Week 1

Binary search

```c
#include <stdio.h>

int binarySearch(int array[], int x, int low, int high) {
 if (high >= low) {
   int mid = low + (high - low) / 2;

   // If found at mid, then return it
   if (array[mid] == x)
     return mid;

   // Search the left half
   if (array[mid] > x)
     return binarySearch(array, x, low, mid - 1);

   // Search the right half
   return binarySearch(array, x, mid + 1, high);
 }
 return -1;
}

int main()
{
 int array[100];
 int n,x;
 printf("enter no of elements in the array :");
 scanf("%d",&n);
  printf("enter elements in the array    \n");
 for (int i = 0; i < n; i++)
 {
   scanf("%d",&array[i]);
 }
  printf("enter elements to search  :");
 scanf("%d",&x);
```

```
 int result = binarySearch(array, x, 0, n - 1);
 if (result == -1)
   printf("Not found");
 else
   printf("Element is found at index %d", result);
}
```

Merge sort with insertion sort

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

void insertionSort (int arr[], int n)
{
 int i, key, j;
 for (i = 0; i <= n; i++)
   {
     key = arr[i];
     j = i - 1;

     while (j >= 0 && arr[j] > key)
 {
   arr[j + 1] = arr[j];
   j = j - 1;
 }
     arr[j + 1] = key;
   }
}

// function to sort the subsection a[i .. j] of the array a[]
void merge_sort (int i, int j, int a[], int temp[])
{
 if (j - i <= 10)
   {
     insertionSort (a, j);
     return;     // the subsection is empty or a single element
   }
 int mid = (i + j) / 2;
```

```
 merge_sort (i, mid, a, temp); // sort the left sub-array recursively
 merge_sort (mid + 1, j, a, temp); // sort the right sub-array recursively

 int left = i;     // left points to the beginning of the left sub-array
 int right = mid + 1;    // right points to the beginning of the right
sub-array
 int k;      // k is the loop counter

 for (k = i; k <= j; k++)
   {
     if (left == mid + 1)
 {     // left pointer has reached the limit
   temp[k] = a[right];
   right++;
 }
     else if (right == j + 1)
 {     // right pointer has reached the limit
   temp[k] = a[left];
   left++;
 }
     else if (a[left] < a[right])
 {     // pointer left points to smaller element
   temp[k] = a[left];
   left++;
 }
     else
 {     // pointer right points to smaller element
   temp[k] = a[right];
   right++;
 }
   }

 for (k = i; k <= j; k++)
   {       // copy the elements from temp[] to a[]
     a[k] = temp[k];
   }
}

int main ()
```

```
{
 int a[10000], temp[10000], n, i;

 printf ("Enter number of elements :\n");
 scanf ("%d", &n);

 printf ("Entering %d integers \n", n);

 for (i = 0; i < n; i++)
   a[i] = rand () % 1000;

 merge_sort (0, n - 1, a, temp);

 printf ("Printing the sorted array:\n");

 for (i = 0; i < n; i++)
   printf ("%d\n", a[i]);

 return 0;
}
```

WEEK 2

Quicksort normal

```
#include<stdio.h>
int partition(int arr[30],int low,int high)
{
 int pivot,i,j,temp;
 pivot=arr[low];
 i=low+1;
 j=high;
   while(i<j)
   {
     while(i<=high && pivot>arr[i])
     {
       i++;
     }
```

```
      while(pivot<arr[j])
      {
        j--;
      }
      if(i<j)
      {
        temp=arr[i];
        arr[i]=arr[j];
        arr[j]=temp;
      }
      else
      {
          temp=arr[j];
          arr[j]=arr[low];
          arr[low]=temp;
            return j;
      }
    }
}

int quicksort(int arr[20],int low,int high)
{
 if(low<high)
 {
   int p=partition(arr,low,high);
     quicksort(arr,low,p-1);
     quicksort(arr,p+1,high);
 }
}
int main()
{
int n;
 printf("Enter the no of elements ");
 scanf("%d",&n);
int arr[30];
 printf("Enter the values to be sorted \n");
 for(int i=0;i<n;i++)
   scanf("%d",&arr[i]);
  quicksort(arr,0,n-1);
       printf("values after  sorting \n");
```

```
//values after sorting
 for(int i=0;i<n;i++)
   printf("%d\n",arr[i]);
}
```

Quicksort and its modification

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

int  count= -1;
int arr[100];

void insertionSort(int n)
{
   int i, key, j;
   for (i = 0; i <= n; i++)
   {
       key = arr[i];
       j = i - 1;
       while (j >= 0 && arr[j] > key)
       {
           arr[j + 1] = arr[j];
           j = j - 1;
       }
       arr[j + 1] = key;
   }
}
int middleOfThree(int a, int b, int c)
{
   if ((arr[a] < arr[b] && arr[b] < arr[c]) || (arr[c] < arr[b] && arr[b]
< arr[a]))
      return b;
   else if ((arr[b] < arr[a] && arr[a] < arr[c]) || (arr[c] < arr[a] &&
arr[a] < arr[b]))
      return a;
   else
      return c;
}
```

```
int partition(int low,int high)
{
 int pivot,i,j,temp,mid;
 mid = middleOfThree(low,high,(low+high)/2);
 pivot=arr[low];
 i=low+1;
 j=high;
 while(low<high)
 {
   while(pivot>arr[i])
   {
     i++;
   }
   while(pivot<arr[j])
   {
     j--;
   }
   if(i<j)
   {
     temp=arr[i];
     arr[i]=arr[j];
     arr[j]=temp;
   }
   else
   {
     temp=arr[j];
     arr[j]=arr[low];
     arr[low]=temp;
     return j;
   }
 }
}

int quicksort(int low,int high)
{
 count++;
 if(low<high)
 {
   if((high-low)>=10)
   {
```

```
      int p=partition(low,high);
      if(((p-1)-low)<(high-(p+1)))
        quicksort(low,p-1);
      else
       quicksort(p+1,high);
    }
    else
     insertionSort(high);
 }
}

int main()
{
int n;
 printf("Enter the no of elements ");
 scanf("%d",&n);
 printf("Enter the values to be sorted \n");
 for(int i=0;i<n;i++)
  {
    arr[i] = rand()%40;
    printf("%d ",arr[i]);
  }
 quicksort(0,n-1);
 printf("\nvalues after  sorting \n");
//values after sorting
 for(int i=0;i<n;i++)
   printf("%d\n",arr[i]);

 printf("no of recusive calls are  are \n");
 printf("%d\n",count);
}
```

Week 3

1)articulation point

```
#include<stdio.h>
int df[20],l[20],adj[20][20]={0};
int num=1,vertex,start=1;
```

```
int min(int a,int b)
{
   if(a>b)
        return b;
   else
        return a;
}
void art(int u,int v)
{
 int w;
 df[u]=l[u]=num;
 num++;
   for(w=1;w<=vertex;w++)
   {
        if(adj[u][w]==1)
        {
            if(df[w]==0)
            {
                art(w,u);
                if(u!=start &&l[w]>=df[u])
                    printf("\n%d is articulation point\n",u);
                l[u]=min(l[u],l[w]);
            }
            else if (w!=v)
                l[u]=min(l[u],df[w]);
        }
   }
}
int main()
{
   int i,j,temp;
   printf("Enter the no of vertex in  graph  : ");
   scanf("%d",&vertex);
   for(i=1;i<=vertex;i++)
   {
        printf("enter the vertex conneted to %d(-1 to move to next
vertex)\n",i);
        while(1)
        {
```

```
            printf("\tenter the vertices: ");
            scanf("%d",&temp);
                if(temp==-1)
                    break;
                else if(temp>vertex || temp<-1)
                    printf("\n please enter valid vertex");
                else if(temp)
                    adj[i][temp]=1;
        }
   }
    art(1,0);
    int u,w;
        for(i=1;i<=vertex;i++)
            printf("%d ",l[i]);
    printf("\n");
        for(i=1;i<=vertex;i++)
            printf("%d ",df[i]);
}
```

2)biconnected components

```
#include<stdio.h>
int df[20],l[20],adj[20][20]={0};
int num=1,vertex,start=1,top=-1;

struct edge
{
   int u;
   int v;
};
struct edge s[20];
void push(struct edge e)
{
   s[++top]=e;
}
struct edge pop()
{
   return s[top--];
}
```

```
int EdgeEquals(struct edge s,struct edge t)  // function to test equality
of edges
{
   if(s.u==t.u && s.v==t.v)
       return 1;
   else if(s.u==t.v && s.v==t.u)
       return 1;
   else
       return 0;
}

int min(int a,int b)
{
   if(a>b)
       return b;
   else
       return a;
}
void art(int u,int v)
{
 int w;
 df[u]=l[u]=num;
 num++;
   for(w=1;w<=vertex;w++)
   {
       if(adj[u][w]==1)
       {
           if(w!=v && df[w]<df[u])
           {
               struct edge t={u,w};
               push(t);
           }
           if(df[w]==0)
           {
               art(w,u);
               if(u!=start &&l[w]>=df[u])
               {
                   struct edge t;
                   struct edge s={u,w};
```

```
                    printf("Found a Bicomponent\n");
                    do
                    {
                        t=pop();
                        printf("(%d,%d)\n",t.u,t.v);
                    }while(!EdgeEquals(t,s));
                }
                l[u]=min(l[u],l[w]);
            }
            else if (w!=v)
                l[u]=min(l[u],df[w]);
        }
    }
}
int main()
{
    int i,j,temp;
    printf("Enter the no of vertex in  graph  : ");
    scanf("%d",&vertex);
    for(i=1;i<=vertex;i++)
    {
        printf("enter the vertex conneted to %d(-1 to move to next
vertex)\n",i);
        while(1)
        {
            printf("\tenter the vertices: ");
            scanf("%d",&temp);
                if(temp==-1)
                    break;
                else if(temp>vertex || temp<-1)
                    printf("\n please enter valid vertex");
                else if(temp)
                    adj[i][temp]=1;
        }
   }
    art(1,0);
    printf("Found a Bicomponent\n");
    while(top>=0)
    {
        struct edge x=pop();
```

```
        printf("(%d,%d)\n",x.u,x.v);
    }
    int u,w;
        for(i=1;i<=vertex;i++)
            printf("%d ",l[i]);
    printf("\n");
        for(i=1;i<=vertex;i++)
            printf("%d ",df[i]);
}
```

3)DISJOINT SETS

```
#include<stdio.h>
#include<stdlib.h>
int arr[20],visited[20];

int value(int i)
{
  int val=i,num=1;
  for(int j=1;j<=20;j++)
  {
      if(arr[j]==val && visited[j]==0)
      {
            num++;
            visited[j]=1;
            value(j);
      }
  }
  return num;
}
int find(int j)
{
   if(arr[j]==-1)
        printf("root is %d   \n",j);
   else
   {
        j=arr[j];
        return find(j);
   }
}
```

```
int join(int i,int j)
{
   int a = value(i);
   int b = value(j);

   if(a>=b)
       arr[j]=i;
   else
       arr[i]=j;

   printf("\n Operation succesfull\n");
}
int create(int data)
{
   int temp;
   while(1)
   {
       printf("\n Enter the vertices connected to %d(-2 if all the
vertices or no vertices are connected)  :",data);
       scanf("%d",&temp);
       if(temp==-2)
           break;

       arr[temp]=data;
       create(temp);
   }
}
int main()
{
   int i,j,n,choice;
   while(1)
   {
       printf("1.Create a set \n");
       printf("2.Create union\n");
       printf("3.Find root of a set \n");
       printf("Any other value to exit \n");
       printf("\n Enter your choice ");
       scanf("%d",&choice);

       switch(choice)
```

```
        {
            case 1:
                    printf("Enter the vertex of the root node  :");
                    scanf("%d",&n);
                    arr[n]=-1;
                    create(n);
                    for(i=1;i<=10;i++)
                        printf("%d ",arr[i]);
                    printf("\n");
                    break;
            case 2:
                    printf("\nEnter 2 root vertices to perform union\n");
                    printf("1 st vertex\n");
                    scanf("%d",&i);
                    printf("2 nd vertex\n");
                    scanf("%d",&j);
                    if(arr[i]==-1 && arr[j]==-1)
                        join(i,j);
                    else
                        printf("invalid input please enter the root ");
                    for(i=1;i<=10;i++)
                        printf("%d ",arr[i]);
                    printf("\n");
                    break;

            case 3:
                    printf("Enter a vertex to find its root :");
                    scanf("%d",&j);
                    find(j);
                    break;
            default:
                    printf("\n please enter valid operation");
                    exit(1);
        }
    }
}
```

Week 4

1)kruskal's algorithm

```
#include <stdio.h>

struct graph
{
   int v1;
   int v2;
   int weight;
};

struct graph arr[20];
int p[20];

int Find(int i)
{
   if(p[i]== -1)
       return i;
   else
       Find(p[i]);
}
void Union(int i, int j)
{
   p[i]=j;
}
void showstatus(int n)
{
 printf("\nThe arrangment in the forest is \n");
 for(int i=0;i<=n;i++)
   printf("%d ",p[i]);
}
int cost(int vertices,int edge)
{
   int u,v,j,k,i,n=0;
   int mincost=0;
  //inialize p[i] values to -1
  for(int k=0;k<=vertices;k++)
```

```
        p[k]=-1;


  for(i=0;i<=edge && n<=(vertices-1);i++)
  {
      u=arr[i].v1;
      v=arr[i].v2;
      j=Find(u);
      k=Find(v);
      if(j!=k)
      {
          n++;
          printf("\n The edge considered is %d - %d",arr[i].v1,arr[i].v2);
          Union(j,k);
          showstatus(vertices);
          mincost=mincost+arr[i].weight;
      }
  }
  if(n!=vertices-1)
      printf("\n No spanning tree is present ");
  else
   printf("\nWeight of Minimum cost spanning tree is :%d",mincost);
}
void sort(int n)
{
   int i, j;
   struct graph temp;

   for (i = 0; i < n - 1; i++)
   {
       for (j = 0; j < (n - 1-i); j++)
       {
           if (arr[j].weight > arr[j + 1].weight)
           {
               temp = arr[j];
               arr[j] = arr[j + 1];
               arr[j + 1] = temp;
           }
       }
   }
```

```
}

int main()
{
 int i,j,n,m;
 printf("Enter the no of vertices present in the graph \n");
 scanf("%d",&j);
 printf("Enter the no of edges present in the graph   :");
 scanf("%d",&n);

 for(i=0;i<n;i++)
 {
     printf("\nEnter the values of Edge   %d",i+1);
     printf("\nEnter the first vertex connected to the edge  :");
     scanf("%d",&arr[i].v1);
     printf("\nEnter the second vertex connected to the edge  :");
     scanf("%d",&arr[i].v2);
     printf("\nEnter the weight of the edge ");
     scanf("%d",&arr[i].weight);
 }
 //sorting the data
 sort(n);
 //printing in sorted order
 for (i = 0; i < n; i++)
       printf(" edges %d-%d and weight %d \n",arr[i].v1,arr[i].v2,
arr[i].weight);
   printf("\n");

 cost(j,n);
}
```

2)prims algorithm

```
#include<stdio.h>
int cost[10][10]={{0,0,0,0,0,0,0,0,0,0},
                  {0,0,4,0,0,0,0,0,8,0},
                  {0,4,0,8,0,0,0,0,11,0},
                  {0,0,8,0,7,0,4,0,0,2},
                  {0,0,0,7,0,9,14,0,0,0},
```

```
                {0,0,0,0,9,0,10,0,0,0},
                {0,0,0,4,14,10,0,2,0,0},
                {0,0,0,0,0,0,2,0,1,6},
                {0,8,11,0,0,0,0,1,0,7},
                {0,0,0,2,0,0,0,6,7,0},
                };
int main()
{
   int visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
 /****  printf("Enter number of nodes ");
   scanf("%d",&n);
   printf("Enter cost in form of adjacency matrix\n");
   //input graph
   for(i=1;i<=n;i++)
   {
       for(j=1;j<=n;j++)
       {
           scanf("%d",&cost[i][j]);
           // cost is 0 then initialize it by maximum value
           if(cost[i][j]==0)
             cost[i][j]=1000;
       }
   }***/
   n=9;
   // logic for finding minimum cost spanning tree
   visited[1]=1; // visited first node
   while(no_e<n)
   {
       min=1000;
       // in each cycle find minimum cost
       for(i=1;i<=n;i++)
       {
           for(j=1;j<=n;j++)
           {
               if(cost[i][j]==0)
                   cost[i][j]=1000;
               if(cost[i][j]<min)
               {
                   if(visited[i]!=0)
                   {
```

```
                    min=cost[i][j];
                    a=i;
                    b=j;
                }
            }
        }
        //if node is not visited
        if(visited[b]==0)
        {
            printf("\n%d to %d  cost=%d",a,b,min);
            min_cost=min_cost+min;
            no_e++;
        }
        visited[b]=1;
        // initialize with maximum value you can also use any other value
       cost[a][b]=cost[b][a]=1000;
    }
    printf("\nminimum weight is %d",min_cost);
    return 0;
}
```

3)dijkstra algorithm

```
#include<stdio.h>
void dijikstra(int G[20][20], int n, int startnode)
{
    int cost[20][20]={0}, distance[20]={0}, pred[20]={0};
    int visited[20], count, mindistance, nextnode, i,j;
    for(i=0;i < n;i++)
        for(j=0;j < n;j++)
            if(G[i][j]==0)
                cost[i][j]=999;
            else
                cost[i][j]=G[i][j];

    for(i=0;i< n;i++)
    {
        distance[i]=cost[startnode][i];
```

```
        pred[i]=startnode;
        visited[i]=0;
    }
    distance[startnode]=0;
    visited[startnode]=1;
    count=1;
    while(count < n-1){
        mindistance=999;
        for(i=0;i < n;i++)
            if(distance[i] < mindistance&&!visited[i])
            {
                mindistance=distance[i];
                nextnode=i;
            }
        visited[nextnode]=1;
        for(i=0;i < n;i++)
            if(!visited[i])
                if(mindistance+cost[nextnode][i] < distance[i])
                {
                    distance[i]=mindistance+cost[nextnode][i];
                    pred[i]=nextnode;
                }
            count++;
    }
    for(i=0;i < n;i++)
        if(i!=startnode)
        {
            printf("\nDistance of %d = %d", i, distance[i]);
            printf("\nPath = %d", i);
            j=i;
            do
            {
                j=pred[j];
                printf(" <-%d", j);
            }
            while(j!=startnode);
        }
}
void main()
{
```

```
    int i, j, n, u;
    /***printf("\nEnter the no. of vertices:: ");
    scanf("%d", &n);
    printf("\nEnter the adjacency matrix::\n");
    for(i=0;i < n;i++)
        for(j=0;j < n;j++)
            scanf("%d", &G[i][j]);***/
    n=9;
    int G[20][20]= {{0,4,0,0,0,0,0,8,0},
                    {4,0,8,0,0,0,0,11,0},
                    {0,8,0,7,0,4,0,0,2},
                    {0,0,7,0,9,14,0,0,0},
                    {0,0,0,9,0,10,0,0,0},
                    {0,0,4,14,10,0,2,0,0},
                    {0,0,0,0,0,2,0,1,6},
                    {8,11,0,0,0,0,1,0,7},
                    {0,0,2,0,0,0,6,7,0},
                    };
    printf("\nEnter the starting node:: ");
    scanf("%d", &u);
    dijikstra(G,n,u);
}
```