Q1)

```
#include<stdio.h>
#include<stdlib.h>

void maxheapify(int arr[],int i,int size)
{
   int largest=i;
   int child1=2*i+1;
   int child2=2*i+2;

   if(child1<size && arr[largest]<arr[child1])
   largest=child1;

   if(child2<size && arr[largest]<arr[child2])
   largest=child2;

   if(largest!=i)
   {
      int temp=arr[largest];
      arr[largest]=arr[i];
      arr[i]=temp;

      maxheapify(arr,largest,size);
   }
   return;
}

void buildheap(int arr[],int size)
{
   int i;
   for(i=size/2;i>=0;i--)
```

```
    maxheapify(arr,i,size);
    return;
}

int main()
{
   int arr[]={56,78,98,67,33,23,66};
   buildheap(arr,7);
   int i;
   for(i=0;i<7;i++)
   printf("%d\t",arr[i]);

   return 0;

}
```

Q2)

```
#include <stdio.h>
#include <stdlib.h>
#define CAPACITY 10
int i,n=10,top=-1;
int stack[CAPACITY];
void push(int item);
void pop(void);
void display();
int main(){
   int item;
   while (1) {
      printf("\nenter the operation you have to perform:\n");
      printf("1.push\n");
      printf("2.pop\n");
      printf("3.display\n4.exit\n");
```

```
        scanf("%d",&i);
        switch (i) {
           case 1:
              printf("enter the element you want to insert:");
              scanf("%d",&item);
              push(item);
              break;
           case 2:
              pop();
              break;
           case 3:

              display();
              break;
           case 4:
              exit(0);
        }
     }

}

void push(int item){
    if (top==CAPACITY-1) {
        printf("the stack is full");
        }
    else{
        top=top+1;
        stack[top]=item;
        printf("\n%d insereted succesfullY!\n",item);

    }
```

```
}

void pop(){
   if (top<0) {
      printf("\nstack is empty\n");
   }
   else{
      printf("\nthe poped item is %d", stack[top]);
      top--;
   }
}

void display()
{
   int i=0;
   printf("\ncontent of the stack are:\n");
   while (i<=top) {
      printf("%d\t",stack[i]);
      i++;
   }



}
```

Q4)

```
#include <stdio.h>
#include <stdlib.h>
#define CAPACITY 3
int i,n=10,top=-1;
int stack[CAPACITY];
void push(int item);
```

```
void pop(){
   if (top<0) {
      printf("\nstack is empty\n");
   }
   else{
      printf("the poped item is %d", stack[top]);
      top--;
   }
}
void isempty()
{
   if(top==-1)
   printf("Stack is empty!");
   else
   printf("Stack is not empty!");
}

void isfull()
{
   if(top==CAPACITY-1)
   {
     printf("Stack is full!");
   }
   else
   {
      printf("Stack is not full!");
   }
}
int main(){
   int item;
   while (1) {
```

```
    printf("\nenter the operation you have to perform:\n");
    printf("1.push\n2-empty\n3-full\n4-pop\n5-exit\n");


    scanf("%d",&i);
    switch (i) {
       case 1:
          printf("enter the element you want to insert:");
          scanf("%d",&item);
          push(item);
          break;
       case 2:
          isempty();
          break;

       case 3:

         isfull();
          break;
       case 4:pop();
       break;

       case 5:

          exit(0);
          break;
       default:
       printf("Invalid choice!");
    }
}
```

```
}

void push(int item){
   if (top==CAPACITY-1) {
      printf("the stack is full");
      }
   else{
      top=top+1;
      stack[top]=item;
      printf("\n%d insereted succesfullY!\n",item);

   }
}
```

Q5) &6)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
        int data;
        struct node* left;
        struct node* right;
};

struct node* newNode(int data)
{
        struct node* node
                = (struct node*)malloc(sizeof(struct node));
        node->data = data;
        node->left = NULL;
```

```
        node->right = NULL;

        return (node);
}


void Postorder(struct node* node)
{
        if (node == NULL)
                return;


   Postorder(node->left);
   Postorder(node->right);
        printf("%d ", node->data);
}

void Inorder(struct node* node)
{
        if (node == NULL)
                return;

        Inorder(node->left);
        printf("%d ", node->data);
        Inorder(node->right);
}

void Preorder(struct node* node)
{
        if (node == NULL)
                return;
```

```
        printf("%d ", node->data);
        Preorder(node->left);
        Preorder(node->right);
}

int main()
{
        struct node* root = newNode(76);
        root->left = newNode(45);
        root->left->left = newNode(34);
        root->left->left->left = newNode(22);
        root->right = newNode(77);
        root->right->right=newNode(88);
        root->right->right->left=newNode(81);

        int ch;
        while(1)
        {
           printf("\n1-preorder traversal\n2-inorder traversal\n3-postorder traversal\n4-exit\n");
           printf("Enter your choice-");
           scanf("%d",&ch);
           switch(ch)
           {
              case 1:printf("\nPreorder traversal of tree\n");
              Preorder(root);
              break;

              case 2:printf("\nInorder traversal of tree\n");
              Inorder(root);
              break;
```

```
                case 3:printf("\nPostorder traversal of tree\n");
                Postorder(root);
                break;

                case 4:exit(0);
                break;

                default:
                printf("Invalid choice");
              }
           }
}
```

Q7)

```
#include<stdio.h>
#include<string.h>
#define CAPACITY 10
int top,stack[CAPACITY];

void push(char c)
{
 if(top==CAPACITY-1){
  printf("Stack overflow!");
 }
 else
 {
    stack[++top]=c;
 }
}
```

```
void pop()
{

   printf("%c",stack[top--]);
}

int main()
{
   char str[10]="RSCOE";
   int len=strlen(str);
   int i;
   printf("In normal order-RSCOE\n");
   printf("In reverse orde-");

   for(i=0;i<len;i++)
   {
   push(str[i]);
   }

   for(i=0;i<len;i++)
   {
      pop();
   }
}
```

Q9)

```
#include<stdio.h>
int binarySearch(char *c, char letter) {
  int lo, mid, hi;
  lo = 0;
```

```
  hi =3;
  while (lo <= hi) {
    mid = lo + (hi - lo) / 2;
    if (c[mid] == letter) {
      return mid;
    } else if (c[mid] > letter) {
      hi = mid-1;
    } else {
      lo = mid+1;
    }
  }
  return -1;
}

int main()
{
   char letter;
   int index,i;
   char c[6]={'A','B','C','D','E','F'};
   for(i=0;i<6;i++)
   {
      printf("%c\t",c[i]);
   }
   printf("\nEnter the element that you want to search in the array-");
   scanf("%c",&letter);
   printf("%c found at index %d",letter,index=binarySearch(c,letter));
}
```

Q10)

```
#include <stdio.h>
```

```
int search(int array[], int n, int x) {

  for (int i = 0; i < n; i++)
    if (array[i] == x)
      return i;
  return -1;
}

int main() {
  int array[] = {10,23,40,1,2,0,14,13,50,9};
  int x;

  printf("Enter the element that you want to search-");
  scanf("%d",&x);
  int n = sizeof(array) / sizeof(array[0]);

  int result = search(array, n, x);

  (result == -1) ? printf("Element not found") : printf("Element found at index: %d", result);
}
```

Q13)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *link;
};
struct node *root=NULL;
```

```
void append()
{
   struct node *temp;
   temp=(struct node*)malloc(sizeof(struct node));
   printf("Enter the element that you want to append=");
   scanf("%d",&temp->data);
   temp->link=NULL;
   if(root==NULL)
   {
      root=temp;
   }
   else
   {
      struct node *p;
      p=root;
      while(p->link!=NULL)
      {
         p=p->link;
      }
      p->link=temp;
   }
   printf("%d inserted successfully!",temp->data);
}

void display()
{
   struct node *p;
   p=root;
   if(p==NULL)
   {
      printf("No element in the linked list to display");
```

```
    }
    else
    {
        while(p!=NULL)
        {
            printf("%d\t",p->data);
            p=p->link;
        }
    }
}


void addbeg()
{
    struct node *temp=NULL;
    temp=(struct node*)malloc(sizeof(struct node));
    printf("Enter the data to add at begin");
    scanf("%d",&temp->data);
    temp->link=NULL;
    if(root==NULL)
    {
        root=temp;
    }
    else
    {
        temp->link=root;
        root=temp;
    }
    printf("%d inserted successfully!",temp->data);
}
```

```
int main()
{
  int ch=0;
  root=(struct node*)malloc(sizeof(struct node));
   printf("Enter the data in the root node that is in the root node=");
   scanf("%d",&root->data);
   root->link=NULL;
   printf("The first element of the linked list is %d",root->data);

   struct node *current=NULL;
   current=(struct node*)malloc(sizeof(struct node));
   printf("\nEnter the data in the 2nd node=");
   scanf("%d",&current->data);
   current->link=NULL;
   root->link=current;
   printf("\nThe second element of the linked list is %d",current->data);

    struct node *current1=NULL;
   current1=(struct node*)malloc(sizeof(struct node));
   printf("Enter the data in the 3rd node=");
   scanf("%d",&current1->data);
   current1->link=NULL;
   current->link=current1;
   printf("The third element of the linked list is %d\n",current->data);

 while(1){

     printf("\nEnter 1-append\n2-adding at the beginning\n\n3-display\n4-to exit\n");
    printf("Enter your choice=");
    scanf("%d",&ch);
       switch(ch)
```

```
      {
         case 1:
         append();
         break;

         case 2:
         addbeg();
         break;

         case 3:
         display();
         break;

         case 4:
         exit(0);
         break;

         default:
         printf("Invalid choice!");

      }
   }

}
```

Q14/Q15)

```
#include<stdio.h>
#include<stdlib.h>
int insertSorted(int arr[], int n,  int key, int capacity)
{
```

```
if (n >= capacity)
return n;
arr[n] = key;
return (n + 1);
}



int main()
{
int arr[20] = {8, 5, 6, 9, 0, 7} ;
int capacity = sizeof(arr) / sizeof(arr[0]);
int n = 6;
int i, key,ch;
int position;
printf("\n displaying the array- ");
for (i = 0; i < n; i++)
printf("%d\t", arr[i]);

while(1)
{
   printf("\n1-insert\n2-Delete\n3-display\n4-exit");
   printf("\nEnter your choice-");
   scanf("%d",&ch);
   switch(ch)
   {
      case 1:
      printf("\nEnter the element that you want to insert-");
      scanf("%d",&key);
       n = insertSorted(arr, n, key, capacity);
       printf("\n After Insertion: ");
```

```
 for (i = 0; i < n; i++)
 printf("%d  ",arr[i]);
 break;

 case 2:
 printf("Enter the location where you wish to delete element\n");
  scanf("%d", &position);

   if (position >= n+1)
  printf("Deletion not possible.\n");
 else
{
for (i = position - 1; i < n - 1; i++)
  arr[i] = arr[i+1];

printf("Resultant array:\n");

for (i = 0; i < n - 1; i++)
  printf("%d\t", arr[i]);

 break;
 }
 case 3:
 printf("Displaying the array-");
 for (i = 0; i < n; i++)
 printf("%d\t", arr[i]);
 break;

 case 4:exit(0);
 break;
 default:
```

```
      printf("Invalid code");
    }




}
return 0;
}
```

Q20)

```
#include<stdio.h>



int q[20],top=-1,front=-1,rear=-1,a[20][20],visited[20],stack[20];
int dequeue();
void enqueue(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();

int main()
{ int n,i,s,j;

  printf("\nEnter the Number of Nodes in Graph: ");
  scanf("%d",&n);
  printf("\n Enter the Adjecency Matrix is: ");
  for(i=1;i<=n;i++)
   { for(j=1;j<=n;j++)
     { scanf(" %d",&a[i][j]);}
     printf("\n");
   }
```

```
   for(i=1;i<=n;i++)
     visited[i]=0;


printf("\n1) B.F.S (Breadth First Search)");

printf("\nEnter source VERTEX:");
scanf("%d",&s);

bfs(s,n);
}

void bfs(int s,int n)
{
 int p,i;
enqueue(s);
 visited[s]=1;
 p=dequeue();
 if(p!=0)
 printf(" %d",p);
 while(p!=0)
 {
  for(i=1;i<=n;i++)
    if((a[p][i]!=0)&&(visited[i]==0))
      { enqueue(i);
        visited[i]=1;
      }
   p=dequeue();
   if(p!=0)
    printf(" %d ",p);
 }
```

```
 for(i=1;i<=n;i++)
    if(visited[i]==0)
       bfs(i,n);
}
void enqueue(int item)
{
   if(rear== -1)
   {
     q[++rear]=item;
     front++;
    }
   else
     q[++rear]=item;
}

int dequeue()
{ int k;
if((front>rear)||(front==-1))
   return(0);
else
{
k=q[front++];
return(k);
}
}


void push(int item)
{ if(top== 19)
printf("\n\nStackoverflow.com");
else
```

```
stack[++top]=item;
}

int pop()
{ int k;
if(top==-1)
return(0);
else
{ k=stack[top--];
return(k);
}
}
```

Q19)

```
#include<stdio.h>
struct student
{
   int marks;
};
int main()

{
int j, i, temp=0;
struct student a[5];
printf("enter the marks=");

for (i=0;i<=4;i++)
{
scanf("%d", &a[i].marks);
}
for (i=0;i<=4; i++)
{
```

```
for (j=i+1; j <= 4; j++)
{
if (a [j] .marks > a [i]. marks)
{
temp = a[j]. marks;

a[j].marks = a[i]. marks, a[i].marks = temp;

} }}

printf(" displaying sorted marks \n");

for (i=0; i<=4; i++)
{
 printf("%d\t", a[i]. marks);
}
printf ("\nIn topper of the   topper of the class is %d", a[0].marks);

}
```

Q21)

```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],visited[20],stack[20];
int dequeue();
void enqueue(int item);
void dfs(int s,int n);
void push(int item);
int pop();

int main()
```

```
{ int n,i,s,ch,j;

  printf("\nEnter the Number of Nodes in Graph: ");
  scanf("%d",&n);
  printf("\n Enter the Adjecency Matrix is: ");
  for(i=1;i<=n;i++)
   { for(j=1;j<=n;j++)
     {
        scanf(" %d",&a[i][j]);

     }
     printf("\n");
   }

    for(i=1;i<=n;i++)
       visited[i]=0;


printf("\n D.F.S (Depth First Search)");
printf("\nEnter source VERTEX:");
scanf("%d",&s);
       dfs(s,n);

return 0;
}


int dequeue()
{ int k;
if((front>rear)||(front==-1))
   return(0);
```

```
else
{
k=q[front++];
return(k);
}
}

void dfs(int s,int n)
{ int i,k;
  push(s);
  visited[s]=1;
  k=pop();
  if(k!=0)
    printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(visited[i]==0))
{
  push(i);
  visited[i]=1; }
  k=pop();
  if(k!=0)
  printf(" %d ",k);
}
for(i=1;i<=n;i++)
  if(visited[i]==0)
  dfs(i,n);
}

void push(int item)
```

```
{ if(top== 19)
printf("\n\nStackoverflow.com");
else
stack[++top]=item;
}

int pop()
{ int k;
if(top==-1)
return(0);
else
{ k=stack[top--];
return(k);
}
}
```

Q23

```
#include <stdio.h>

void insert(int a[], int n)
{
   int i, j, temp;
   for (i = 1; i < n; i++) {
      temp = a[i];
      j = i - 1;

      while(j>=0 && temp <= a[j])
      {
         a[j+1] = a[j];
         j = j-1;
      }
      a[j+1] = temp;
```

```
    }
}

void printArr(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{
    int a[] = { 12, 45, 76, 42, 66, 27 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are - \n");
    printArr(a, n);
    insert(a, n);
    printf("\nAfter sorting array elements are - \n");
    printArr(a, n);

    return 0;
}
```

Q24)

```
#include<stdio.h>

#include<stdlib.h>
int partition (int a[], int low, int high)

{int pivot= a[low];
```

```
int i = low;

int j=high;
while (i<j)
{
while (a[i]<=pivot)

{ i++;}
while(a[j]>pivot)
{ j--;}
if (i < j)
{
int temp=a [i];

a [i] = a[j] ;

a[j]=temp;}}



int temp=a[low];

a [low] = a[j] ;

a [j] = temp;

return j; }
void quicksort (int a[], int low, int high)
{ int p;

if (low<high) {
```

```
p=partition (a, low, high); quicksort (a, low, p-1);
quicksort (a, p + 1 , high);
}}

int main()
{
int i;

int a[6] = {12, 45, 76, 42, 66, 27} ;
 quicksort (a, 0, 6) ;

printf("sorted elements by Quick sort\n");
for (i = 0; i < 6;i++)
{
printf(" \%d " , a[i]) ;
}
}
```

Q21)

```
#include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],visited[20],stack[20];
int dequeue();
void enqueue(int item);
void bfs(int s,int n);

void push(int item);
int pop();
```

```
int main()
{ int n,i,s,j;


  printf("\nEnter the Number of Nodes in Graph: ");
  scanf("%d",&n);
  printf("\n Enter the Adjecency Matrix is: ");
  for(i=1;i<=n;i++)
   { for(j=1;j<=n;j++)
     { scanf(" %d",&a[i][j]);}
     printf("\n");
   }

    for(i=1;i<=n;i++)
       visited[i]=0;


printf("\n B.F.S (Breadth First Search)");
printf("\nEnter source VERTEX:");
scanf("%d",&s);
bfs(s,n);

return 0;
}

void bfs(int s,int n)
{
 int p,i;
enqueue(s);
 visited[s]=1;
```

```
 p=dequeue();
 if(p!=0)
 printf(" %d",p);
 while(p!=0)
 {
  for(i=1;i<=n;i++)
     if((a[p][i]!=0)&&(visited[i]==0))
       { enqueue(i);
         visited[i]=1;
       }
    p=dequeue();
    if(p!=0)
      printf(" %d ",p);
 }
 for(i=1;i<=n;i++)
    if(visited[i]==0)
       bfs(i,n);
}
void enqueue(int item)
{
   if(rear== -1)
   {
     q[++rear]=item;
     front++;
    }
   else
     q[++rear]=item;
}

int dequeue()
{ int k;
```

```
if((front>rear)||(front==-1))
   return(0);
else
{
k=q[front++];
return(k);
}
}

void push(int item)
{ if(top== 19)
printf("\n\nStackoverflow.com");
else
stack[++top]=item;
}

int pop()
{ int k;
if(top==-1)
return(0);
else
{ k=stack[top--];
return(k);
}}
```

Q17)

```
#include <stdio.h>
#include<stdlib.h>


void hanoi (int n, char rodFrom, char rodMid, char rodTo)
```

```
{if (n == 1)

{printf("\nDisk 1moved from %c to %c\n ", rodFrom, rodTo) ;
return ;}


hanoi (n-1, rodFrom, rodTo, rodMid);
 printf ("\nDisk %d moved from %c  to %c\n" , n, rodFrom, rodTo) ;

hanoi(n-1, rodMid, rodFrom, rodTo) ;}

int main()

{hanoi (3, 'A', 'B', 'C');

return 0;}
```

Q3)

```
#include<stdio.h>
#include<stdlib.h>
#define CAPACITY 5
int front=0;
int rear=0;
int queue[CAPACITY];
void enqueue(int ele)
{
   if(CAPACITY==rear)
   {
      printf("Queue is full!");
   }
```

```
    else
    {
        queue[rear]=ele;
        rear++;
        printf("%d inserted successfully!",ele);
    }
}

void dequeue()
{
    if(front==rear)
    {
        printf("Queue is empty!");
    }
    else
    {
        printf("%d deleted",queue[front]);
    }
    for(int i=0;i<rear-1;i++)
    {
        queue[i]=queue[i+1];
    }
    rear--;
}

int main()
{
    int ele,ch;
    while(1)
    {
         printf("\n1=enqueue\n2=dequeue\n3=exit\n");
```

```
        printf("Enter your choice-");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:
           {
            printf("Enter the element you want to add in queue=");
           scanf("%d",&ele);
           enqueue(ele);
           break;
           }
           case 2:
           dequeue();
           break;

           case 3:
           exit(0);
           break;

           default:
           printf("Invalid choice!");
        }
     }
}
```

Q18)

```
/* program for addition of two polynomials
 * polynomial are stored using structure
 * and program uses array of structure
 */
#include<stdio.h>
```

```
/* declare structure for polynomial */
struct poly
{
        int coeff;
        int expo;
};
/* declare three arrays p1, p2, p3 of type structure poly.
* each polynomial can have maximum of ten terms
* addition result of p1 and p2 is stored in p3 */

struct poly p1[10],p2[10],p3[10];

/* function prototypes */
int readPoly(struct poly []);
int addPoly(struct poly [],struct poly [],int ,int ,struct poly []);
void displayPoly( struct poly [],int terms);

int main()
{
        int t1,t2,t3;

        /* read and display first polynomial */
        t1=readPoly(p1);
        printf(" \n First polynomial : ");
        displayPoly(p1,t1);
        /* read and display second polynomial */
        t2=readPoly(p2);
        printf(" \n Second polynomial : ");
        displayPoly(p2,t2);
```

```
	/* add two polynomials and display resultant polynomial */
	t3=addPoly(p1,p2,t1,t2,p3);
	printf(" \n\n Resultant polynomial after addition : ");
	displayPoly(p3,t3);
	printf("\n");

	return 0;
}

int readPoly(struct poly p[10])
{
	int t1,i;

	printf("\n\n Enter the total number of terms in the polynomial:");
	scanf("%d",&t1);

	printf("\n Enter the COEFFICIENT and EXPONENT in DESCENDING ORDER\n");
	for(i=0;i<t1;i++)
	{
		printf("   Enter the Coefficient(%d): ",i+1);
		scanf("%d",&p[i].coeff);
		printf("      Enter the exponent(%d): ",i+1);
		scanf("%d",&p[i].expo);       /* only statement in loop */
	}
	return(t1);
}

int addPoly(struct poly p1[10],struct poly p2[10],int t1,int t2,struct poly p3[10])
{
	int i,j,k;
```

```
i=0;
j=0;
k=0;

while(i<t1 && j<t2)
{
	if(p1[i].expo==p2[j].expo)
	{
		p3[k].coeff=p1[i].coeff + p2[j].coeff;
		p3[k].expo=p1[i].expo;

		i++;
		j++;
		k++;
	}
	else if(p1[i].expo>p2[j].expo)
	{
		p3[k].coeff=p1[i].coeff;
		p3[k].expo=p1[i].expo;
		i++;
		k++;
	}
	else
	{
		p3[k].coeff=p2[j].coeff;
		p3[k].expo=p2[j].expo;
		j++;
		k++;
	}
}
```

```
        /* for rest over terms of polynomial 1 */
        while(i<t1)
        {
                p3[k].coeff=p1[i].coeff;
                p3[k].expo=p1[i].expo;
                i++;
                k++;
        }
        /* for rest over terms of polynomial 2 */
        while(j<t2)
        {
                p3[k].coeff=p2[j].coeff;
                p3[k].expo=p2[j].expo;
                j++;
                k++;
        }

        return(k); /* k is number of terms in resultant polynomial*/
}

void displayPoly(struct poly p[10],int term)
{
        int k;

        for(k=0;k<term-1;k++)
        printf("%d(x^%d)+",p[k].coeff,p[k].expo);
        printf("%d(x^%d)",p[term-1].coeff,p[term-1].expo);
}
```