Guru Nanak Dev Engineering College

DESIGN and ANALYSIS of ALGORITHM PRACTICAL FILE

Submitted By: -

TARANJEET SINGH

1805996

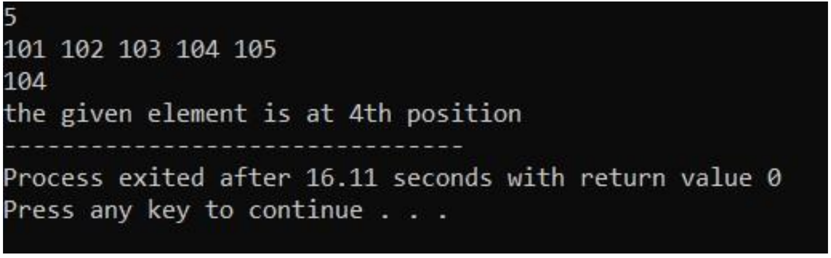
D3 CSE(E3)

1.Write a program to find out a roll no. From college database using Binary Search algorithm.

ANS-

```
#include<iostream>

using namespace std;

int binary_search(int arr[], int size,int beg,int end,int term)

{

        int mid=0;

        if(end>=beg)

        {

                mid=(beg+end)/2;

                if(arr[mid]==term)

                {

                        return mid;

                }


                else if(arr[mid]<term)

                {

                        binary_search(arr,size,mid+1,end,term);

                }


                else

                {

                        binary_search(arr,size,beg,mid-1,term);

                }


                return 0;

        }
```

```
}
int main()
{
        int n,location=0,element=0;
        //cout<<"enter no. of students"<<endl;
        cin>>n;
        int arr[n];
        //cout<<"enter elements";
        for(int i=1;i<=n;i++)
        {
                cin>>arr[i];
        }
        int beg=arr[1];
        int end=arr[n];

        int size=n;
        //cout<<"enter the element you want to find"<<endl;
        cin>>element;

        location=binary_search(arr,size,beg,end,element);



        if(location!=0)
        {
                cout<<"the given element is at "<<location<<"th position";
        }
        else

        {

                cout<<"the given element is not present";

        }

        return 0;


}
```



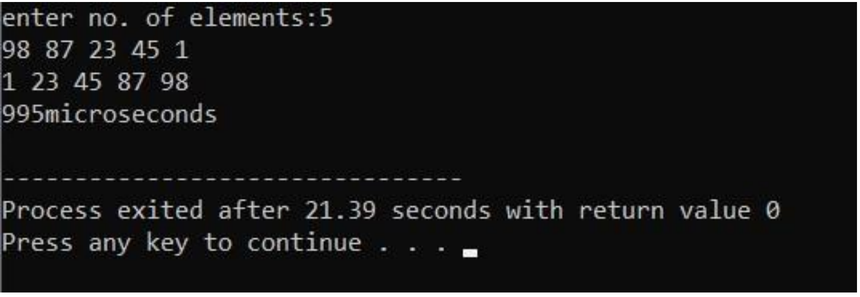
```
5
101 102 103 104 105
104
the given element is at 4th position
--------------------------------
Process exited after 16.11 seconds with return value 0
Press any key to continue . . .
```

2.Write a program to sort the class roll no. of your class using merge sort and determine the time required to sort the elements.

ANS-

```
#include<iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;


void merge(int[],int,int,int,int);


void mergesort(int arr[], int beg, int end,int size)

{

   int mid;

   if(beg<end)

   {

      mid=(beg+end)/2;

      mergesort(arr,beg,mid,size);
```

```
        mergesort(arr,mid+1,end,size);

        merge(arr,beg,mid,end,size);

    }

}

void merge(int arr[], int beg, int mid, int end,int size)

{

    int i=beg,j=mid+1,k,index=beg;

    int temp[size];

    while(i<=mid && j<=end)

    {

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

        {

            temp[index] = arr[i];

            i = i+1;

        }

        else

        {

            temp[index] = arr[j];

            j = j+1;

        }

        index++;

    }

    if(i>mid)

    {

        while(j<=end)

        {

            temp[index] = arr[j];
            index++;

            j++;

        }

    }

    else

    {

        while(i<=mid)

        {

            temp[index] = arr[i];

            index++;

            i++;

        }

    }

    k = beg;

    while(k<index)

    {

        arr[k]=temp[k];

        k++;

    }

}


int main()

{

            int n;

            cout<<"enter no. of elements:";

            cin>>n;
            int arr[n];


            for(int i=0;i<n;i++)
            {
                cin>>arr[i];
            }
            auto start = high_resolution_clock::now();
            mergesort(arr,0,n-1,n);


            for(int i=0;i<n;i++)
            {
                cout<<arr[i]<<" ";
            }
            cout<<endl;
            auto stop = high_resolution_clock::now();
            auto duration = duration_cast<microseconds>(stop - start);



            cout << duration.count() <<"microseconds"<< endl;
            return 0;
}
```



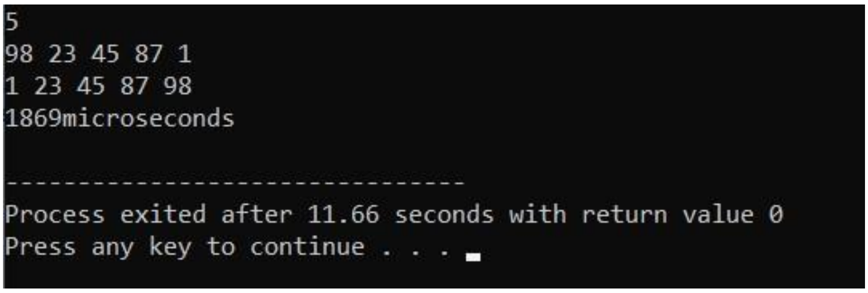
```
enter no. of elements:5
98 87 23 45 1
1 23 45 87 98
995microseconds

--------------------------------
Process exited after 21.39 seconds with return value 0
Press any key to continue . . .
```

3. Write a program to sort the university roll no. of your class using quick sort method and determine the time required to sort the elements.

ANS-

```
#include<iostream>

#include <chrono>

using namespace std::chrono;

using namespace std;


int partition(int arr[], int beg, int end)

{

        int left,right,temp,loc,flag;

        loc=left=beg;

        right=end;

        flag=0;

        while(flag!=1)

        {

                while((arr[loc]<=arr[right])&& (loc!=right))

                right--;

                if(loc==right)

                {

                        flag=1;

                }

                else if(arr[loc]>arr[right])

                {

                        temp=arr[loc];

                        arr[loc]=arr[right];

                        arr[right]=temp;

                        loc=right;

                }
                if(flag!=1)
                {
                        while((arr[loc]>=arr[left])&& (loc!=left))
                        left++;
                        if(loc==left)
                        {
                                flag=1;
                        }

                        else if(arr[loc]<arr[left])
                        {
                                temp=arr[loc];
                                arr[loc]=arr[left];
                                arr[left]=temp;
                                loc=left;
                        }
                }
        }
        return loc;
}

void quicksort(int arr[], int beg, int end)
{
        int loc;
        if(beg<end)
        {
```

```
        loc=partition(arr,beg,end);

        quicksort(arr,beg,loc-1);

        quicksort(arr,loc+1,end);

    }

}



int main()

{

    int n;

    cin>>n;

    int arr[n];

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

    {

        cin>>arr[i];

    }

    auto start = high_resolution_clock::now();

    quicksort(arr,0,n-1);



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

    {

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

    }

    cout<<endl;

    auto stop = high_resolution_clock::now();

    auto duration = duration_cast<microseconds>(stop - start);
    cout << duration.count() <<"microseconds"<< endl;

    return 0;
}
```



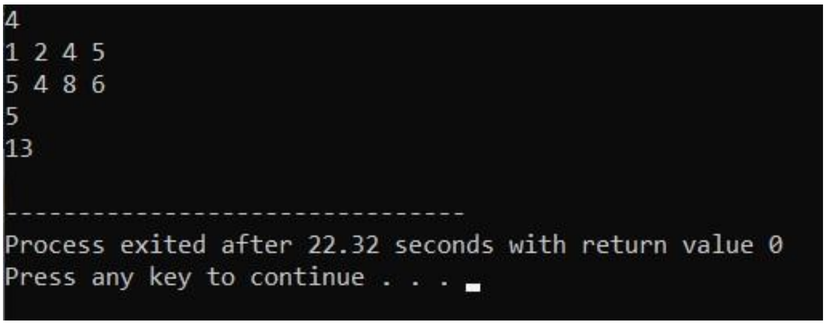
```
5
98 23 45 87 1
1 23 45 87 98
1869microseconds

--------------------------------
Process exited after 11.66 seconds with return value 0
Press any key to continue . . .
```

4. Write a program to solve 0/1 knapsack using greedy method.

ANS-

```
#include<cstring>

#include<iostream>

#include<vector>

using namespace std;

int knapsack(int* weight, int*value, int n, int maxweight)

{

    vector<vector<int>> dp(2,vector<int>(maxweight+1,0));

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

    {

        for(int j=1;j<=maxweight;++j)

        {

            if(weight[i]<=j)

            {
```

```
				dp[i%2][j]=max(value[i]+dp[(1+i)%2][j-weight[i]], dp[(i+1)%2][j]);
			}
			else
			{
				dp[i%2][j]=dp[(i+1)%2][j];
			}
		}
	}

	return dp[(n+1)%2][maxweight];
}
int main()
{
	int n;
	cin>>n;
	int *wt=new int[n];
	int *val=new int[n];

	for(int i=0;i<n;i++)
	{
		cin>>wt[i];
	}

	for(int i=0;i<n;i++)
	{
		cin>>val[i];
	}
	int w;

	cin>>w;


	cout<<knapsack(wt,val, n, w)<<endl;


	delete []wt;

	delete []val;


	return 0;

}
```



```
4
1 2 4 5
5 4 8 6
5
13

--------------------------------
Process exited after 22.32 seconds with return value 0
Press any key to continue . . .
```

5. Write a program to find minimum cost to set the phone lines to connect all the cities using Prim's algorithm.

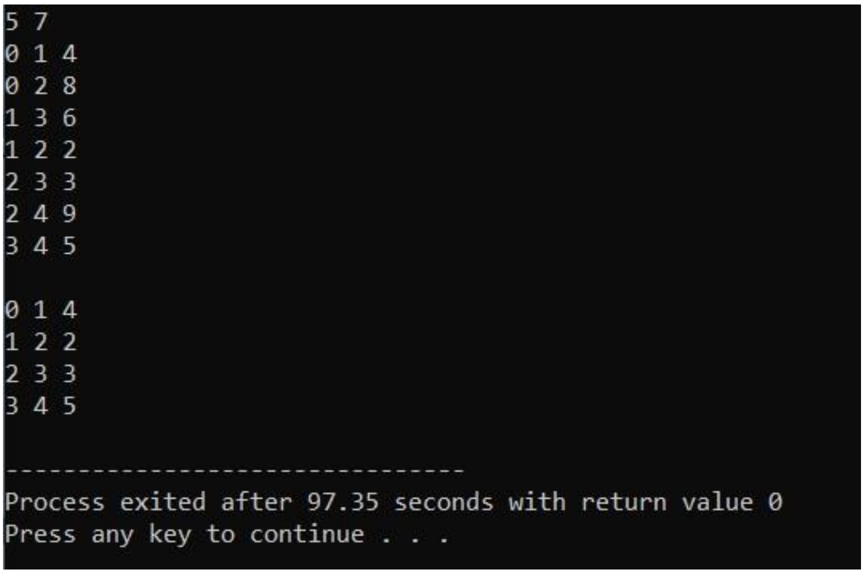
ANS-

```
#include<iostream>

#include<climits>

using namespace std;

int findminvertex(int *weights, bool*visited, int n)

{

	int minvertex=-1;


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

```
    {
        if(!visited[i]&& (minvertex==-1|| weights[i]< weights[minvertex]))
        {
            minvertex=i;
        }
    }
    return minvertex;
}

void prims(int** edges,int n)
{
    int *parent =new int[n];
    int *weights=new int [n];
    bool*visited=new bool[n];

    for(int i=0;i<n;i++)
    {
        visited[i]=false;
        weights[i]= INT_MAX;
    }

    parent[0]=-1;
    weights[0]=0;

    for(int i=0;i<n;i++)
    {
        int minvertex= findminvertex(weights,visited,n);
        visited[minvertex]=true;

        for(int j=0;j<n;j++)
        {
            if(edges[minvertex][j]!=0 && !visited[j])
            {
                if(edges[minvertex][j]<weights[j])
                {
                    weights[j]=edges[minvertex][j];
                    parent[j]=minvertex;
                }
            }
        }
    }

    for(int i=1;i<n;i++)
    {
        if(parent[i]<i)
        {
            cout<<parent[i]<<" "<< i<< " "<<weights[i]<<endl;
        }
        else
        {
            cout<<i<<" "<<parent[i]<<weights[i]<<endl;
        }
    }
}
```

```
int main()
{
        int n;
        int e;
        cin>>n>>e;
        int **edges=new int*[n];
        for(int i=0;i<n;i++)
        {
                edges[i]=new int[n];
                for(int j=0;j<n;j++)
                {
                        edges[i][j]=0;
                }
        }

        for(int i=0;i<e;i++)
        {
                int f,s,weights;

                cin>>f>>s>>weights;
                edges[f][s]=weights;
                edges[s][f]=weights;
```

```
	}

	cout<<endl;
	prims(edges,n);

	for(int i=0;i<n;i++)
	{
		delete[] edges[i];
	}

	delete [] edges;
	return 0;
}
```



```
5 7
0 1 4
0 2 8
1 3 6
1 2 2
2 3 3
2 4 9
3 4 5

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

--------------------------------
Process exited after 97.35 seconds with return value 0
Press any key to continue . . .
```

6.Write a program to find the minimum cost of connecting all the engineering colleges in your state using Kruskal's algorithm.

ANS-

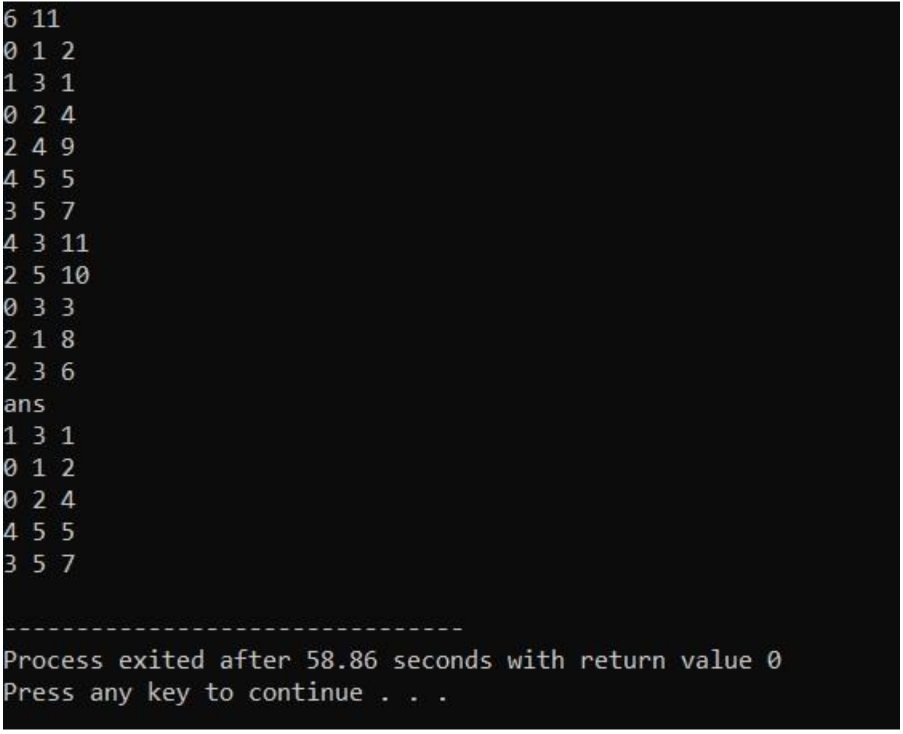
```
#include<iostream>
#include<bits/stdc++.h>
using namespace std;
```

```
class Edge

{

        public:

                int source;

                int dest;

                int weight;

};


bool compare(Edge e1,Edge e2)

{

        return e1.weight<e2.weight;

}


int findparent(int v,int *parent)

{


        if(parent[v]==v)

        {

                return v;

        }


        return findparent(parent[v],parent);

}



void kruskal(Edge *input, int n, int e)
```

```
{


        sort(input, input+e,compare);



        Edge *output=new Edge[n-1];


        int *parent=new int[n];

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

        {

                parent[i]=i;

        }


        int count=0;

        int i=0;

        while(count!=n-1)

        {


                Edge currentedge= input[i];




                int sourceparent=findparent(currentedge.source,parent);

                int destparent=findparent(currentedge.dest,parent);


                if(sourceparent!=destparent)

                {

                        output[count]=currentedge;
```

```
            count++;

            parent[sourceparent]=destparent;
        }
        i++;
    }

    cout<<"ans"<<endl;

    for(int i=0;i<n-1;i++)
    {
        if(output[i].source<output[i].dest)
        {
            cout<<output[i].source<<" "<<output[i].dest<<" "<<output[i].weight<<endl;
        }

        else
        {
            cout<<output[i].dest<<" "<<output[i].source<<" "<<output[i].weight<<endl;
        }
    }
}

int main()
{
    int n,e;
```

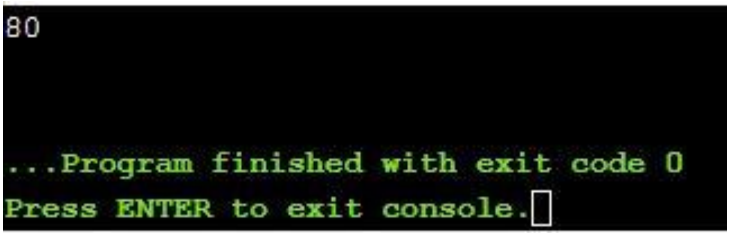
```
	cin>>n>>e;


	Edge *input=new Edge[e];


	for(int i=0;i<e;i++)
	{
		int s,d,w;
		cin>>s>>d>>w;
		input[i].source=s;
		input[i].dest=d;
		input[i].weight=w;
	}


	kruskal(input,n,e);
	return 0;
}
```



```
6 11
0 1 2
1 3 1
0 2 4
2 4 9
4 5 5
3 5 7
4 3 11
2 5 10
0 3 3
2 1 8
2 3 6
ans
1 3 1
0 1 2
0 2 4
4 5 5
3 5 7

--------------------------------
Process exited after 58.86 seconds with return value 0
Press any key to continue . . .
```

7. Write a program to find minimum route for a newspaper distributer of your locality using Greedy algorithm.

```
#include <bits/stdc++.h>
using namespace std;
int travllingSalesmanProblem(int graph[][V], int s)
{
        vector<int> vertex;
        for (int i = 0; i < V; i++)
                if (i != s)
                        vertex.push_back(i);
        int min_path = INT_MAX;
        do {
                int current_pathweight = 0;
                int k = s;
                for (int i = 0; i < vertex.size(); i++) {
                        current_pathweight += graph[k][vertex[i]];
                        k = vertex[i];
                }
                current_pathweight += graph[k][s];
                min_path = min(min_path, current_pathweight);

        } while (
                next_permutation(vertex.begin(), vertex.end()));

        return min_path;
}
int main()
{
        int graph[][V] = { { 0, 10, 15, 20 },
                                        { 10, 0, 35, 25 },
                                        { 15, 35, 0, 30 },
                                        { 20, 25, 30, 0 } };
        int s = 0;
        cout << travllingSalesmanProblem(graph, s) << endl;
        return 0;
}
```



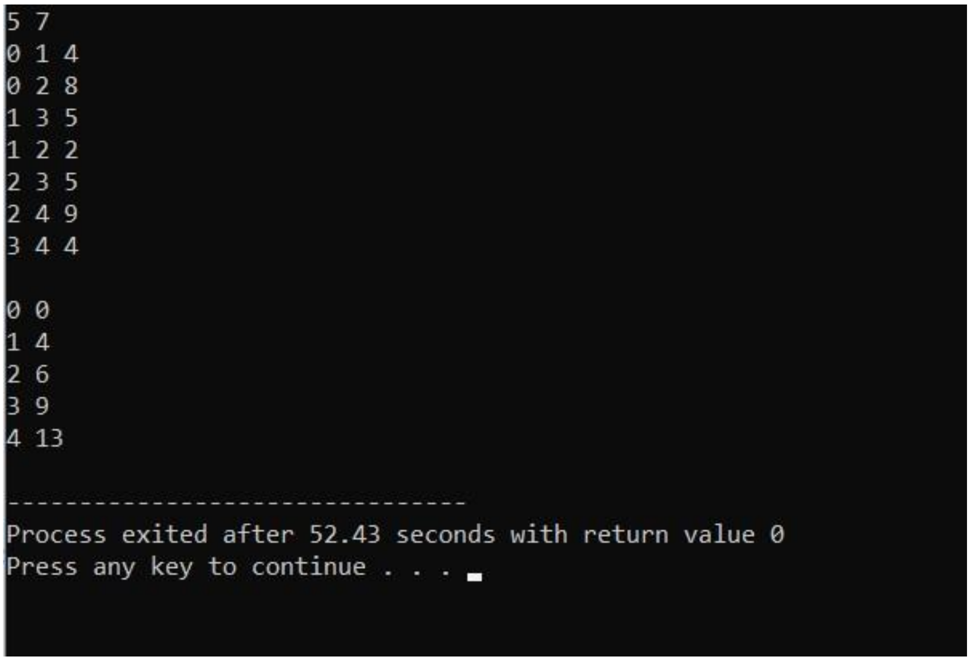
```
80

...Program finished with exit code 0
Press ENTER to exit console.
```

8. Write a program to find shortest path from your home to college using Dijkstra's algorithm.

```
#include<iostream>
using namespace std;

int findminvertex(int *distance, bool *visited, int n)
{
        int minvertex=-1;
        for(int i=0;i<n;i++)
        {
                if(!visited[i] && (minvertex ==-1 || distance[i]<distance[minvertex]))
                {
                        minvertex=i;
                }
        }

        return minvertex;
}

void dijkstra(int **edges, int n)
```

```
{
	int *distance= new int[n];
	bool *visited= new bool[n];

	for(int i=0;i<n;i++)
	{
		distance[i]=INT_MAX;
		visited[i]=false;
	}

	distance[0]=0;
	for(int i=0;i<n-1;i++)
	{
		int minvertex=findminvertex(distance,visited,n);
		visited[minvertex]=true;
		for(int j=0;j<n;j++)
		{
			if(edges[minvertex][j]!=0 && !visited[j])
			{
				int dist =distance[minvertex]+edges[minvertex][j];
				if(dist<distance[j])
				{
					distance[j]=dist;
				}
			}
		}
	}

	for(int i=0;i<n;i++)
	{
		cout<<i<<" "<<distance[i]<<endl;
	}

	delete[]visited;
	delete[]distance;
}

int main()
{
	int n;
	int e;
	cin>>n>>e;
	int **edges=new int *[n];
	for(int i=0;i<n;i++)
	{
		edges[i]=new int[n];
		for(int j=0;j<n;j++)
		{
			edges[i][j]=0;
		}
	}

	for(int i=0;i<e;i++)
	{
		int f,s,weight;
		cin>>f>>s>>weight;
		edges[f][s]=weight;
		edges[s][f]=weight;
	}
```

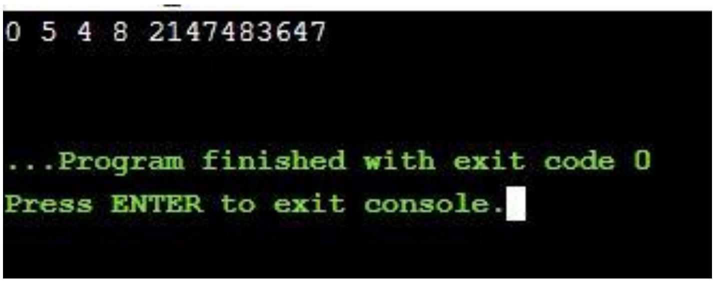
```
        cout<<endl;
        dijkstra(edges,n);

        for(int i=0;i<n;i++)
        {
                delete[]edges[i];
        }

        delete [] edges;
}
```



```
5 7
0 1 4
0 2 8
1 3 5
1 2 2
2 3 5
2 4 9
3 4 4

0 0
1 4
2 6
3 9
4 13

--------------------------------
Process exited after 52.43 seconds with return value 0
Press any key to continue . . .
```

9. Write a program to find shortest path from your home to college using Bellman-Ford algorithm.

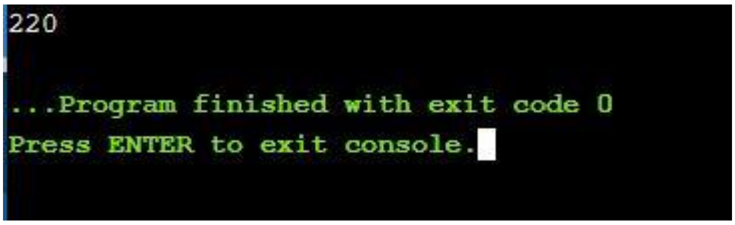
```
#include <bits/stdc++.h>
struct Edge {
  int u;  //start vertex of the edge
  int v;  //end vertex of the edge
  int w;  //w of the edge (u,v)
};
struct Graph {
  int V;        // Total number of vertices in the graph
  int E;        // Total number of edges in the graph
  struct Edge* edge;  // Array of edges
};
struct Graph* createGraph(int V, int E) {
  struct Graph* graph = new Graph;
  graph->V = V;  // Total Vertices
  graph->E = E;  // Total edges

  graph->edge = new Edge[E];
  return graph;
}
void printArr(int arr[], int size) {
  int i;
  for (i = 0; i < size; i++) {
    printf("%d ", arr[i]);
  }
  printf("\n");
}

void BellmanFord(struct Graph* graph, int u) {
  int V = graph->V;
  int E = graph->E;
  int dist[V];
  for (int i = 0; i < V; i++)
    dist[i] = INT_MAX;
  dist[u] = 0;
  for (int i = 1; i <= V - 1; i++) {
    for (int j = 0; j < E; j++) {
      // Get the edge data
      int u = graph->edge[j].u;
      int v = graph->edge[j].v;
      int w = graph->edge[j].w;
      if (dist[u] != INT_MAX && dist[u] + w < dist[v])
        dist[v] = dist[u] + w;
    }
  }
  for (int i = 0; i < E; i++) {
    int u = graph->edge[i].u;
    int v = graph->edge[i].v;
    int w = graph->edge[i].w;
    if (dist[u] != INT_MAX && dist[u] + w < dist[v]) {
      printf("Graph contains negative w cycle");
      return;
    }
  }
  printArr(dist, V);

  return;
}

int main() {

  int V = 5;
  int E = 8;
  struct Graph* graph = createGraph(V, E);
  graph->edge[0].u = 0;
  graph->edge[0].v = 1;
  graph->edge[0].w = 5;

  //edge 0 --> 2
  graph->edge[1].u = 0;
  graph->edge[1].v = 2;
  graph->edge[1].w = 4;
  graph->edge[2].u = 1;
  graph->edge[2].v = 3;
  graph->edge[2].w = 3;

  graph->edge[3].u = 2;
  graph->edge[3].v = 1;
  graph->edge[3].w = 6;

  graph->edge[4].u = 3;
  graph->edge[4].v = 2;
  graph->edge[4].w = 2;

  BellmanFord(graph, 0);  //0 is the source vertex

  return 0;
}
```



```
0 5 4 8 2147483647


...Program finished with exit code 0
Press ENTER to exit console.
```

10. Write a program to solve 0/1 knapsack using dynamic programming

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

int max(int a, int b) {
	if (a>b){
		return a;
	}else{
		return b;
	}
}

int knapSack(int W, int wt[], int val[], int n)
{
	if (n == 0 || W == 0)
		return 0;
	if (wt[n - 1] > W)
		return knapSack(W, wt, val, n - 1);
	else
		return max( val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),
			knapSack(W, wt, val, n - 1));
}
int main()
{
	int val[] = { 60, 100, 120 };
	int wt[] = { 10, 20, 30 };
	int W = 50;
	int n = sizeof(val) / sizeof(val[0]);
	cout << knapSack(W, wt, val, n);
	return 0;
}
```



```
220

...Program finished with exit code 0
Press ENTER to exit console.
```