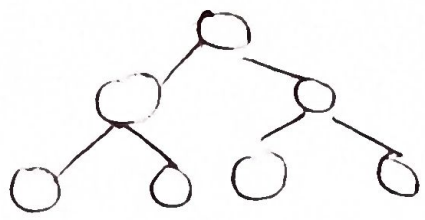
0 → 3 → 5 → 2 → 4 → 1

Program

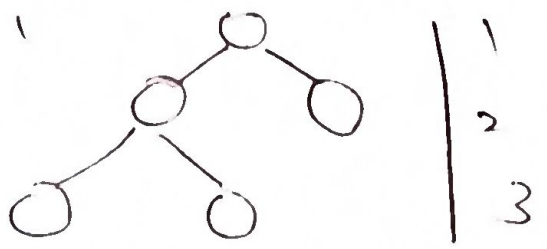
```
void enqueue (int q[], int n, int *front, int *rear, int data)
    if (*rear == n) {
        printf ("Full");
    }
    else {
        q[*rear] = data;
        (*rear)++;
    }
```

Heaps

Full tree : no more nodes can be added



complete tree : full till h-1



binary heap : a complete binary tree

minheap : children > parents

maxheap : parents > children

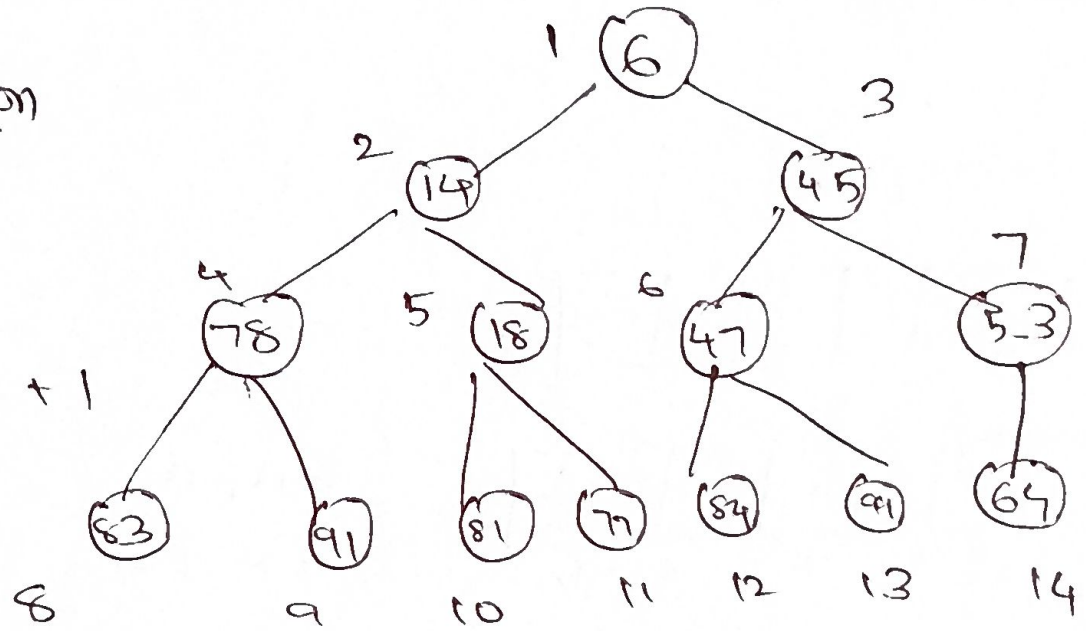
Array Implementation

index : i

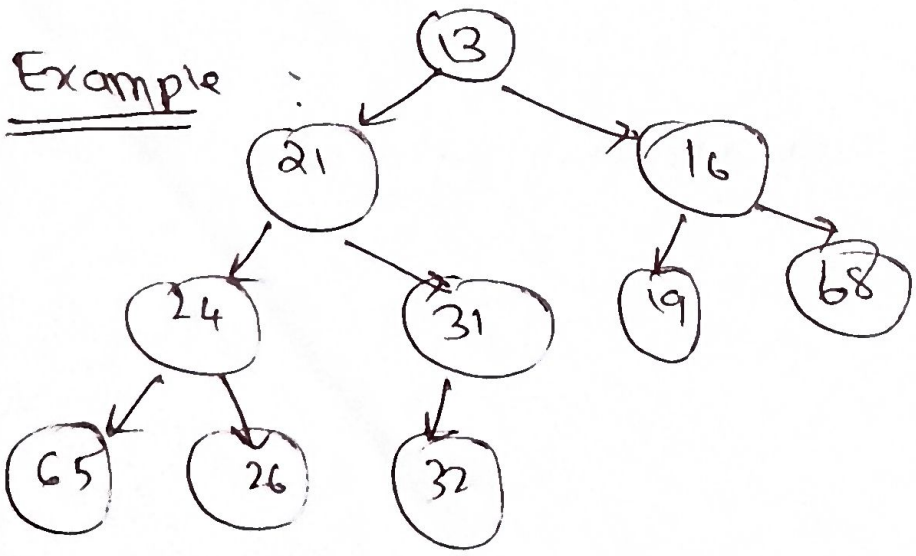
left child : 2*i

right child : 2*i+1

parent : i/2



Example :

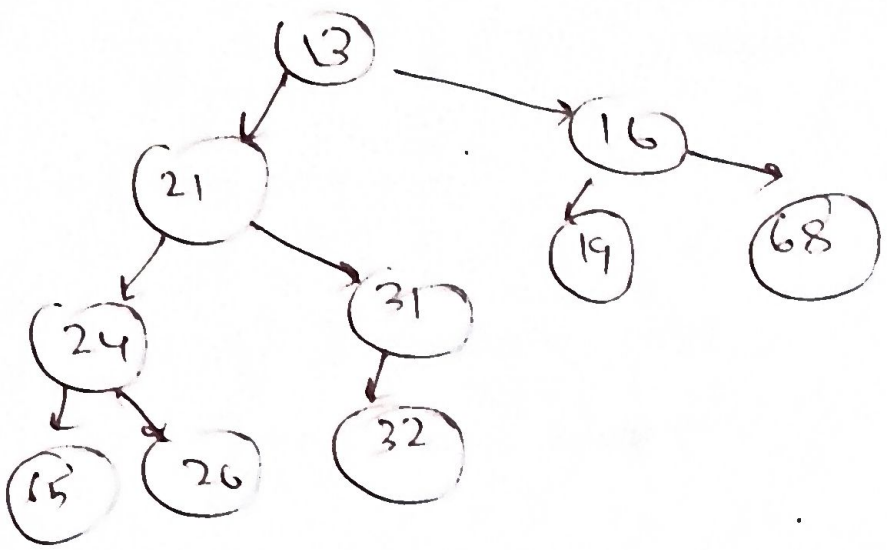


max no. of nodes = $2^h - 1$

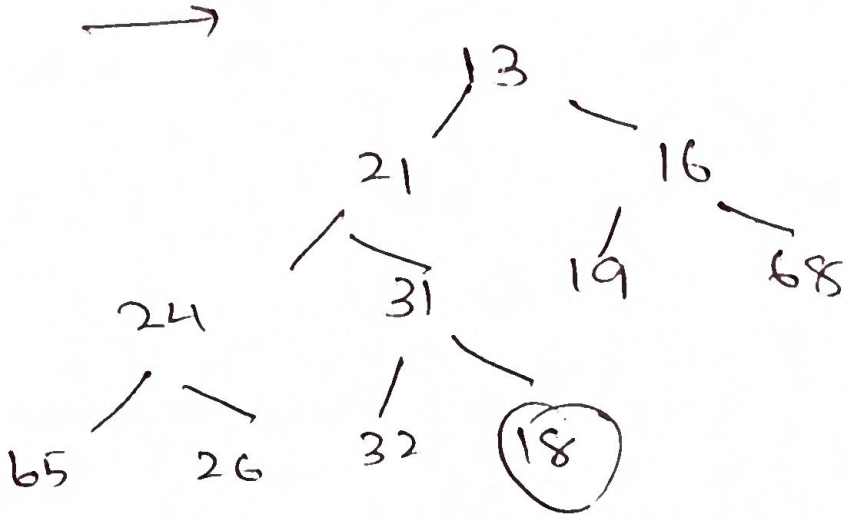
= 13

13	21	16	24	31	19	68	65	26	32	.		
1	2	3	4	5	6	7	8	9	10	11	12	B

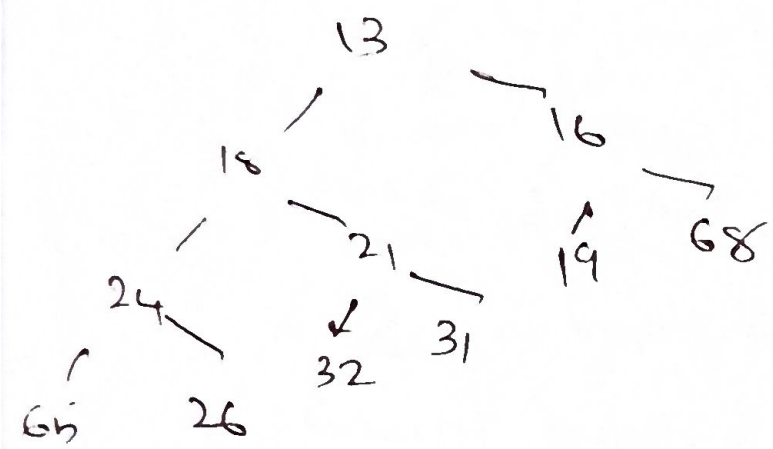
Min Heap Insertion



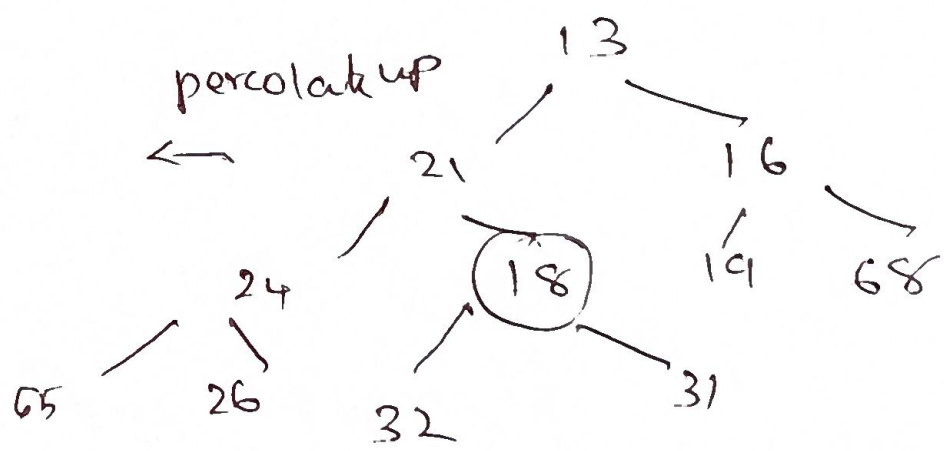
Insert 18



percolate up

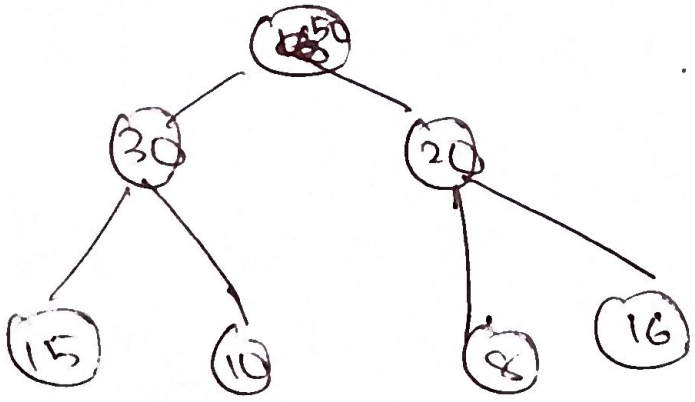


percolate up



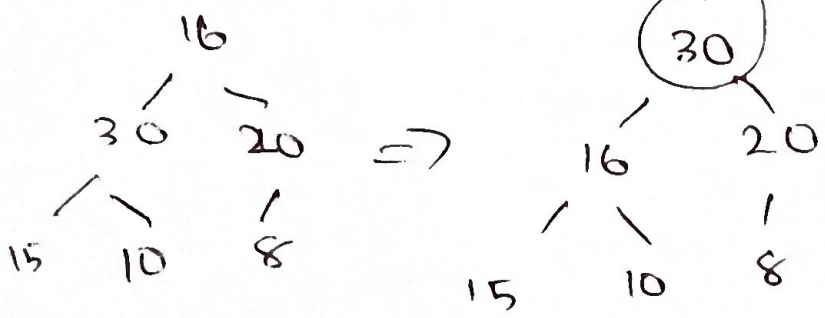
Deletion

(1) in max heap



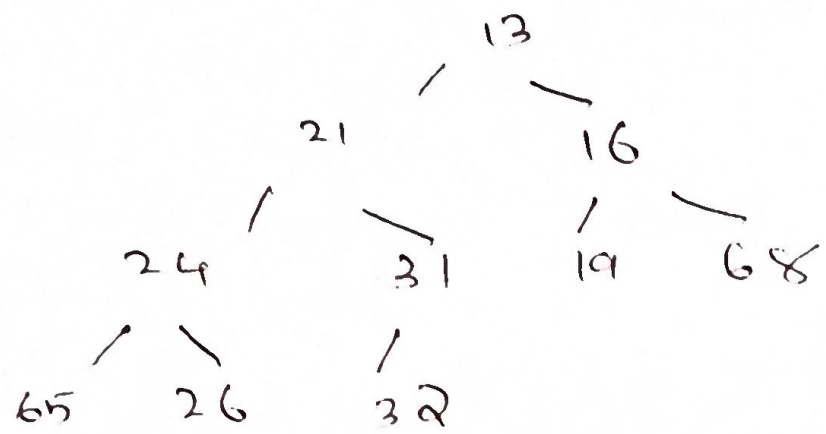
can only delete ~~16~~ 50

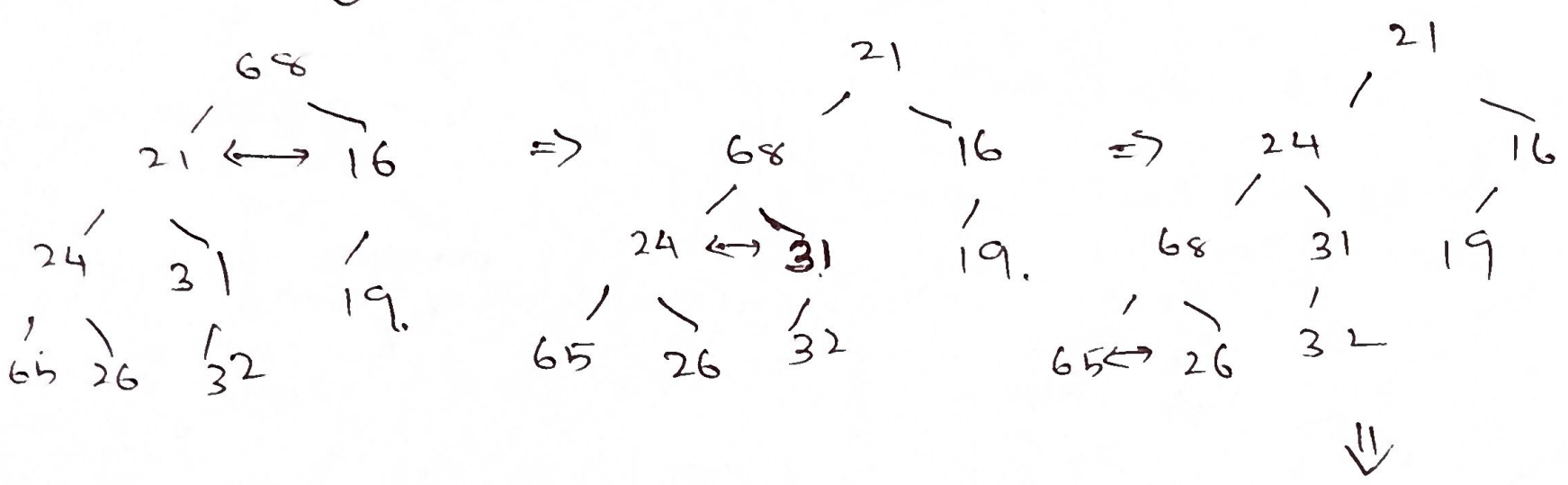
replace w/ last node, i.e 16

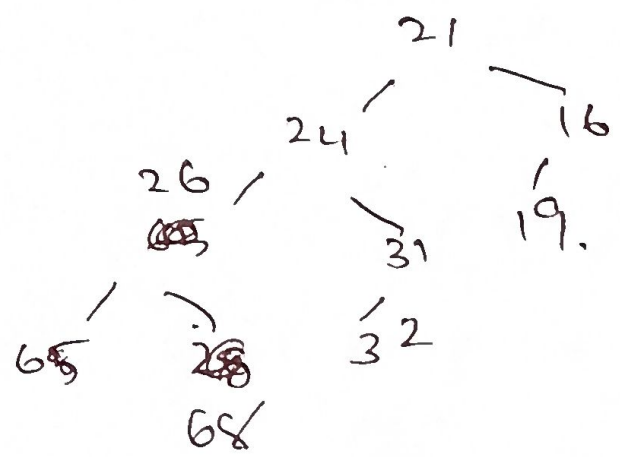


(2) in min heap

replace 13 w/ 68







Trees

Structure

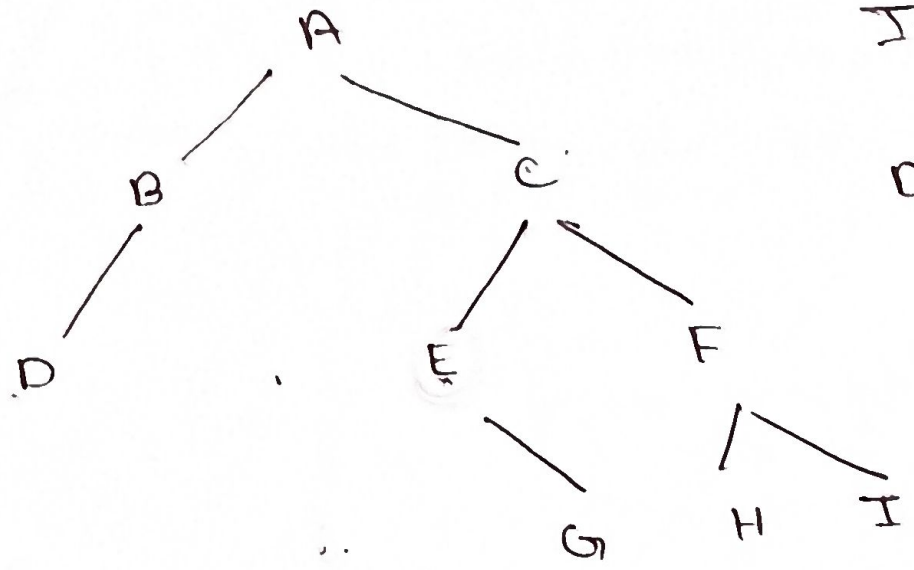
```
struct node {
    int data;
    struct node * left;
    struct node * right;
}

struct node * create_node ( struct node * root , int data) {
    root = (struct node *) malloc (sizeof (struct node));
    root -> data = data
    root -> left = NULL
    root -> right = NULL
    return root;
}

void insertNode ( struct node * root, int data) {
    if (root -> data < root -> data) {
        if ( root -> left == NULL)
            root -> left = createNode (root -> left , data)
        else
            insertNode (root -> left , data)
    }
```

```
else if (data > root -> data ) {

    if (root -> right == NULL )

        root -> right = createNode ( root -> right , data)

    else

        insertNode (root -> right , data)

}

struct node * createBST ( struct node * root, int data ) {

    if (root == NULL)

        createNode ( root , data)

    else

        insertNode (root, data)

    return root

}
```

Traversals

Inorder

```
if (root -> left != NULL)

    inorder (root -> left , data)

print (root -> data )

if (root -> right != NULL )

    inorder (root -> right , data)
```

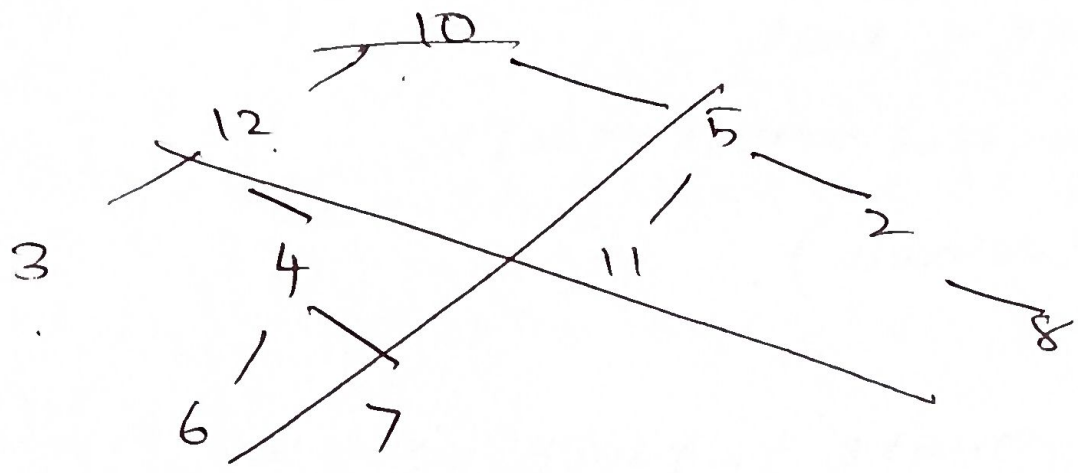


Inorder: LDR

D B A E G C H F I

Preorder DLR

Postorder: LRD



ABDECFG

DEBFGCA

~~Inorder~~:

pred

```
struct node * pred  = root -> left
while (root -> right != NULL)
        pred = pred -> right)
return pred
```

succ

```
struct node* succ = root -> right
while (root -> left != NULL)
        succ = succ -> left
```

Deletion

```
if (data < root -> data)
        root -> left = ~~delNode~~ deletion(root -> left, data)
else if (data > root -> data)
        root -> right = deletion( root -> right, data)
```

```
else }
    if (root → left == NULL && root → right == NULL){
        free(root)
        root = NULL
    }
else if (root → right == NULL && root → left != NULL) {
        delnode = root
        root = root → root → left
        free(delnode)
}
else if (root → right != NULL && root → left != NULL)
        delnode = root
        root = root → right
        free(delnode)
}
else if (root → left != NULL && root → right != NULL)
        pred = inOrderpred (root)
        root → data = pred
        root → left = delete(root → left, root → data)
return root
```

```
int height (shuct node *root )

    int lh = 0
    int rh = 0
    if (root → right == NULL && root → left
                                == NULL) {

        return 1;
    }

else if (root → left != NULL ) {
            lh = height (root → left)
}
if (root→right != NULL) {
            rh = height(root → right )
}
if (lh > rh)
    return 1+lh

else
    return
        1+rh
```