/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q9. Dijkastra algorithm

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<stdio.h>

#include<limits.h>

#include<stdlib.h>

int min(unsigned int result[][3])

{

int S=INT\_MAX,i,index;

//printf("hello");

for(i=1;i<5;i++)

{

if(S>result[i][0] && result[i][2]!=1)

{

S=result[i][0];

index=i;

}

//printf("HELLO");

}

//printf("%d\t%d",S,index);

return index;

}

int main()

{

unsigned int adj[5][5]={0,3,2,9,INT\_MAX,3,

0,INT\_MAX,INT\_MAX,7,

2,INT\_MAX,0,6,INT\_MAX,

9,INT\_MAX,6,0,INT\_MAX,

INT\_MAX,7,INT\_MAX,INT\_MAX,0},result[5][3];

int i,j,s=0,d;

//initialisation of resultant matrix

for(i=0;i<5;i++)

{

result[i][1]=result[i][0]=INT\_MAX;

result[i][2]=0;

}

for(i=0;i<5;i++)

{

printf("%d\t%d\t%d\n",result[i][0],result[i][1],result[i][2]);

}

result[0][0]=result[0][1]=0;

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

if(j!=s || result[i][2]!=1)

{

d=result[s][0]+adj[s][j];

if(d<result[j][0])

{

result[j][0]=d;

result[j][1]=s;

}

}

}

result[s][2]=1;

s=min(result);

//printf("\n%d\t",result[j-1][0]);

}

// printf("\nFinal Output Matrix");

for(i=0;i<5;i++)

{

printf("\n");

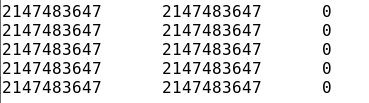
printf("\t%d",result[i][0]);

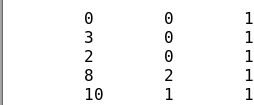
printf("\t%d",result[i][1]);

printf("\t%d",result[i][2]);

}

}





#include<iostream>

using namespace std;

int i,j,k,a,b,u,v,n,ne=1;

int min1,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

int main()

{

cout<<"Enter the No. of Vertices: ";

cin>>n;

cout<<"Enter the Cost Adjacency Matrix: \n";

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

cin>>cost[i][j];

if(cost[i][j]==0)

cost[i][j]=99;

}

}

cout<<"\nThe Edges of Minimum Cost Spanning Tree are\n";

while(ne<n)

{

for(i=1,min1=99;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(cost[i][j]<min1)

{

min1=cost[i][j];

a=u=i;

b=v=j;

}

}

}

u=find(u);

v=find(v);

if(uni(u,v))

{

cout<<"Edge %d is"<<ne++<<a<<b<<"and Cost is: "<<min1<<"\n";

mincost +=min1;

}

cost[a][b]=cost[b][a]=99;

}

printf("\nCost of Minimum Spanning Tree is: %d",mincost);

}

int find(int i)

{

while(parent[i])

i=parent[i];

return i;

}

int uni(int i,int j)

{

if(i!=j)

{

parent[j]=i;

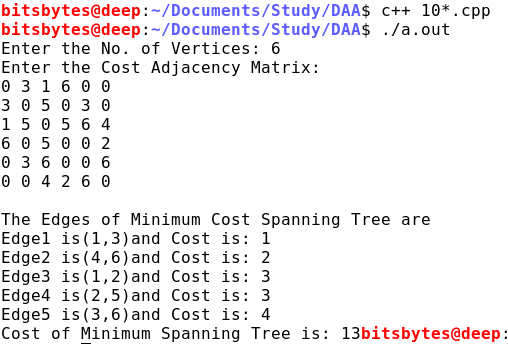
return 1;

}

return 0;

}

OUTPUT



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q12:Naive

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#include<limits>

#include<vector>

using namespace std;

class Native

{

private:

string t,p;

public: Native()

{

input();

}

Native(string t,string p )

{

this->t=t;

this->p=p;

}

void input()

{

cout<<"Enter your Text:- ";

cin>>t;

fflush(stdin);

cout<<"\nEnter your Pattern:- ";

cin>>p;

}

int search()

{

int shift=0,n=t.length(),m=p.length();

int count=0,j;

for(shift=0;shift<=n-m;shift++)

{

count=0;

for(j=0;j<m;j++)

{

if(t[shift+j]==p[j])

count++;

else

break;

}

if(j==m & count==m)

{

cout<<"Start:-"<<shift<<"\tend:-"<<shit+m;<<endl;

}

}

if(count==m)

{

cout<<"start:-"<<shift<<"\tend:-"<<shift+m;

}

return 0;

}

};

int main()

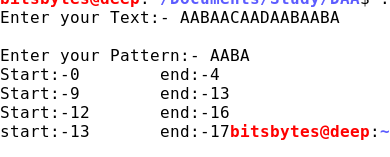
{

Native obj;

obj.search();

}

OUTPUT



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

q13:RabinKharp

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#include<string>

#define d 256

using namespace std;

class RabinKharp

{

private:

string t,p;

public:

RabinKharp()

{

input();

}

RabinKharp(string t,string s)

{

this->t=t;

this->p=s;

}

void input()

{

cout<<"Enter the Text";

cin>>t;

cout<<"Enter the Pattern";

cin>>p;

}

void Search()

{

int t\_l=t.length(),p\_l=p.length();

int flag=0,i,j;

int p1=0,t1=0,q=101;

for(int i=0;i<p\_l;i++)

{

p1=((p1+p[i]))%q;

t1=((t1+t[i]))%q;

}

for(i=0;i<=(t\_l-p\_l);i++)

{

j=0;

if(t1==p1)

{

for(j=0;j<p\_l;j++)

{

if(t[i+j]!=p[j])

break;

}

}

if(j==p\_l)

cout<<"Found Patern Started from "<<i+1<<endl;

if(i<t\_l-p\_l)

{

t1=((t1-t[i])+t[i+p\_l])%q;

}

}

}

};

int main()

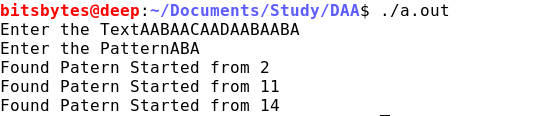
{

RabinKharp obj;

obj.Search();

}

Output



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q1. Insertion sort for worst, best and avg case analysis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

void insertion\_sort()

{

int n,i,j,key;

cout<<"Enter the Length of the array: ";

cin>>n;

int arr[n];

for(i=0;i<n;i++)

{

cin>>arr[i];

j=i-1;

key=arr[i];

while(j>=0 && key<arr[j])

{

arr[j+1]=arr[j];

j--;

}

arr[j+1]=key;

}

for(i=0;i<n;i++)

{

cout<<arr[i]<<"\t";

}

}

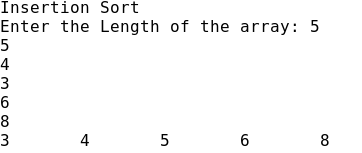
int main()

{

insertion\_sort();

}

**Output**



|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | N=5 | N=7 | N=10 |
| Worst Case | 14 | 27 | 54 |
| Best Case | 4 | 6 | 9 |
| Average Case | 7 | 12 | 24 |

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q2. Selection sort for worst, best and avg case analysis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

void Selection\_Sort()

{

int n;

cout<<"Enter the Length of the array";

cin>>n;

int arr[n],min,i,j;

for(i=0;i<n;i++)

cin>>arr[i];

for(i=0;i<n;i++)

{min=i;

for(j=i+1;j<n;j++)

{

if(arr[i]>arr[j])

{

min=j;

}

}

if(min!=i)

{

arr[i]=arr[i]^arr[min];

arr[min]=arr[i]^arr[min];

arr[i]=arr[i]^arr[min];

}

}

for(i=0;i<n;i++)

{

cout<<arr[i]<<"\t";

}

}

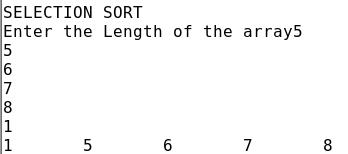
int main()

{

Selection\_Sort();

}

**Output**

****

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q3. Quick sort for worst and avg case analysis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

class QuickSort

{

public:

void Input()

{

int n;

cout<<"Enter the lenght of the array";

cin>>n;

int arr[n];

for(int i=0;i<n;i++)

{cout<<i<<"\t";cin>>arr[i];}

cout<<"Result called";

result(arr,n);

}

void result(int arr[],int n)

{

quicksort(arr,0,n-1);

for(int i=0;i<n;i++)

cout<<arr[i]<<"\t";

}

void quicksort(int arr[],int initial,int end)

{

if(initial<end)

{

int j=patition(arr,initial,end);

quicksort(arr,initial,j-1);

quicksort(arr,j+1,end);

}

}

int patition(int arr[],int initial,int end)

{

int mid=(initial+end)/2;

int i=initial,j=end;

int key=arr[mid];

while(i<j)

{

while(arr[i]<key)

i++;

while(arr[j]>key)

j--;

if(i<j)

swap(arr[i],arr[j]);

}

return j;

}

void swap(int &arr1,int &arr2)

{

arr1+=arr2;

arr2=arr1-arr2;

arr1=arr1-arr2;

}

};

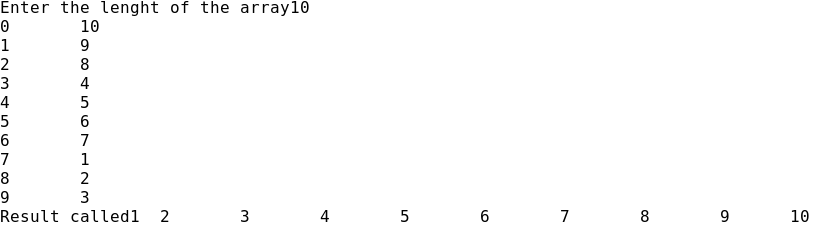
int main()

{

QuickSort obj;

obj.Input();

}



|  |  |  |  |
| --- | --- | --- | --- |
| INPUT | N=5 | N=7 | N=10 |
| WORST CASE | 18 | 33 | 63 |
| BEST CASE | 10 | 12 | 21 |
| AVERAGE CASE | 12 | 16 | 51 |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q4. Merge sort for worst and avg case analysis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

class MergeSorting

{

public:

static void MergeSort(int arr[],int initial,int ending)

{

if(initial<ending)

{

int mid=(initial+ending)/2;

MergeSort(arr,initial,mid);

MergeSort(arr,mid+1,ending);

Merge(arr,initial,mid,ending);

}

}

static void Merge(int arr[],int initial,int mid,int end)

{

int left[mid-initial+1],right[end-mid];

int i,j=0,k=0;

for(i=0;i<(mid-initial+1);i++)

left[i]=arr[initial+i];

for(i=0;i<(end-mid);i++)

right[i]=arr[mid+i+1];

i=initial;

while(j<(mid-initial+1) && k<(end-mid))

{

if(left[j]<=right[k])

{

arr[i]=left[j];

j++;

}

else if(left[j]>right[k])

{

arr[i]=right[k];

k++;

}

i++;

}

while(j<(mid-initial+1))

{

arr[i]=left[j];

i++;

j++;

}

while(k<(end-mid))

{

arr[i]=right[k];

i++;

k++;

}

}

static void input()

{

int n;

cout<<"Enter the Array Size:- ";

cin>>n;

int arr[n],i;

for(i=0;i<n;i++)

cin>>arr[i];

Result(arr,n);

}

static void Result(int arr[],int n)

{

MergeSort(arr,0,n-1);

cout<<endl;

for(int i=0;i<n;i++)

cout<<arr[i]<<"\t";

}

};

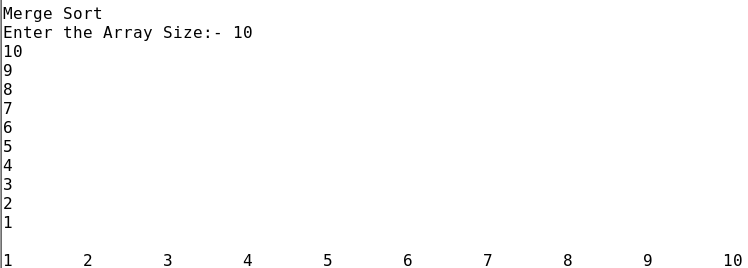
int main()

{

MergeSorting::input();

}

output



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q5. Binary search for worst and best case analysis

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

class BinarySearch

{

public:

void input()

{

int k;

cout<<"Enter the size of the list";

cin>>k;

int a[k];

cout<<"Enter the " <<k<<" Element\n";

for(int i=0;i<k;i++)

{

cin>>a[i];

}

result(a,k);

}

void result(int a[],int k)

{

int tar;

sort(a,k);

cout<<"Enter the target element:-";

cin>>tar;

search(a,tar,k);

}

void sort(int a[],int end)

{

for(int i=0;i<end;i++)

{

for(int j=i+1;j<end;j++)

{

if(a[i]>a[j])

{

a[i]^=a[j];

a[j]=a[i]^a[j];

a[i]^=a[j];

}

}

}

}

void search(int a[],int tar,int k)

{ int mid=k/2;

int left=0,right=k-1;

while(a[mid]!=tar && left<=right)

{

if(a[mid]>tar)

{

right=mid-1;

}

else if(a[mid]<tar)

{

left=mid+1;

}

mid=(left+right)/2;

}

if(a[mid]==tar)

{

cout<<tar<<" is found on "<<mid<<endl;

}

else

{

cout<<tar<<" is not found\n";

}

for(int i=0;i<k;i++)

{

cout<<a[i]<<"\t";

}

}

void swap(int &arr1,int &arr2)

{

arr1^=arr2;

arr2=arr1^arr2;

arr1^=arr2;

}

}obk;

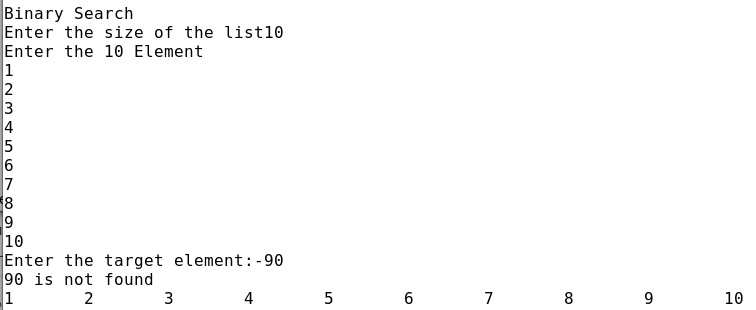
int main()

{

obk.input();

}

Output



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q6. Matrix Chain Multiplication for 6 matrices

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#define p 123456234567;

using namespace std;

void optimal\_Solution\_Value(int n,int d[][2]);

int main()

{

int n;

cout<<"Enter how many array you have?";

cin>>n;

int di[n][2];

for(int i=0;i<n;i++)

{

for(int j=0;j<2;j++)

{

cin>>di[i][j];

}

}

optimal\_Solution\_Value(n,di);

return 0;

}

void optimal\_Solution\_Value(int n,int d[][2])

{

long int m[n][n],s[n][n],q;

int i,j,k,l;

for(i=0;i<n;i++)

{

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

{

if(i<j)

{

m[i][j]=p;

}

else

{

m[i][j]=0;

}

}

}

for(l=1;l<n;l++)

{

for(i=0;i<(n-l);i++)

{

j=i+l;

for(k=i;k<j;k++)

{

q=m[i][k]+m[k+1][j]+d[i][0]\*d[k+1][0]\*d[j][1];

if(q<m[i][j])

{

m[i][j]=q;

s[i][j]=k;

}

}

}

}

cout<<endl;

for(i=0;i<n;i++)

{

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

{

cout<<m[i][j]<<"\t";

}

cout<<endl;

}

for(i=0;i<n;i++)

{

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

{

cout<<s[i][j]<<"\t";

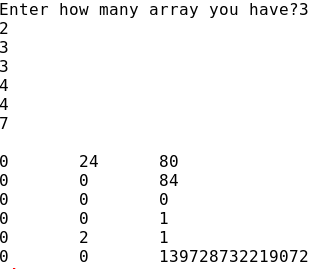
}

cout<<"\n";

}

}

Output



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q7. Min Heap Sort implementation to create a heap from the given set of values

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

class HeapSort

{

public:

void input()

{

int n;

int arr[7]={18,7,2,4,90,5,60};

heapsort(arr,7);

}

void heapify(int a[],int size,int lar)

{

int largest=lar;

int lff=2\*lar+1;

int rff=2\*lar+2;

if(lff<=size && a[lff]<a[largest])

{

largest=lff;

}

if(rff<=size && a[rff]<a[largest])

{

largest=rff;

}

if(largest!=lar)

{

swap(a[lar],a[largest]);

heapify(a,size,largest);

}

}

void heapsort(int a[],int size)

{

int i;

for(i=size/2-1;i>=0;i--)

{

heapify(a,size,i);

}

for(i=0;i<size;i++)

{

cout<<a[i]<<"\t";

}

}

void swap(int &a1,int &a2)

{

a1^=a2;

a2=a1^a2;

a1^=a2;

}

};

int main()

{

HeapSort obk;

obk.input();

}

Output



/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q8. Radix sort to sort upto 4 digit numbers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

using namespace std;

void radixSort(int arr[],int n);

int maxget(int arr[],int n);

void countSort(int arr[] ,int n,int digi);

void input(int arr[],int n);

void print(int arr[],int n)

{

for(int i=0;i<n;i++)

{

cout<<arr[i]<<"\t";

}

}

int i,j;

int main()

{

int n;

cout<<"Enter the Size of array";

cin>>n;

int arr[n];

input(arr,n);

}

void input(int arr[],int n)

{

cout<<"Enter "<<n <<"Values:-\n";

for(i=0;i<n;i++)

{

cin>>arr[i];

}

radixSort(arr,n);

print(arr,n);

}

void radixSort(int arr[],int n)

{

int m=maxget(arr,n);

for(int i=1;m/i>0;i\*=10)

{

countSort(arr,n,i);

}

}

int maxget(int arr[],int n)

{

j=arr[0];

for(int i=1;i<n;i++)

{

if(j<arr[i])

{

j=arr[i];

}

}

return j;

}

void countSort(int arr[],int n,int digi)

{

int output[n];

int i, count[10]={0};

for(i=0;i<n;i++)

{

count[(arr[i]/digi)%10]++;

}

for(i=1;i<10;i++)

{

count[i]+=count[i-1];

}

for(i=n-1;i>=0;i--)

{

output[count[(arr[i]/digi)%10]-1]=arr[i];

count[(arr[i]/digi)%10]--;

}

for(i=0;i<n;i++)

arr[i]=output[i];

}

Output

