DEV BHOOMI INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

Year: 3rd   Semester: 5th

Algorithm lab- PCS-553

LAB MANUAL

Prepared By: HOD (CSE)

DEV BHOOMI INSTITUTE OF TECHNOLOGY

Department of Computer Science and Engineering

INDEX

LAB MANUAL

	Course Name: Design and Analysis of Algorithm.	**Experiment No.** 1	
	Course Code : ** PCS-553 **Faculty : Mr. Dhajveer Singh Rai	**Branch:** CSE	**Semester:** V

Objective: Implement Recursive Binary search and linear search and determine the time taken to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

Program:

```
/* Implementation of recursive binary search and sequential search */

#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<stdlib.h>
#define max 20

int pos;
int binsearch(int,int[],int,int,int);
int linsearch(int,int[],int);

void main()
{
  int ch=1;
  double t;
  int n,i,a[max],k,op,low,high,pos;
  clock_t begin,end;
  clrscr();
  while(ch)
  {
  printf("\n.....MENU.....\n 1.Binary Search\n 2.Linear     Search\n 3.Exit\n");
  printf("\nEnter your choice\n");
  scanf("%d",&op);

  switch(op)
  {
      case 1:printf("\nEnter the number of elements \n");
                 scanf("%d",&n);
           printf("\nEnter the elements of an array in order\n");
                 for(i=0;i<n;i++)
                        scanf("%d",&a[i]);
           printf("\nEnter the elements to be searched\n");
```

```
                scanf("%d",&k);
                low=0;high=n-1;
                begin=clock();
                pos=binsearch(n,a,k,low,high);
                end=clock();
                if(pos==-1)
                  printf("\n\n Unsuccessful search");
                else
            printf("\n Element %d is found at position %d",k,pos+1);
                                    printf("\n Time taken is %lf CPU1 cycles\n",(end-
                              begin)/CLK_TCK);
                  getch();
                  break;
     case 2:printf("\nEnter the number of elements\n");
                scanf("%d",&n);
                printf("\nEnter the elements of an array\n");
                for(i=0;i<n;i++)
                       scanf("%d",&a[i]);
                              printf("\nEnter the elements to be searched\n");
                 scanf("%d",&k);
                 begin=clock();
                 pos=linsearch(n,a,k);
                 end=clock();
                if(pos==-1)
                        printf("\n\n Unsuccessful search");
                else
                                    printf("\n Element %d is found at position %d",k,pos+1);
                            printf("\n Time taken is %lf CPU cycles\n",(end-begin)/CLK_TCK);
                getch();
                break;
   default:printf("\nInvalid choice entered\n");
               exit(0);

  }


  printf("\n Do you wish to run again (1/0) \n");
  scanf("%d",&ch);
  }
  getch();
}


int binsearch(int n,int a[],int k,int low,int high)
{
 int mid;
```

```
    delay(1000);
    mid=(low+high)/2;
    if(low>high)
      return -1;
    if(k==a[mid])
      return(mid);
    else
     if(k<a[mid])
       return binsearch(n,a,k,low,mid-1);
     else
      return binsearch(n,a,k,mid+1,high);

  }

  int linsearch(int n,int a[],int k)
  {
    delay(1000);
    if(n<0)
     return -1;
    if(k==a[n-1])
      return(n-1);
     else
      return linsearch(n-1,a,k);

   }
```

OUTPUT

Case 1

```
.....MENU.....
1. Binary Search
2. Linear Search
3. Exit

Enter your choice
1

Enter the number of elements
3

Enter the elements of an array
4
8
12
```

```
Enter the elements to be searched
12

 Element 12 is found at position 2
 Time taken is 1.978022 CPU1 cycles
```

Case 2

```
.....MENU.....
 1.Binary Search
 2.Linear Search
 3.Exit

Enter your choice
2

Enter the number of elements
4

Enter the elements of an array
3
6
9
12

Enter the elements to be searched
9

 Element 9 is found at position 3
 Time taken is 3.021978 CPU cycles
```

Outcome:

To understand theImplementation of recursive binary search and sequential search**.**

LAB MANUAL

	Course Name: Design and Analysis of Algorithm.	**Experiment No.** 2	
	Course Code : **PCS-553 **Faculty :Mr. Dhajveer Singh Rai	**Branch:** CSE	**Semester:**V

Objective: Sort a given set of elements using the Heap sort method and determine the time taken 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.

Program:

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

void heapcom(int a[],int n)
{
	int i,j,k,item;
	for(i=1;i<=n;i++)
	{
		item=a[i];
		j=i;
		k=j/2;
		while(k!=0 && item>a[k])
		{
			a[j]=a[k];
			j=k;
			k=j/2;
		}
		a[j]=item;
	}
}
void adjust(int a[],int n)
{
	int item,i,j;
	j=1;
	item=a[j];
	i=2*j;
	while(i<n)
	{
		if((i+1)<n)
```

```
                {
                        if(a[i]<a[i+1])
                        i++;
                }
                if(item<a[i])
                {
                        a[j]=a[i];
                        j=i;
                        i=2*j;
                }
                else
                break;
        }
        a[j]=item;
}
void heapsort(int a[],int n)
{
        int i,temp;
        delay(1000);
        heapcom(a,n);
        for(i=n;i>=1;i--)
        {
                temp=a[1];
                a[1]=a[i];
                a[i]=temp;
                adjust(a,i);
        }
}
void main()
        {
          int i,n,a[20],ch=1;
          clock_t start,end;
          clrscr();
          while(ch)
           {
                printf("\n enter the number of elements to sort\n");
                scanf("%d",&n);
                printf("\n enter the elements to sort\n");
                for(i=1;i<=n;i++)
                   scanf("%d",&a[i]);
                start=clock();
                heapsort(a,n);
                end=clock();
                printf("\n the sorted list of elemnts is\n");
                for(i=1;i<=n;i++)
                   printf("%d\n",a[i]);
                printf("\n Time taken is %lf CPU cycles\n",(end-start)/CLK_TCK);
                printf("do u wish to run again (0/1)\n");
                scanf("%d",&ch);
        }
        getch();
}
```

OUTPUT

```
enter the number of elements to sort
5

 enter the elements to sort
8
5
6
3
1

 the sorted list of elements is
1
3
5
6
8
```

Outcome:

To understand the implementation of heap sort and time taken by the algorithm.

DEV BHOOMI INSTITUTE OF TECHNOLOGY

LAB MANUAL

Course Name: Design and Analysis of Algorithm.	**Experiment No.** 3	
Course Code : **PCS-553 **Faculty :Mr. Dhajvir Singh Rai	**Branch:** CSE	**Semester:**V

Objective: Sort a given set of elements using Merge sort method and determine the time taken 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.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
#define max 20
void mergesort(int a[],int low,int high);
void merge(int a[],int low,int mid,int high);
void main()
{
        int n,i,a[max],ch=1;
        clock_t start,end;
        clrscr();
        while(ch)
         {
           printf("\n\t enter the number of elements\n");
           scanf("%d",&n);
           printf("\n\t enter the elements\n");
           for(i=0;i<n;i++)
                 scanf("%d",&a[i]);
           start= clock();
           mergesort(a,0,n-1);
           end=clock();
           printf("\nthe sorted array is\n");
           for(i=0;i<n;i++)
                 printf("%d\n",a[i]);
           printf("\n\ntime taken=%lf",(end-start)/CLK_TCK);
           printf("\n\ndo u wish to continue(0/1) \n");
           scanf("%d",&ch);
        }
        getch();
}
```

```
void mergesort(int a[],int low,int high)
{
        int mid;
        delay(100);
        if(low<high)
        {
                mid=(low+high)/2;
                mergesort(a,low,mid);
                mergesort(a,mid+1,high);
                merge(a,low,mid,high);
        }
}

void merge(int a[],int low,int mid,int high)
{
        int i,j,k,t[max];
        i=low;
        j=mid+1;
        k=low;
         while((i<=mid) && (j<=high))
        if(a[i]<=a[j])
        t[k++]=a[i++];
        else
        t[k++]=a[j++];
        while(i<=mid)
        t[k++]=a[i++];
        while(j<=high)
        t[k++]=a[j++];
        for(i=low;i<=high;i++)
        a[i]=t[i];
}
```

OUTPUT

```
Enter the number of elements

5
Enter the elements
6
3
4
1
```

```
9
        The sorted array is
1
3
4
6
9

time taken=0.824176
```

Outcome:

To understand the implementation of merge sort and the time complexity of the algorithm.

OBJECTIVE: Sort a given set of elements using Selection sort and hence find the time required to sort 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 versusn.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
 void main()
  {
        int i,n,j,min,k,a[20],ch=1;
        clock_t begin,end;
        clrscr();
        while(ch)
        {
          printf("\n enter the number of elements\n");
          scanf("%d",&n);
          printf("\n enter the elements to be sorted\n");
          for(i=0;i<n;i++)
            scanf("%d",&a[i]);
          begin=clock();
          for(i=0;i<=n-2;i++)
           {
                min=i;
                delay(200);
                for(j=i+1;j<=n-1;j++)
                {
                        if(a[j]<a[min])
                        min=j;
                }
                k=a[i];
                a[i]=a[min];
                a[min]=k;
           }
          end=clock();
          printf("\n\t the sorted list of elements are:\n");
          for(i=0;i<n;i++)
```

```
            printf("\n%d",a[i]);
        printf("\n\n\t time taken:%lf",(end-begin)/CLK_TCK);
        printf("\n\n do u wish to continue (0/1)\n");
        scanf("%d",&ch);
      }
      getch();
  }
```

<u>OUTPUT</u>

```
enter the number of elements
5

 enter the elements to be sorted
8
3
5
1
9

      the sorted list of elements are:
     1      3      5      8      9
      time taken:0.824176
```

Outcome:

To implement the selection sort and time complexity of the algorithm.

OBJECTIVE:

Obtain the Topological ordering of vertices in a given digraph.

Program:

```
#include<stdio.h>
#include<conio.h>
#define max 20

int a[max][max],n;
void topological_sort();
void main()
{
        int i,j;
        clrscr();
        printf("\n enter the number of vertices\n");
        scanf("%d",&n);
        printf("\n enter the adjacency matrix\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        topological_sort();
        getch();
}

void topological_sort()
{
        int v[max],ver[max],i,j,p=1,flag=0;
        for(i=1;i<=n;i++)
        v[i]=0;
        while(p<=n)
        {
                j=1;
                while(j<=n)
                {
                        flag=0;
                        if(v[j]==0)
                        {
```

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

                if((a[i][j]!=0) && (v[i]==0))
                {
                    flag=1;
                    break;
                }
                if(flag==0)
                {
                    v[j]=1;
                    ver[p++]=j;
                    break;
                }
            }
            j++;
            if(j>n)
            {
                printf("\n topological order is not
                     possible\n");
                getch();
                exit(0);
            }
        }
    }
    printf("\n topological order obtained is...\n");
    for(i=1;i<p;i++)
    printf("\t%d",ver[i]);
    getch();
}
```

OUTPUT

```
enter the number of vertices
4

 enter the adjacency matrix
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

 topological order obtained is...
        1       2       4       3
```

Outcome: To understand the implementation of topological sort.

OBJECTIVE: Implement All Pair Shortest paths problem using Floyd's algorithm.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int cost[10][10],a[10][10];
void all_paths(int [10][10],int [10][10],int);
int min1(int,int);

void main()
{
        int i,j,n;
        clrscr();
        printf("\n enter the number of vertices\n");
        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]);
        all_paths(cost,a,n);
        printf("\n\t the shortest path obtained is\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("\t %d",a[i][j]);
                printf("\n");
        }
        getch();
}
void all_paths(int cost[10][10],int a[10][10],int n)
{
        int i,j,k;
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        a[i][j]=cost[i][j];
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        a[i][j]=min1(a[i][j],a[i][k]+a[k][j]);
}
int min1(int a,int b)
{
        return(a<b)?a:b;
}
```

OUTPUT

```
enter the number of vertices
4

 enter the adjacency matrix
999    999    3    999
2      999   999   999
999     7    999    1
 6     999   999   999

         the shortest path obtained is
         10      10      3       4
         2       12      5       6
         7       7       10      1
         6       16      9       10
```

Outcome:

To understand the meaning of All pair shortest path algorithm using Floyd algorithm.

OBJECTIVE: Implement 0/1 Knapsack problem using dynamic programming.

Program:

```
#include<stdio.h>
#include<conio.h>
int v[20][20];
int max1(int a,int b)
{
        return(a>b)?a:b;
}
void main()
{
        int i,j,p[20],w[20],n,max;
        clrscr();
        printf("\n enter the number of items\n");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                printf("\n enter the weight and profit of the
                         item %d:",i);
                scanf("%d %d",&w[i],&p[i]);
        }
        printf("\n enter the capacity of the knapsack");
        scanf("%d",&max);
        for(i=0;i<=n;i++)
        v[i][0]=0;
        for(j=0;j<=max;j++)
        v[0][j]=0;
        for(i=1;i<=n;i++)
        for(j=1;j<=max;j++)
        {
                if(w[i]>j)
                v[i][j]=v[i-1][j];
                else
                v[i][j]=max1(v[i-1][j],v[i-1][j-w[i]]+p[i]);
        }
        printf("\n\nThe table is\n");
        for(i=0;i<=n;i++)
                {
                for(j=0;j<=max;j++)
                printf("%d\t",v[i][j]);
                printf("\n");
        }
        printf("\nThe maximum profit is %d",v[n][max]);
        printf("\nThe most valuable subset is:{");
        j=max;
        for(i=n;i>=1;i--)
        if(v[i][j]!=v[i-1][j])
        {
                printf("\t item %d:",i);
                j=j-w[i];
        }
        printf("}");
        getch();
}
```

OUTPUT

```
enter the number of items
4

 enter the weight and profit of the item 1:2 12

 enter the weight and profit of the item 2:1 10

 enter the weight and profit of the item 3:3 20

 enter the weight and profit of the item 4:2 15

 enter the capacity of the knapsack5


The table is
0       0       0       0       0       0
0       0       12      12      12      12
0       10      12      22      22      22
0       10      12      22      30      32
0       10      15      25      30      37

The maximum profit is 37
The most valuable subset is :{    item 4:         item 2:         item 1 :}
```

Outcome:

To understand the implementation of dynamic programming using 0/1 knapsack problem.

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

Program:

```
#include<stdio.h>
main ()
{
  int n, cost[15][15], i, j, s[15], v, u, w, dist[15],
                      num, min;
  clrscr();
  printf ("Enter the vertices please\n");
  scanf ("%d", &n);
  printf ("Enter the cost of the edges please\n");
  printf ("Enter 999 if the edgeis not present or for the
                      self loop\n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      scanf ("%d", &cost[i][j]);
  printf ("Enter the Source vertex please\n");
  scanf ("%d", &v);

  for (i = 1; i <= n; i++)
    {
      s[i] = 0;
      dist[i] = cost[v][i];
    }

  s[v] = 1;
  dist[v] = 0;

  for (num = 2; num <= n - 1; num++)
    {
      min = 999;
      for (w = 1; w <= n; w++)
            if (s[w] == 0 && dist[w] < min)
              {
                min = dist[w];
                u = w;
              }

      s[u] = 1;

      for (w = 1; w <= n; w++)
            {
              if (s[w] == 0)
                {
                  if (dist[w] > (dist[u] + cost[u][w]))
                        dist[w] = (dist[u] + cost[u][w]);
                }
            }
    }

  printf ("VERTEX\tDESTINATION\tCOST\n");
  for (i = 1; i <= n; i++)
    printf ("    %d\t    %d\t\t %d\n", v, i, dist[i]);
    getch();
}
```

OUTPUT

```
Enter the vertices please
n = 5

Enter the cost of the edges please
Enter 999 if the edge is not present or for the self loop
The cost of the edges are  :

999     1       2       999     999
1       999     3       4       999
2       3       999     5       6
999     4       5       999     6
999     999     6       6       999

Enter the Source vertex please : 1

VERTEX     DESTINATION              COST
   1               1           0
   1               2           1
   1               3           2
   1               4           5
   1               5           8
```

Outcome:

To understand the implantation of Dijkstra's algorithmfor finding the path between source and destination.

Course Name: Design and Analysis of Algorithm.	**Experiment No.** 9	
Course Code : PCS-553 **Faculty :**Mr. Dhajvir Singh Rai	**Branch:** CSE	**Semester:**V

OBJECTIVE: Sort a given set of elements using Quick sort method and determine the time taken 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.

Program:

```
#include<stdio.h>
#include<conio.h>
void quicksort(int[],int,int);
int partition(int[],int,int);
void main()
{
        int i,n,a[20],ch=1;
        clrscr();
        while(ch)
        {
           printf("\n enter the number of elements\n");
           scanf("%d",&n);
           printf("\n enter the array elements\n");
           for(i=0;i<n;i++)
                 scanf("%d",&a[i]);
           quicksort(a,0,n-1);
           printf("\n\nthe sorted array elements are\n\n");
           for(i=0;i<n;i++)
                 printf("\n%d",a[i]);
           printf("\n\n do u wish to continue (0/1)\n");
           scanf("%d",&ch);
        }
        getch();
}

void quicksort(int a[],int low,int high)
{
        int mid;
        if(low<high)
        {
                mid=partition(a,low,high);
                quicksort(a,low,mid-1);
                quicksort(a,mid+1,high);
        }
}
int partition(int a[],int low,int high)
{
        int key,i,j,temp,k;
        key=a[low];
        i=low+1;
        j=high;
        while(i<=j)
        {
                while(i<=high && key>=a[i])
                i=i+1;
                while(key<a[j])
                j=j-1;
                if(i<j)                                         {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
                else
                {
                        k=a[j];
                        a[j]=a[low];
                        a[low]=k;
                }
        }
        return j;
}
```

<u>OUTPUT</u>

```
enter the number of elements
5

 enter the elements to be sorted
8
5
2
4
1
```

```
the sorted list of elements are:
1      2      4      5      8
time taken:0.824176
```

Outcome :

To understand the implementation of quick sort and time complexity of the algorithm.

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

Program:

```
#include<stdio.h>
#include<conio.h>
int root[10], flag = 0, count=0, temp, min;
int a[20], cost[20][20], n, i, j, k, totalcost = 0, x, y;
void find_min (), check_cycle (), update ();
main ()
{
  clrscr();
  printf ("Enter the number of vertices please\n");
  scanf ("%d", &n);
  printf ("Enter the cost of the matrix please\n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      scanf ("%d", &cost[i][j]);
  find_min ();
  while (min != 999 && count != n - 1)
    {
     check_cycle ();
      if (flag)
            {
              printf ("%d  --->  %d  =  %d\n", x, y,
                                   cost[x][y]);
              totalcost += cost[x][y];
              update ();
              count++;
            }
      cost[x][y] = cost[y][x] = 999;
      find_min ();
    }

  if (count < n - 2)
    printf ("The graph is not connected\n");
  else
```

```
    printf ("The graph is connected & the min cost is
                        %d\n", totalcost);
     getch();
}

void check_cycle ()
{
  if ((root[x] == root[y]) && (root[x] != 0))
    flag = 0;
  else
    flag = 1;
}

void find_min ()
{
  min = 999;
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      if (min > cost[i][j])
            {
              min = cost[i][j];
              x = i;
              y = j;
            }
}


void update ()
{

  if (root[x] == 0 && root[y] == 0)
    root[x] = root[y] = x;

  else if (root[x] == 0)
    root[x] = root[y];

  else if (root[y] == 0)
    root[y] = root[x];

  else
    {
      temp = root[y];
      for (i = 1; i <= n; i++)
            if (root[i] == temp)
              root[i] = root[x];
    }
```

OUTPUT

```
Enter the number of vertices please
4
Enter the cost of the matrix please
999   1    5    2
 1    999  999  999
 5    999  999   3
 2    999   3   999
1  ---> 2  =  1
1  ---> 4  =  2
3  ---> 4  =  3
The graph is connected & the min cost is 6
```

Outcome:

To understand the implementation of kruskal’s algorithm for finding the minimum spanning tree.