# ALGORITHMS LAB

Subject Code: CS4L01 Credit: 0-0-3-0-1.5

Course Objectives:

1. To understand different algorithms for searching, sorting and graph problems and analysis of the same.
2. To learn how to analyze the performance of algorithms practically.
3. To understand the various algorithmic design techniques.

Divide And Conquer:

1. Sort a given set of elements using Merge 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 and also compute space complexity.

```
#include<stdio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int main()
{
 int arr[30];
 int i,size;
 printf("\n\t------- Merge sorting method -------\n\n");
 printf("Enter total no. of elements : ");
 scanf("%d",&size);
 for(i=0; i<size; i++)
 {
   printf("Enter %d element : ",i+1);
   scanf("%d",&arr[i]);
 }
 part(arr,0,size-1);
 printf("\n\t------- Merge sorted elements -------\n\n");
 for(i=0; i<size; i++)
 printf("%d ",arr[i]);
 return 0;
}


void part(int arr[],int min,int max)
{
 int mid;
 if(min<max)
 {
   mid=(min+max)/2;
   part(arr,min,mid);
```

```
   part(arr,mid+1,max);
   merge(arr,min,mid,max);
 }
}


void merge(int arr[],int min,int mid,int max)
{
  int tmp[30];
  int i,j,k,m;
  j=min;
  m=mid+1;
  for(i=min; j<=mid && m<=max ; i++)
  {
    if(arr[j]<=arr[m])
    {
       tmp[i]=arr[j];
       j++;
    }
    else
    {
       tmp[i]=arr[m];
       m++;
    }
  }
  if(j>mid)
  {
    for(k=m; k<=max; k++)
    {
       tmp[i]=arr[k];
       i++;
    }
  }
  else
  {
    for(k=j; k<=mid; k++)
    {
      tmp[i]=arr[k];
      i++;
    }
  }
  for(k=min; k<=max; k++)
    arr[k]=tmp[k];
}


------- Merge sorting method -------
Enter total no. of elements :
5
Enter 5 element :
56
34
```

```
2
8
99

------- Merge sorted elements -------
2  8 34 56 99
```

2. Sort a given set of elements using 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 and also compute space complexity.

```
#include<stdio.h>
void qsort(int arr[20], int fst, int last);
int main()
{
 int arr[30];
 int i,size;
 printf("Enter total no. of the elements : ");
 scanf("%d",&size);
 printf("Enter total %d elements : \n",size);
 for(i=0; i<size; i++)
   scanf("%d",&arr[i]);
 qsort(arr,0,size-1);
 printf("Quick sorted elements are as  : \n");
 for(i=0; i<size; i++)
   printf("%d\t",arr[i]);
 return 0;
}
void qsort(int arr[20], int fst, int last)
{
 int i,j,pivot,tmp;
 if(fst<last)
 {
  pivot=fst;
  i=fst;
  j=last;
```

```
  while(i<j)
  {
    while(arr[i]<=arr[pivot] && i<last)
      i++;
    while(arr[j]>arr[pivot])
      j--;
    if(i<j)
    {
      tmp=arr[i];
      arr[i]=arr[j];
      arr[j]=tmp;
    }
  }
  tmp=arr[pivot];
  arr[pivot]=arr[j];
  arr[j]=tmp;
  qsort(arr,fst,j-1);
  qsort(arr,j+1,last);
 }
}
```

Output:

```
Enter total no. of the elements :
 5
Enter total 5 elements
67
45
32
16
9
Quick sorted elements are as  :
 9    16 32 45 67
```

**Decrease And Conquer :**

3. Order the vertices of a directed acyclic graph using topological sorting.

```
#include <stdio.h>
 int main()
{
   int i,j,k,n,a[10][10],indeg[10],flag[10],count=0;

   printf("Enter the no of vertices:\n");
   scanf("%d",&n);

   printf("Enter the adjacency matrix:\n");
   for(i=0;i<n;i++)
{
      printf("Enter row %d\n",i+1);
      for(j=0;j<n;j++)
         scanf("%d",&a[i][j]);
   }

   for(i=0;i<n;i++)
{
      indeg[i]=0;
      flag[i]=0;
   }

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         indeg[i]=indeg[i]+a[j][i];

   printf("\nThe topological order is:");

   while(count<n)
{
      for(k=0;k<n;k++)
    {
         if((indeg[k]==0) && (flag[k]==0)){
            printf("%d ",(k+1));
            flag [k]=1;
         }

         for(i=0;i<n;i++)
    {
            if(a[i][k]==1)
               indeg[k]--;
         }
      }

      count++;
   }

   return 0;
}
```

## Output:

```
Enter the no of vertices:
4
Enter the adjacency matrix:
Enter row 1
0 1 1 0
Enter row 2
0 0 0 1
Enter row 3
0 0 0 1
Enter row 4
0 0 0 0
The topological order is:1 2 3 4
```

## Transform And Conquer:

4. Sort 'N' number of elements using Heap Sort. 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.

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

Void heapsort(int a[],int n)
{
        inti,temp;
        heapfun(a,n);
        for(i=n-1;i>=0;i--)
        {
        temp=a[0];
        a[0]=a[i];
        a[i]=temp;
        heapfun(a,i);
        }
}

heapfun(int a[],int n)
{
        intp,c,k,key;
        for(k=0;k<n;k++)
          {
        key=a[k];
            c=k;
            p=(c-1)/2;
        while(c>0&&key>a[p])
              {
```

```
        a[c]=a[p];
                c=p;
                p=(c-1)/2;
              }
        a[c]=key;
          }
return;
}

void main()
{
        int a[40],i,n;
        clock_tstart,end;
        float t;
        clrscr();

printf("enter the no of elements\n");
        scanf("%d",&n);

printf("elements are\n");
        for(i=0;i<n;i++)
         {
        a[i]=rand()%100;
         }

        printf("randomly selected elements are\n");
        for(i=0;i<n;i++)
         {
        printf("%d\n",a[i]);
         }
        start=clock();
        delay(100);
        for(i=0;i<n;i++)
         {
        heapsort(a,n-i);
         }
        end=clock();
        t=(end-start)/CLK_TCK;
        printf("sorted array is\n");
        for(i=0;i<n;i++)
         {
        printf("%d\n",a[i]);
         }
        printf("time complexity is %f",t);
        }
```

output

enter the no of elements: 5
elements are
randomly selected elements are
46
30
82
90
56

sorted array is
30
46
56
82
90

time complexity is 0.109890

**Space and Time Tradeoffs:**

5. Implement Horspool algorithm for String Matching.

```
#include<stdio.h>
#include<string.h>
#define MAX 500
int t[MAX];

void shifttable(char p[])
{
int i,j,m;
        m=strlen(p);
        for(i=0;i<MAX;i++)
        t[i]=m;
        for(j=0;j<m-1;j++)
        t[p[j]]=m-1-j;
}

int horspool(char src[],char p[])
{
        int i,k,m,n;
        n=strlen(src);
        m=strlen(p);
        printf("\n length of text =%d",n);
        printf("\n length of pattern =%d",m);
        i=m-1;
        while(i<n)
          {
                k=0;
                while((k<m)&&(p[m-1-k]==src[i-k]))
```

```
                k++;
                if(k==m)
                return(i-m+1);
                else
                i+=t[src[i]];
            }
        return -1;
}

void main()
{
        char src[100],p[100];
        int pos;

printf("enter the text which pattern is to be searched:\n");
        gets(src);

printf("enter the pattern to be searched:\n");
        gets(p);

shifttable(p);
        pos = horspool(src,p);
        if(pos>=0)
        printf("\n the desired pattern was found from position %d",pos+1);
        else
        printf("\n the pattern was not found in the given text\n");
}
```

output:
1. enter the text which pattern is to be searched:
dear students check this output
enter the pattern to be searched:
check

length of text =31
length of pattern =5
the desired pattern was found from position 15

2. enter the text which pattern is to be searched:
this is second output
enter the pattern to be searched:
cond

length of text =21
length of pattern =4
the desired pattern was found from position 11

## Greedy Technique:

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

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1,vis[10],min,mincost=0,cost[10][10];

 main()
{
   printf("enter the no of nodes\n");
        scanf("%d",&n);

        printf("\n enter the cost of adjacency matrix\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&cost[i][j]);
        for(i=2;i<=n;i++)
        vis[i]=0;
        printf("\n the edges of the spanning tree are \n");
        vis[1]=1;
        while(ne<n)
          {
                for(i=1,min=999;i<=n;i++)
                        for(j=1;j<=n;j++)
                        if(cost[i][j]<min)
           if(vis[i]==0)
           continue;
           else
                              {
              min=cost[i][j];
                                a=i;
                                b=j;
                        }

      if(vis[a]==0||vis[b]==0)
                              {
                        printf("%d edge (%d%d)=%d\n",ne++,a,b,min);
                        mincost+=min;
                        vis[b]=1;
                              }
         cost[a][b]=cost[b][a]=999;
         }
        printf("mincost=%d\n",mincost);
```

```
        return 0;
}
enter the no of nodes
4

 enter the cost of adjacency matrix
0 9 25 41
999 0 10 999
999 999 0 3
999 999 999 0

 the edges of the spanning tree are
1 edge (12)=9
2 edge (23)=10
3 edge (34)=3
mincost=22
```

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

```
#include<stdio.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);
 main()
{
        printf("\n\tImplementation of Kruskal's algorithm\n");
        printf("\nEnter the no. of vertices:");
        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("The edges of Minimum Cost Spanning Tree are\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("%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);
        return 0;
        }
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;
}
```

```
Enter the no. of vertices:5

Enter the cost adjacency matrix:
0 1 0 9 0
1 0 8 0 6
0 8 0 3 4
9 0 3 0 0
0 6 4 0 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) =1
2 edge (3,4) =3
3 edge (3,5) =4
4 edge (2,5) =6

      Minimum cost = 14
```

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

```
#include<stdio.h>
#define infinity 999

void main()
{
int n,v,i,j,cost[10][10],dist[10],u,count,w,flag[10],min;
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 node:");
scanf("%d",&v);

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];
}

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

}
enter the number of nodes:
4

 enter the cost matrix:
0 9 25 8
9 0 25 2
25 6 0 3
8 2 3 0

 enter the source node:1
 shortest path :1->2,cost=9
1->3,cost=11
1->4,cost=8
```

**Dynamic Programming:**

9. Solve Knapsack problem and print the solution vector.

```
#include<stdio.h>
#include<math.h>
#include<process.h>
#include<stdlib.h>
int max(int a,int b)
{
   if (a>b)
      return a;
   else
      return b;
}
 main()
{
int m,n,p[10],w[10],i,j,k[10],v[20][20];


printf("enter the no. of objects");
scanf("%d",&n);

printf("enter the weight");
for(i=1;i<=n;i++)
scanf("%d",&w[i]);

printf("enter the profit");
for(i=1;i<=n;i++)
scanf("%d",&p[i]);

printf("enter capacity");
scanf("%d",&m);

for(i=0;i<=n;i++)
  {
for(j=0;j<=m;j++)
   {
if(i==0||j==0)
         v[i][j]=0;
else
         if(j<w[i])
```

```
        v[i][j]=v[i-1][j];
        else
            v[i][j]=max(v[i-1][j],p[i]+v[i-1][j-w[i]]);
    }
   }

printf("knapsack filling\n");
for(i=0;i<=n;i++)
  {
for(j=0;j<=m;j++)
    {
printf("%d ",v[i][j]);
    }
printf("\n");
  }
for(i=0;i<=n;i++)
k[i]=0;
i=n;
j=m;
while(i!=0 && j!=0)
{
if(v[i][j]!=v[i-1][j])
          {
         k[i]=1;
   j=j-w[i];
          }
i--;
}


printf("\n the optimal solution is %d\n ",v[n][m]);


printf("solution vector is\n");
for(i=1;i<=n;i++)
printf("%d ",k[i]);
return 0;
}
```

output

enter the no. of objects 4
enter the weight 2 1 3 2
enter the profit 12 10 20 15

enter capacity 5

knapsack filling
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 optimal solution is 37solution vector is: 1101

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,j,k,a[20][20],n;
printf("\nEnter the value of n ");
scanf("%d",&n);
printf("\nEnter the adjacency matrix ");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
scanf("%d",&a[i][j]);
}
}
for(k=0;k<n;k++)
{
for(j=0;j<n;j++)
{
for(i=0;i<n;i++)
{
a[i][j]=a[i][j]<(a[i][k]+a[k][j])?a[i][j]:(a[i][k]+a[k][j]);
}
}
}
printf("\nFloyd's shortest path is ");
```

```
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("%c%d",j==0?'\n':' ',a[i][j]);
}
}
}
```

Output:

Enter the value of n

4

Enter the adjacency matrix

0 3 999 5

2 0 999 4

0 1 0 999

999 999 2 0

Floyd's shortest path is

0 3 7 5

2 0 6 4

3 1 0 5

5 3 2 0

**Back Tracking:**

11. Implement N Queen's algorithm.

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

int board[20],count;

int main()
{
 int n,i,j;
 void queen(int row,int n);
```

```
 printf(" - N Queens Problem Using Backtracking -");
 printf("\n\nEnter number of Queens:");
 scanf("%d",&n);
 queen(1,n);
 return 0;
}

//function for printing the solution
void print(int n)
{
 int i,j;
 printf("\n\nSolution %d:\n\n",++count);

 for(i=1;i<=n;++i)  // to print numbers above table
  printf("\t%d",i);

 for(i=1;i<=n;++i)
 {
  printf("\n\n%d",i);// to print numbers columnwise
  for(j=1;j<=n;++j) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
 }
}

/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
 int i;
 for(i=1;i<=row-1;++i)
 {
  //checking column and digonal conflicts
  if(board[i]==column)  // diagonal conflict checking i.e. (1,1) (2,2)(3,3)(4,4) …
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))   // eg (2,1) (1,2) compute (abs (1-2)) = abs(2-1)) so can't be
//placed
     return 0;
 }

 return 1; //no conflicts
}

//function to check for proper positioning of queen
void queen(int row,int n)
```

```
{
 int column;
 for(column=1;column<=n;++column)
 {
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end i.e. all the queens are placed
    print(n); //printing the board configuration
   else //try queen with next position
    queen(row+1,n);
  }
 }
}
```

```
Enter number of Queens:4
Solution 1:

     1   2   3   4

1    -   Q   -   -

2    -   -   -   Q

3    Q   -   -   -

4    -   -   Q   -

Solution 2:

     1   2   3   4

1    -   -   Q   -

2    Q   -   -   -

3    -   -   -   Q

4    -   Q   -   -
```

12. Find a subset of a given set S of N positive integers whose sum is equal to a given positive integer

```
#include<stdio.h>
int s[10] , x[10],d ;
void sumofsub ( int , int , int ) ;
void main ()
{
int n , sum = 0 ;
int i ;
```

```
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 ) ;
}

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 : 10

 Enter the set in increasing order:
1 2 3 4 5 6 7 8 9 10

 Enter the value of d :
 11
Subset: 1      2      3      5
Subset: 1      2      8
Subset: 1      3      7
Subset: 1      4      6
Subset: 2      3      6
Subset: 2      4      5
```

Subset: 2 9
Subset: 3 8
Subset: 4 7
Subset: 5 6
Subset: 11

**Course Outcomes:**

1. Know the big O, omega, and theta notations and their usage to give asymptotic upper, lower,
and tight bounds on time and space complexity of algorithms.
2. To apply classical sorting, searching, optimization and graph algorithms.
3. To understand basic techniques for designing algorithms, including the techniques of
recursion, divide-and-conquer, and greedy.