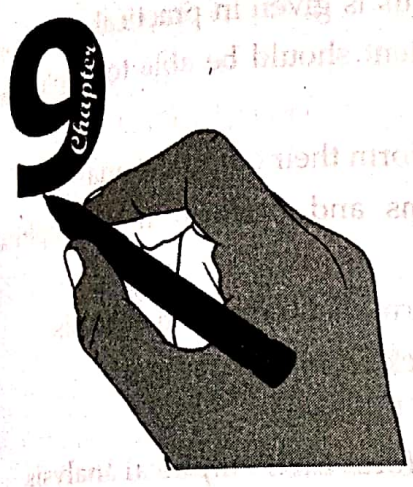


LABORATORY WORK

This course can be learnt in effective way only if we give focus is given in practical aspects of algorithms and techniques discussed in class. Therefore student should be able to implement the algorithms and analyze their behavior. Students should:

- Implement comparison sorting algorithms and perform their empirical analysis.
- Implement divide-and-conquer sorting algorithms and perform their empirical analysis.
- Implement algorithms for order statistics and perform their empirical analysis.
- Implement algorithms by using Greedy, DP and backtracking paradigm
- Implement NP-complete problems and realize their hardness.

Lab No 1: Implement comparison sorting algorithms and perform their empirical analysis.

In comparison based sorting, elements of an array are compared with each other to find the sorted array.

- Bubble sort
- Insertion sort
- Selection sort
- Merge sort
- Quick sort
- Heap sort

Lab no 1.1: Complete program of Bubble sort and their empirical analysis.

```
#include<iostream>
using namespace std;
int main()
{
        int n, i, arr[50], j, temp, count=0;
        count++;
        cout<<"Enter total number of elements :";
        count++;
        cin>>n;
        count++;
        cout<<"Enter "<<n<<" numbers :";
        count++;
        count++;
        for(i=0; i<n; i++)
        {
                cin>>arr[i];
                count++;
                count++;
                count++;
        }
```

```
    count++;
    count++;
    for(i=0; i<(n-1); i++)
    {
        count++;
        count++;
        count++;
        for(j=0; j<(n-i-1); j++)
        {
            count++;
            count++;
            if(arr[j]>arr[j+1])
            {
                temp=arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
                count++;
                count++;
                count++;
            }
        }
    }
    cout<<"Sorted list in ascending order :\n";
    count++;
    count++;
    for(i=0; i<n; i++)
    {
        cout<<arr[i]<<" ";
        count++;
        count++;
    }
    count++;
    cout<<endl<<"Total number of steps ="<<count;
    return 0;
}
```

Output

Enter total number of elements: 10

Enter 10 numbers: 2 3 55 1 89 45 32 11 65 34

Sorted list in ascending order:

1 2 3 11 32 34 45 55 65 89

Total number of steps =228

Lab no 1.2: Complete program of insertion sort and their empirical analysis.

```
#include<iostream>
using namespace std;
int main()
{
        int size, arr[50], i, j, temp, count=0;
        count++;
        cout<<"Enter Array Size : ";
        count++;
        cin>>size;
        count++;
        cout<<"Enter Array Elements : ";
        count++;
        count=count+size+1;
        count=count+size;
        for(i=0; i<size; i++)
        {
                cin>>arr[i];
                count++;
        }
        count++;
        count=count+size;
        count=count+size-1;
        for(i=1; i<size; i++)
        {
                temp=arr[i];
                count++;
                j=i-1;
                count++;
                while((temp<arr[j]) && (j>=0))
                {
                        arr[j+1]=arr[j];
                        j=j-1;
                        count++;
                        count++;
                }
                arr[j+1]=temp;
                count++;
        }
```

```
        cout<<"Array after sorting : \n";
        count++;
        count++;
        count=count+size+1;
        count=count+size;
        for(i=0; i<size; i++)
        {
                cout<<arr[i]<<" ";
                count=count+size;
        }
        cout<<endl<<"Total number of steps="<<count;
        return 0;
}
```

Output:

Enter Array Size: 10

Enter Array Elements: 1 33 2 77 65 45 89 97 66 100

Array after sorting:

1 2 33 45 65 66 77 89 97 100

Total number of steps=219

Lab no 1.3: Complete program of selection sort and their empirical analysis.

```
#include<iostream>
using namespace std;
int main()
{
        int size, arr[50], i, j, temp, count=0;
        count++;
        cout<<"Enter Array Size : ";
        count++;
        cin>>size;
        count++;
        cout<<"Enter Array Elements : ";
        count++;
        count++;
        count=count+size+1;
        count=count+size;
```

```
for(i=0; i<size; i++)
{
        cin>>arr[i];
        count++;
}
count++;
count=count+size+1;
count=count+size;
for(i=0; i<size; i++)
{
        count++;
        count++;
        count++;
        for(j=i+1; j<size; j++)
        {
                count=count+1;
                if(arr[i]>arr[j])
                {
                        temp=arr[i];
                        arr[i]=arr[j];
                        arr[j]=temp;
                        count++;
                        count++;
                        count++;
                }
        }
}
cout<<"Now the Array after sorting is :\n";
count++;
count++;
count=count+size+1;
count=count+size;
for(i=0; i<size; i++)
```

```
    {
        cout<<arr[i]<<" ";
        count++;
    }
    cout<<endl<<"Total number of steps="<<count;
    return 0;
}
```

Output

```
Enter Array Size: 10
Enter Array Elements: 23 11 22 65 43 87 90 4 3 78
Now the Array after sorting is:
3 4 11 22 23 43 65 78 87 90
Total number of steps=226
```

Lab no 1.4: Complete program of Heap sort.

```
#include<iostream>
using namespace std;
void heapify(int arr[], int n, int i)
{
    int temp;
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && arr[l] > arr[largest])
        largest = l;
    if (r < n && arr[r] > arr[largest])
        largest = r;
    if (largest != i)
    {
        temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}
```

```
void heapSort(int arr[], int n)
{
        int temp;
        for (int i = n / 2 - 1; i >= 0; i--)
                heapify(arr, n, i);
        for (int i = n - 1; i >= 0; i--)
        {
                temp = arr[0];
                arr[0] = arr[i];
                arr[i] = temp;
                heapify(arr, i, 0);
        }
}
int main()
{
        int arr[] = { 20, 7, 1, 54, 10, 15, 90, 23, 77, 25};
        int n = 10;
        int i;
        cout<<"Given array is: "<<endl;
        for (i = 0; i<n; i++)
                cout<<arr[i]<<" ";
        cout<<endl;
        heapSort(arr, n);
        printf("\n Sorted array is: \n");
        for (i = 0; i < n; ++i)
                cout<<arr[i]<<" ";
}
```

Output

Given array is:

20 7 1 54 10 15 90 23 77 25

Sorted array is:

1 7 10 15 20 23 25 54 77 90

Lab No 2: Implement divide-and-conquer sorting algorithms and perform their empirical analysis.

Divide-and-conquer

Both merge sort and quick sort employ a common algorithmic paradigm based on recursion. This paradigm, divide-and-conquer, breaks a problem into sub-problems that are similar to the original problem, recursively solves the sub-problems, and finally combines the solutions to the sub-problems to solve the original problem. Because divide-and-conquer solves sub-problems recursively, each sub-problem must be smaller than the original problem, and there must be a base case for sub-problems. We should think of a divide-and-conquer algorithm as having three parts:

- **Divide the problem** into a number of sub-problems that are smaller instances of the same problem.
- **Conquer** the sub-problems by solving them recursively. If they are small enough, solve the sub-problems as base cases.
- **Combine** the solutions to the sub-problems into the solution for the original problem.

Lab no 2.1: Complete program of merge sort.

```
#include <iostream>
using namespace std;
// A function to merge the two half into a sorted data.
void Merge(int *a, int low, int high, int mid)
{
    int i, j, k, temp[high-low+1];
    i = low;
    k = 0;
    j = mid + 1;
    // Merge the two parts into temp[].
    while (i <= mid && j <= high)
    {
        if (a[i] < a[j])
        {
            temp[k] = a[i];
            k++;
            i++;
        }
        else
        {
```

```
                temp[k] = a[j];
                k++;
                j++;
            }
        }
        // Insert all the remaining values from i to mid into temp[].
        while (i <= mid)
        {
            temp[k] = a[i];
            k++;
            i++;
        }
        while (j <= high)
        {
            temp[k] = a[j];
            k++;
            j++;
        }
        for (i = low; i <= high; i++)
        {
            a[i] = temp[i-low];
        }
}
// A function to split array into two parts.
void MergeSort(int *a, int low, int high)
{
        int mid;
        if (low < high)
        {
            mid=(low+high)/2;
            MergeSort(a, low, mid);
            MergeSort(a, mid+1, high);
            Merge(a, low, high, mid); // Merge them to get sorted output.
        }
}
int main()
{
        int n, i;
        cout<<"\n Enter the number of data element to be sorted: ";
```

```
    cin>>n;
    int arr[n];
    for(i = 0; i < n; i++)
    {
            cout<<"Enter element "<<i+1<<": ";
            cin>>arr[i];
    }
    MergeSort(arr, 0, n-1);
    cout<<"\n Sorted Data ";
    for (i = 0; i < n; i++)
            cout<<"->"<<arr[i];

     return 0;
}
```

Output

```
Enter the number of data element to be sorted: 10
Enter element 1: 22
Enter element 2: 1
Enter element 3: 55
Enter element 4: 34
Enter element 5: 77
Enter element 6: 56
Enter element 7: 33
Enter element 8: 90
Enter element 9: 77
Enter element 10: 12
Sorted Data ->1->12->22->33->34->55->56->77->77->90
```

Lab no 2.2: Complete program of Quick sort.

```
#include<iostream>
using namespace std;
int partition(int *a,int start,int end)
{
   int pivot=a[end];
   int P_index=start;
   int i,t; //t is temporary variable
   for(i=start;i<end;i++)
```

```
    {
        if(a[i]<=pivot)
        {
        t=a[i];
        a[i]=a[P_index];
        a[P_index]=t;
        P_index++;
      }
    }
    t=a[end];
    a[end]=a[P_index];
    a[P_index]=t;
    return P_index;
}
void Quicksort(int *a,int start,int end)
{
  if(start<end)
  {
       int P_index=partition(a,start,end);
       Quicksort(a,start,P_index-1);
       Quicksort(a,P_index+1,end);
  }
}
int main()
{
       int n;
       cout<<"Enter number of elements: ";
       cin>>n;
       int a[n];
       cout<<"Enter the array elements:\n";
       for(int i=0;i<n;i++)
       {
            cin>>a[i];
```

```
    }
    Quicksort(a,0,n-1);
    cout<<"After Quick Sort the array is:\n";
    for(int i=0;i<n;i++)
    {
            cout<<a[i]<<" ";
    }
    return 0;
}
```

Output

How many elements? 10

Enter array elements: 22 1 44 3 76 8 90 45 33 28

Array after sorting: 1 3 8 22 28 33 44 45 76 90

Lab no 2.3: Sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
# include <stdio.h>
# include <conio.h>
# include <time.h>
#include<stdlib.h>>
void Exch(int *p, int *q)
{
    int temp = *p;
    *p = *q;
    *q = temp;
}
void QuickSort(int a[], int low, int high)
{
    int i, j, key, k;
    if(low>=high)
    return;
    key=low; i=low+1; j=high;
    while(i<=j)
```

```
        {
                while ( a[i] <= a[key] )
                        i=i+1;
                while ( a[j] > a[key] )
                        j=j-1;
                if(i<j) Exch(&a[i], &a[j]);
        }
        Exch(&a[j], &a[key]);
        QuickSort(a, low, j-1);
        QuickSort(a, j+1, high);
}
int main()
{
        int n, a[10000],k;
        clock_t st,et;
        printf("\n Enter How many Numbers: ");
        scanf("%d", &n);
        printf("\n The Random Numbers are:\n");
        for(k=1; k<=n; k++)
        {
                a[k]=rand();
                printf("%d\t", a[k]);
        }
        st=clock();
        QuickSort(a, 1, n);
        et=clock();
        printf("\nSorted Numbers are: \n ");
        for(k=1; k<=n; k++)
                printf("%d\t", a[k]);
        printf("\n The time taken is %f",(float)(et-st)/CLOCKS_PER_SEC);
        return 0;
}
```

Output

```
Enter how many Numbers: 60
The Random Numbers are:
41   18467  6334   26500  19169  15724  11478  29358  26962  24464
5705  28145  23281  16827  9961   491    2995   11942  4827   5436
32391  14604  3902   153    292    12382  17421  18716  19718  19895
5447  21726  14771  11538  1869   19912  25667  26299  17035  9894
28703  23811  31322  30333  17673  4664   15141  7711   28253  6868
25547  27644  32662  32757  20037  12859  8723   9741   27529  778

Sorted Numbers are:
41   153   292   491   778   1869   2995   3902   4664   4827
5436  5447  5705  6334  6868  7711   8723   9741   9894   9961
11478  11538  11942  12382  12859  14604  14771  15141  15724  16827
17035  17421  17673  18467  18716  19169  19718  19895  19912  20037
21726  23281  23811  24464  25547  25667  26299  26500  26962  27529
27644  28145  28253  28703  29358  30333  31322  32391  32662  32757
The time taken is 0.000000
```

Lab No 2.4: Implement a parallelized Merge Sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
# include <stdio.h>
# include <conio.h>
#include<time.h>
void Merge(int a[], int low, int mid, int high)
{
    int i, j, k, b[20];
    i=low; j=mid+1; k=low;
    while ( i<=mid && j<=high )
    {
        if( a[i] <= a[j] )
            b[k++] = a[i++] ;
```

```
                else
                        b[k++] = a[j++] ;
        }
        while (i<=mid)
                b[k++] = a[i++] ;
        while (j<=high)
                b[k++] = a[j++] ;
        for(k=low; k<=high; k++)
                a[k] = b[k];
}
void MergeSort(int a[], int low, int high)
{
        int mid;
                if(low >= high)
        return;
        mid = (low+high)/2 ;
        MergeSort(a, low, mid);
        MergeSort(a, mid+1, high);
        Merge(a, low, mid, high);
}
void main()
{
        int n, a[2000],k;
        clock_t st,et;
        double ts;
        clrscr();
        printf("\n Enter How many Numbers:");
        scanf("%d", &n);
        printf("\nThe Random Numbers are:\n");
        for(k=1; k<=n; k++)
        {
                a[k]=rand();
                printf("%d\t", a[k]);
```

```
}
st=clock();
MergeSort(a, 1, n);
et=clock();
ts=(double)(et-st)/CLOCKS_PER_SEC;
printf("\n Sorted Numbers are : \n ");
for(k=1; k<=n; k++)
        printf("%d\t", a[k]);
printf("\nThe time taken is %e",ts);
getch();
}
```

Output

```
Enter How many Numbers:15

The Random Numbers are:
346     130     10982   1090    11656   7117    17595   6415    22948   31126
9004    14558   3571    22879   18492
 Sorted Numbers are :
 130    346     1090    3571    6415    7117    9004    10982   11656   14558
17595   18492   22879   22948   31126
The time taken is 0.000000e+00_
```

Lab No 3: Implement algorithms for order statistics and perform their empirical analysis.

```
#include<iostream>
#include<algorithm>
#include<climits>
using namespace std;
int partition(int arr[], int l, int r, int k);
int findMedian(int arr[], int n)
{
        sort(arr, arr+n);  // Sort the array
        return arr[n/2];   // Return middle element
}
int kthSmallest(int arr[], int l, int r, int k)
{
```

```
    // If k is smaller than number of elements in array
    if (k > 0 && k <= r - l + 1)
    {
        int n = r-l+1; // Number of elements in arr[l..r]
        int i, median[(n+4)/5]; // There will be floor((n+4)/5) groups;
        for (i=0; i<n/5; i++)
           median[i] = findMedian(arr+l+i*5, 5);
        if (i*5 < n) //For last group with less than 5 elements
        {
           median[i] = findMedian(arr+l+i*5, n%5);
           i++;
        }
        int medOfMed = (i == 1)? median[i-1]:
        kthSmallest(median, 0, i-1, i/2);
        int pos = partition(arr, l, r, medOfMed);
        if (pos-l == k-1)
            return arr[pos];
        if (pos-l > k-1)  // If position is more, recur for left
            return kthSmallest(arr, l, pos-1, k);
        return kthSmallest(arr, pos+1, r, k-pos+l-1);
    }
    return INT_MAX;
}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int partition(int arr[], int l, int r, int x)
{
    int i;
```

```
    for (i=l; i<r; i++)
        if (arr[i] == x)
            break;
        swap(&arr[i], &arr[r]);
        i = l;
        for (int j = l; j <= r - 1; j++)
        {
            if (arr[j] <= x)
            {
                swap(&arr[i], &arr[j]);
                i++;
            }
          }
        swap(&arr[i], &arr[r]);
        return i;
}
int main()
{
  int arr[100];
  int n,i,k;
  cout<<"Enter number of elements";
  cin>>n;
  cout<<"Enter "<<n<<" elements"<<endl;
  for(i=0;i<n;i++)
  {
      cin>>arr[i];
  }
  cout<<"Enter value of k:"<<endl;
  cin>>k;
  cout <<k<< "th smallest element is "<<endl << kthSmallest(arr, 0, n-1, k);
  return 0;
}
```

Output

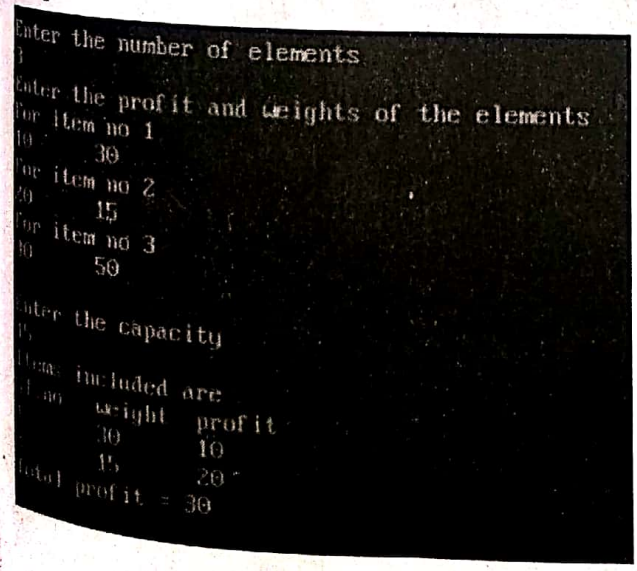
```
Enter number of elements10
Enter 10 elements
88 77 66 55 44 33 22 11 9 100
Enter value of k:
8
8th smallest element is
77
```

4. Implement 0/1 Knapsack problem using Dynamic Programming.

```
#include<stdio.h>
#include<conio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j)
{
        return ((i>j)?i:j);
}
int knap(int i,int j)
{
        int value;
        if(v[i][j]<0)
        {
                if(j<w[i])
                        value=knap(i-1,j);
                else
                        value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
                v[i][j]=value;
        }
        return(v[i][j]);
}
void main()
{
        int profit,count=0;
        clrscr();
        printf("\nEnter the number of elements\n");
        scanf("%d",&n);
        printf("Enter the profit and weights of the elements\n");
        for(i=1;i<=n;i++)
        {
                printf("For item no %d\n",i);
                scanf("%d%d",&p[i],&w[i]);
        }
```

```
printf("\n Enter the capacity \n");
scanf("%d", &cap);
for(i=0;i<=n;i++)
        for(j=0;j<=cap;j++)
                if((i==0)||(j==0))
                        v[i][j]=0;
                else
                        v[i][j]=-1;
                profit=knap(n,cap);
                i=n;
                j=cap;
                while(j!=0&&i!=0)
                {
                        if(v[i][j]!=v[i-1][j])
                        {
                                x[i]=1;
                                j=j-w[i];
                                i--;
                        }
                        else
                                i--;
                }
                printf("Items included are\n");
                printf("Sl.no\tweight\tprofit\n");
                for(i=1;i<=n;i++)
                        if(x[i])
                printf("%d\t%d\t%d\n",++count,w[i],p[i]);
                printf("Total profit = %d\n",profit);
                getch();
}
```

Output

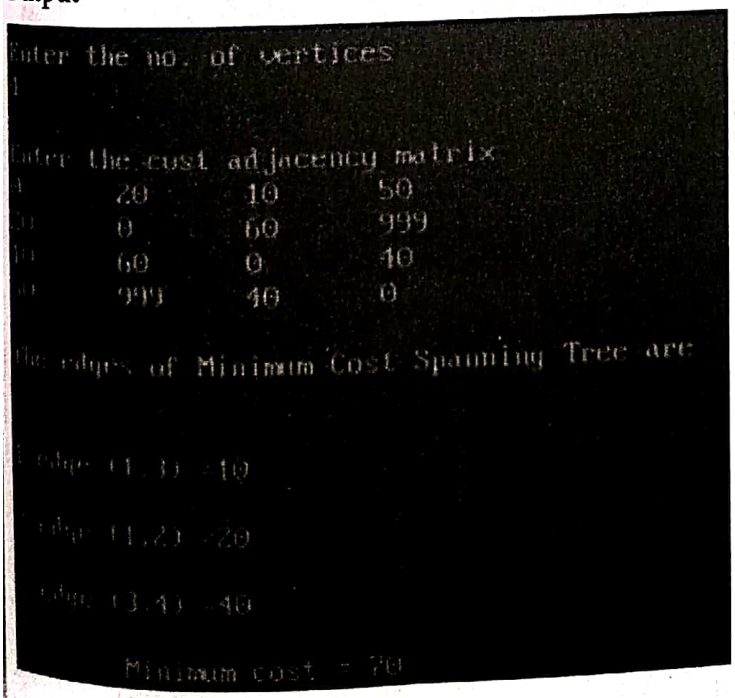


5.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
        clrscr();
        printf("\n\n\tImplementation of Kruskal's algorithm\n\n");
        printf("\nEnter the no. of vertices\n");
        scanf("%d",&n);
        printf("\nEnter the cost adjacency matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=999;
                }
        }
        printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");
        while(ne<n)
        {
                for(i=1,min=999;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                                if(cost[i][j]<min)
                                {
                                        min=cost[i][j];
                                        a=u=i;
                                        b=v=j;
                                }
                        }
                }
```

```
            u=find(u);
            v=find(v);
            if(uni(u,v))
            {
                  printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
                  mincost +=min;
            }
            cost[a][b]=cost[b][a]=999;
      }
      printf("\n\tMinimum cost = %d\n",mincost);
      getch();

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



6. Find a subset of a given set S = {s1, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

```
#include<stdio.h>
#include<conio.h>
int s[10] , x[10],d ;
void sumofsub ( int , int , int ) ;
void main ()
{
        int n , sum = 0 ;
        int i ;
        clrscr () ;
        printf ( " \n Enter the size of the set : " ) ;
        scanf ( "%d" , &n ) ;
        printf ( " \n Enter the set in increasing order:\n" ) ;
        for ( i = 1 ; i <= n ; i++ )
                scanf ("%d", &s[i] ) ;
        printf ( " \n Enter the value of d : \n " ) ;
        scanf ( "%d" , &d ) ;
        for ( i = 1 ; i <= n ; i++ )
                sum = sum + s[i] ;
                if ( sum < d | | s[1] > d )
                        printf ( " \n No subset possible : " ) ;
                else
                        sumofsub ( 0 , 1 , sum ) ;
        getch () ;
}
void sumofsub ( int m , int k , int r )
{
        int i=1 ;
        x[k] = 1 ;
        if ( ( m + s[k] ) == d )
        {
```

```
        printf("Subset:");
        for ( i = 1 ; i <= k ; i++ )
        if ( x[i] == 1 )
        printf ( "\t%d" , s[i] ) ;
        printf ( "\n" ) ;
    }
    else
        if ( m + s[k] + s[k+1] <= d )
            sumofsub ( m + s[k] , k + 1 , r - s[k] ) ;
        if ( ( m + r - s[k] >= d ) && ( m + s[k+1] <=d ) )
        {
            x[k] = 0;
            sumofsub ( m , k + 1 , r - s[k] ) ;
        }
}
```

Output

```
Enter the size of the set : 5

Enter the set in increasing order:
1       2       5       6       8

Enter the value of d :
9
Subset: 1       2       6
Subset: 1       8
```

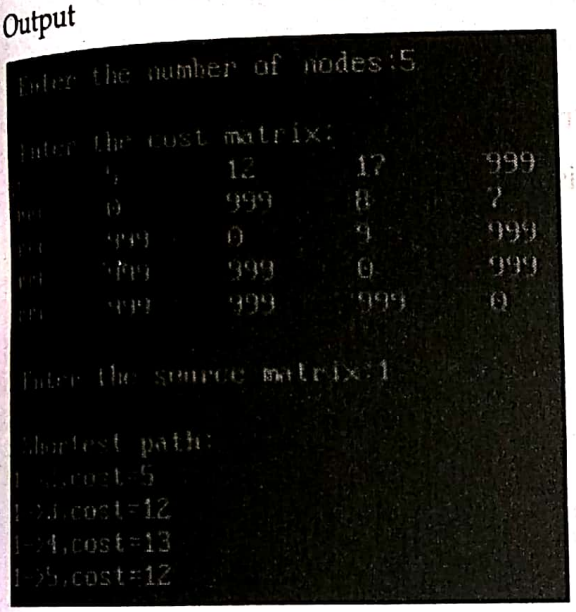
7. From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

```
#include<stdio.h>
#include<conio.h>
#define infinity 999
void dij(int n,int v,int cost[10][10],int dist[100])
{
    int i,u,count,w,flag[10],min;
    for(i=1;i<=n;i++)
    flag[i]=0,dist[i]=cost[v][i];
    count=2;
```

```
        while(count<=n)
        {
                min=99;
                for(w=1;w<=n;w++)
                        if(dist[w]<min && !flag[w])
                                min=dist[w],u=w;
                        flag[u]=1;
                        count++;
                        for(w=1;w<=n;w++)
                                if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
                                        dist[w]=dist[u]+cost[u][w];
        }
}
void main()
{
        int n,v,i,j,cost[10][10],dist[10];
        clrscr();
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the cost matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                        if(cost[i][j]==0)
                                cost[i][j]=infinity;
                }
        }
        printf("\n Enter the source matrix:");
        scanf("%d",&v);
        dij(n,v,cost,dist);
        printf("\n Shortest path:\n");
```

```
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
    getch();
}
```

Output



8. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

```
#include<stdio.h>
#include<conio.h>
int a,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
    clrscr();
    printf("\n Enter the number of nodes:");
    scanf("%d",&n);
    printf("\n Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
```

```
        }
        visited[1]=1;
        printf("\n");
        while(ne<n)
        {
                for(i=1,min=999;i<=n;i++)
                        for(j=1;j<=n;j++)
                                if(cost[i][j]<min)
                                        if(visited[i]!=0)
                                        {
                                                min=cost[i][j];
                                                a=u=i;
                                                b=v=j;
                                        }
                                        if(visited[u]==0 || visited[v]==0)
                                        {
                                printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
                                mincost+=min;
                                visited[b]=1;
                        }
                        cost[a][b]=cost[b][a]=999;
                }
                printf("\n Minimun cost=%d",mincost);
        getch();
}
```

Output

```
 Enter the number of nodes:4

 Enter the adjacency matrix:
0       20      10      50
20      0       60      999
10      60      0       40
50      999     40      0


 Edge 1:(1 3) cost:10
 Edge 2:(1 2) cost:20
 Edge 3:(3 4) cost:40
 Minimun cost=70
```

9. Implement All-Pairs Shortest Paths Problem using Floyd's algorithm. Parallelize this algorithm and determine the speed-up achieved.

```
#include<stdio.h>
#include<conio.h>
int min(int,int);
void floyds(int p[10][10],int n)
{
        int i,j,k;
        for(k=1;k<=n;k++)
                for(i=1;i<=n;i++)
                        for(j=1;j<=n;j++)
                                if(i==j)
                                        p[i][j]=0;
                                else
                                        p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b)
{
        if(a<b)
                return(a);
        else
                return(b);
}
void main()
{
        int p[10][10],w,n,e,u,v,i,j;;
        clrscr();
        printf("\n Enter the number of vertices:");
        scanf("%d",&n);
        printf("\n Enter the number of edges:\n");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
```

```
            p[i][j]=999;
        }
        for(i=1;i<=e;i++)
        {
            printf("\n Enter the end vertices of edge%d with its weight \n",i);
            scanf("%d%d%d",&u,&v,&w);
            p[u][v]=w;
        }
        printf("\n Matrix of input data:\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
                printf("%d \t",p[i][j]);
            printf("\n");
        }
        floyds(p,n);
        printf("\n Transitive closure:\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
                printf("%d \t",p[i][j]);
            printf("\n");
        }
        printf("\n The shortest paths are:\n");
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
            {
                if(i!=j)
                    printf("\n <%d,%d>=%d",i,j,p[i][j]);
            }
    getch();
}
```

Output

```
Enter the number of vertices:4

Enter the number of edges:
6

Enter the end vertices of edge1 with its weight
1       3       3

Enter the end vertices of edge2 with its weight
2       1       2

Enter the end vertices of edge3 with its weight
3       2       7

Enter the end vertices of edge4 with its weight
3       4       1

Enter the end vertices of edge5 with its weight
4       1       6_
```

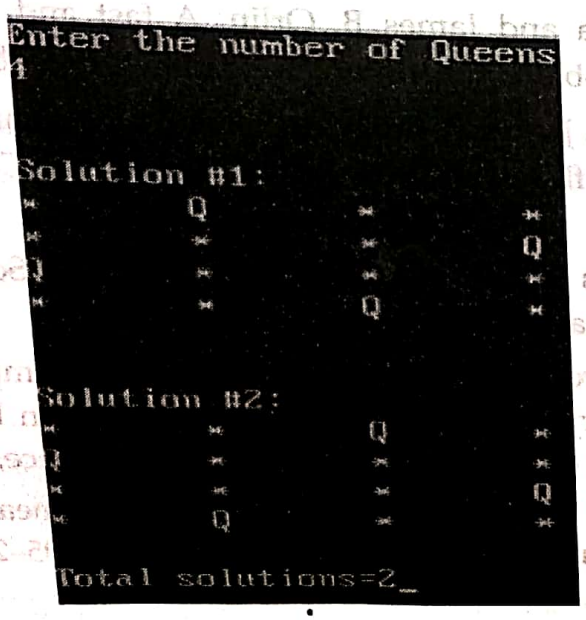
```
999     999     3       999
2       999     999     999
999     7       999     1
6       999     999     999

Transitive closure:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0

The shortest paths are:

<1,2>=10
<1,3>=3
<1,4>=4
<2,1>=2
<2,3>=5
<2,4>=6
<3,1>=7
<3,2>=7
<3,4>=1
<4,1>=6
<4,2>=16
<4,3>=9
```

10. Implement N Queen's problem using Back Tracking.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
{
        int i;
        for(i=1;i<pos;i++)
        {
                if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
                        return 0;
        }
        return 1;
}
void print_sol(int n)
{
        int i,j;
        count++;
        printf("\n\nSolution #%d:\n",count);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        if(a[i]==j)
                                printf("Q\t");
                        else
                                printf("*\t");
                }
                printf("\n");
        }
}
void queen(int n)
{
        int k=1;
        a[k]=0;
        while(k!=0)
        {
                a[k]=a[k]+1;
                while((a[k]<=n)&&!place(k))
                        a[k]++;
                if(a[k]<=n)
                {
```

```
            if(k==n)
                print_sol(n);
            else
            {
                k++;
                a[k]=0;
            }
        }
    else
        k--;
    }
}
void main()
{
    int i,n;
    clrscr();
    printf("Enter the number of Queens\n");
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
    getch();
}
```

Output



□□□