CS2432 - ALGORITHM LABORATORY

LAB MANUAL

Department of Computer Science and Engineering

HINDUSTAN UNIVERSITY

HINDUSTAN INSTITUTE OF TECHNOLOGY & SCIENCE

INDEX

I LAB SYLLABUS

L	T	P	C
0	0	3	2

II LAB PLAN

III PSEUDO CODES FOR LAB EXPERIMENTS

CS2432	ALGORITHMS LABORATORY
Goal	To implement different algorithmic techniques and analyze an efficiency of algorithms.

Objectives	Outcomes
This course should enable the students to understand concepts learned in "CS2404 Design and Analysis of Algorithm".	The students should be able to implement various algorithms and demonstrate their 1. complexities..

LIST OF EXPERIMENTS (Using C++)

1. Write a program in C to implement Binary Search using Divide and Conquer Method.

2. Write a program in C to implement MaxMin Problem using Divide and Conquer Method

3. Write a program in C to implement merge sort using Divide and Conquer Method

4. Write a program in C to implement all pairs shortest path using dynamic programming.

5. Write a program in C to travelling salesman problem using dynamic programming.

6. Write a program in C to solve Knapsack Problem using Greedy Method

7. Write a program in C to implement the following traversal techniques.
    a. Depth First Search
    b. Breadth First Search
8. Write a program in C to solve 8-Queens Problem using Backtracking.

LAB PLAN

SI.NO	NAME OF THE EXPERIMENT
1	BINARY SEARCH USING DIVIDE AND CONQUER METHOD
2	MAXMIN PROBLEM USING DEVIDE AND CONQUER METHOD
3	MERGE SORT USING DIVIDE AND CONQUER METHOD
4	ALL PAIR SHORTEST PATH USING DYNAMIC PROGRAMMING
5	TRAVELLING SALESMAN PROBLEM USING DYNAMIC PROGRAMMING
6	TRAVERSAL TECHNIQUES:DEPTH FIRST SEARCH
7	TRAVERSAL TECHNIQUES:BREADTH FIRST SEARCH
8	KNAPSACK PROBLEM USING GREEDY METHOD
9	8-QUEENS PROBLEM USING BACKTRACING

1 BINARY SEARCH USING DIVIDE AND CONQUER METHOD

AIM:

Write a program in C to implement Binary Search using Divide and Conquer Method

ALGORITHM:

Step1:Start
Step2:Declare the variables n,a[30],i,item,mid,top,bottom
Step3:Get the values of n,a[30]and the item to be searched.
Step4:Declare top=nand bottom=1
Step5:In do while loop get the value mid=top+bottom/2
Step6:if(item>a[mid]) increment mid else decrement mid.
Step7:Continue the do while loop until the condition item !=a[mid] and bottom<=top.
Step8:Then print the position of the search key.
Step9:And also use the heder time,h to find the time spent at different executions.
Step10:Stop.

PROGRAM:

```
#include <stdio.h>

int main()
{
  int c, first, last, middle, n, search, array[100];

  printf("Enter number of elements\n");
  scanf("%d",&n);

  printf("Enter %d integers\n", n);

  for ( c = 0 ; c < n ; c++ )
    scanf("%d",&array[c]);

  printf("Enter value to find\n");
  scanf("%d",&search);

  first = 0;
  last = n - 1;
  middle = (first+last)/2;

  while( first <= last )
  {
    if ( array[middle] < search )
      first = middle + 1;
    else if ( array[middle] == search )
    {
      printf("%d found at location %d.\n", search, middle+1);
      break;
    }
    else
      last = middle - 1;

    middle = (first + last)/2;
  }
  if ( first > last )
    printf("Not found! %d is not present in the list.\n", search);
```

```
  return 0;
}
```

2. MAXMIN PROBLEM USING DEVIDE AND CONQUER METHOD

AIM:

Write a program in C to implement MaxMin Problem using Divide and Conquer Method

ALGORITHM:

1. Start
2. Start th clock cycle to calculate time complixity.
3. Enter the size of the array then enter element.
4. pass the argument to maxmin function
5. It calculates and stores the max and min element in *max, and *min respectively.
6. In main () print the maximum and minimum element
7. End clock..
8. calculate time complixity and print it
9. stop.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
maxmin(int,int,int,int);
int num[20],max,min,max1,min1;
main()
{ int n,i;
printf("enter number of elements in the array\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter the number\n");
scanf("%d",&num[i]);
}
```

```
maxmin(0,n-1,max,min);
printf("the maximum element is- %d\n",max);
printf("the minimum element is- %d\n",min);
getch();
}
maxmin(int i,int j,int max,int min)
{
if(i=j) max=min=num[i];
else if(j=i+1) { if(num[i]<num[j]) { max=num[j]; min=num[i];}
else if(num[i]>num[j]){max=num[i]; min=num[j];}
else max=min=num[i];
}
else
{
int mid=(i+j)/2;
maxmin(i,mid,max,min);
maxmin(mid+1,j,max1,min1);
}
```

3.MERGE SORT USING DIVIDE AND CONQUER METHOD

AIM:

Write a program in C to implement merge sort using Divide and Conquer Method

ALGORITHM:

Step 1: Start the process.
Step 2: Declare the variables.
Step 3: Enter the list of elements to be sorted using the get function.
Step 4: Divide the array list into two halves the lower array list and upper array list using the merge sort function.
Step 5: Sort the two array list.
Step 6: Combine the two sorted arrays.
Step 7: Display the sorted elements using the get () function.
Step 8: Stop the process.

PROGRAM:

```
#include<stdio.h>
#include<conio.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]);
 getch();
 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];
}
```

4. ALL PAIR SHORTEST PATH USING DYNAMIC PROGRAMMING

AIM:

Write a program in C to implement all pairs shortest path using dynamic programming.

ALGORITHM:

Step1: Start the program.
Step2: Declare the variables.
Step3: Using the get function get the number of vertices and enter their weights.
Step4: Using the cal function calculate the shortest path
Step5: Display the shortest path distance graph.
Step6: End the program.

PROGRAM:

```
#include<stdio.h>
#include<time.h>
# define size 10
int a[size][size];
int i,j,n;
void floyd(int x[size][size],int y)
{ int k,i,j;
for(k=0;k<n;k++)
{ for(i=0;i<n;i++)
{ for(j=0;j<n;j++)
{
if(x[i][j]>(x[i][k]+x[k][j])) x
[i][j]=(x[i][k]+x[k][j]);
}
}
}
}
void main()
{
```

```
clock_t beg,end;
double timespent;
beg=clock();
printf("Enter the no of Vertices:");
scanf("%d",&n);
printf("Give the initial weighted graph in weight natrix form:\n") ; for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
printf("Enter the value of a [%d][%d]:",i,j);
scanf("%d",&a[i][j]);
}
}
printf("The input Weight matrix is :\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
if(a[i][j]==9999)
printf("inft");
else
printf("%d\t",a[i][j]);
}
printf("\n");
}
floyd(a,n);
printf("\n Final matrix where we can find shortest dist:"); for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
printf("%5d",a[i][j]); printf("\n");
}
end=clock();
timespent=(double)(end-beg)/CLOCKS_PER_SEC; printf("%f\n",timespent);
}
```

5. TRAVELLING SALESMAN PROBLEM USING DYNAMIC PROGRAMMING

AIM:

Write a program in C to travelling salesman problem using dynamic programming.

ALGORITHM:

Step1: Start the process
Step2: Enter the number of cities
Step3: Enter the cost matrix of all the cities
Step4: Find all possible feasible solutions by taking the permutation of the cities which is to be covered.
Step5: Find the cost of each path using the cost matrix.
Step6: Find out the path with minimum cost.
Step7: If more than one path having the same cost considers the first occurring path.
Step8: That is selected as the optimum solution.
Step9: stop the process.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
int main()
{
 int cost[20][20],min,l,m,sr[20],sc[20],flag[20][20],i,j,k,rf[20],cf[20],n;
 int nrz[20],ncz[20],cn,a,noz,nrz1[20],ncz1[20],counter =0;

 printf("\n\tC PROGRAM FOR TRAVELLING SALESMAN PROBLEM");
 printf("\n\nEnter the total number of assignments:");
 scanf("%d",&n);
 /* Enter the cost matrix*/
 printf("\nEnter the cost matrix\n");
 for(i=0;i<n;i++)
 {
   printf("\n");
   for(j=0;j<n;j++)
   {
     printf("cost[%d][%d] = ",i,j);
     scanf("%d",&cost[i][j]);
    }
 }
 printf("\n\n");

 printf("Cost matrix:\n");
 for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
    printf("\t%d\t",cost[i][j]);
    printf("\n");
 }

 for(i=0;i<n;i++)
 {
  min=cost[i][0];
  /* find the minmum element in each row*/
  for(j=0;j<n;j++)
  {
    if(min>cost[i][j])
    min=cost[i][j];
  }

  for(j=0;j<n;j++)
   cost[i][j]=cost[i][j]-min;
 }

 for(i=0;i<n;i++)
 {
  min=cost[0][i];
  /* find the minimum element in each column*/
  for(j=0;j<n;j++)
  {
   if(min>cost[j][i])
```

```
    min=cost[j][i];
  }
     for(j=0;j<n;j++)
   cost[j][i]=cost[j][i]-min;
}
printf("\n\n");
printf("Cost matrix after row & column operation:\n");
for(i=0;i<n;i++)
{ for(j=0;j<n;j++)
   printf("\t%d\t",cost[i][j]);
  printf("\n");
}
repeatx:;

a=0;noz=0,min=1000;
for(i=0;i<n;i++)
{ for(j=0;j<n;j++)
   flag[i][j]=0;
}
for(i=0;i<n;i++)
{ cn=0;
  for(j=0;j<n;j++)
  { if(cost[i][j]==0)
    { cn++;
  flag[i][j]=1;
    }
  }
  nrz[i]=cn;
  noz=noz+cn;
}
for(i=0;i<n;i++)
{ cn=0;
  for(j=0;j<n;j++)
  { if(cost[j][i]==0)
    { cn++;
  flag[j][i]=1;
    }
  }
  ncz[i]=cn;
  noz=noz+cn;
}
for(i=0;i<n;i++)
{ nrz1[i]=nrz[i];
  ncz1[i]=ncz[i];
}
k=0;
while(nrz[k]!=0||ncz[k]!=0)
{
for(i=0;i<n;i++)
 { cn=0;
  for(j=0;j<n;j++)
  { if(flag[i][j]==1)
     cn++;
    nrz[i]=cn;
  }
  if(nrz[i]==1)
```

```
  { for(j=0;j<n;j++)
    { if(flag[i][j]==1)
  { flag[i][j]=2;
    for(k=0;k<n;k++)
    { if(flag[k][j]==1)
      flag[k][j]=0;
    }
  }
    }
  }
}
for(i=0;i<n;i++)
{ cn=0;
  for(j=0;j<n;j++)
  { if(flag[j][i]==1)
     cn++;
    ncz[i]=cn;
  }
  if(ncz[i]==1)
  { for(j=0;j<n;j++)
    { if(flag[j][i]==1)
  { flag[j][i]=2;
    for(k=0;k<n;k++)
    { if(flag[j][k]==1)
      flag[j][k]=0;
    }
  }
    }
  }
}
k++;
}
 for(i=0;i<n;i++)
 { for(j=0;j<n;j++)
   { if(flag[i][j]==2)
   a++;
   }
 }


 if(a==n)
 {
   printf("\nAssignments completed in order!!\n");
   /* Display the order in which assignments will be completed*/
   for(i=0;i<n;i++)
   { for(j=0;j<n;j++)
     { if(flag[i][j]==2)
     printf(" %d->%d ",i+1,j+1);
     }
     printf("\n");
   }
   getch();
   exit(0);
 }
```

```
else
{ for(i=0;i<n;i++)
  { rf[i]=0,sr[i]=0;
    cf[i]=0,sc[i]=0;
  }
  for(k=n;(k>0&&noz!=0);k--)
  { for(i=0;i<n;i++)
    { m=0;
  for(j=0;j<n;j++)
  { if((flag[i][j]==4)&&(cost[i][j]==0))
    m++;
  }
     sr[i]=m;
    }
    for(i=0;i<n;i++)
    { if(nrz1[i]==k&&nrz1[i]!=sr[i])
  {  rf[i]=1;
     for(j=0;j<n;j++)
     { if(cost[i][j]==0)
         flag[i][j]=4;
     }
     noz=noz-k;
  }
    }
    for(i=0;i<n;i++)
    {
    l=0;
    for(j=0;j<n;j++)
    { if((flag[j][i]==4)&&(cost[j][i]==0))
     l++;
    }
      sc[i]=l;
    }
    for(i=0;i<n;i++)
    { if(ncz1[i]==k&&ncz1[i]!=sc[i])
  {  cf[i]=1;
     for(j=0;j<n;j++)
     { if(cost[j][i]==0)
         flag[j][i]=4;
     }
     noz=noz-k;
  }
    }
    for(i=0;i<n;i++)
    { for(j=0;j<n;j++)
  { if(flag[i][j]!=3)
    { if(rf[i]==1&&cf[j]==1)
      {  flag[i][j]=3;
        if(cost[i][j]==0)
       noz=noz+1;
      }
    }
  }
    }
  }
  for(i=0;i<n;i++)
```

```
   { for(j=0;j<n;j++)
     { if(rf[i]!=1&&cf[j]!=1)
   { if(min>cost[i][j])
      min=cost[i][j];
   }
     }
   }
   for(i=0;i<n;i++)
   { for(j=0;j<n;j++)
     { if(rf[i]!=1&&cf[j]!=1)
    cost[i][j]=cost[i][j]-min;
     }
   }
   for(i=0;i<n;i++)
   { for(j=0;j<n;j++)
     { if(flag[i][j]==3)
    cost[i][j]=cost[i][j]+min;
     }
   }
 }
 printf("\n\n");
 if (counter < 10)
 {
  counter = counter+1;
 printf("\n\nIntermediate Matrix: \n");
 for(i=0;i<n;i++)
 {
   for(j=0;j<n;j++)
   printf("\t%d\t",cost[i][j]);
   printf("\n");
 }
 }
 else
 {
     printf("\n\nOptimal solution to given problem is not possible");
     getch();
     return 0;
     }
 goto repeatx;
}
```

6. TRAVERSAL TECHNIQUES:DEPTH FIRST SEARCH

AIM:

Write a program in C to implement the following traversal techniques Depth First Search

ALGORITHM:

1. Start
2. Declare the matrix,i ,j,n.
3. Declare the function dfs.

4. in the function dfs if there is an unvisited node x the visit(x)
5. pop the values which are visited until the stack is not empty.
6. if the visited nodes have unvisited neighbours visit y.
7. push the non visited neighbours int the stack.
8. in function main get the input of the matrix
9. include the time spent calculation
10. stop

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<time.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
int i;
reach[v]=1;
for(i=1;i<=n;i++)
if (a[v][i] && !reach[i])
{
printf("\n %d->%d",v,i);
dfs(i);
}
}
void main()
{
int i,j,count=0;
clock_t beg,end;
double timespent;
beg=clock();
clrscr();
printf("\nEnter no of vertices:");
scanf("%d",&n);
for(i=1;i<=n;i++)
 {
 reach[i]=0;
 for(j=1;j<=n;j++)
 a[i][j]=0;
 }
 printf("\n Enter the adjacency matrix:\n");
 for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
}
dfs(1);
printf("\n");
for(i=1;i<=n;i++)
{
if(reach[i])
count++;
}
```

```
if(count==n)
printf("\nGraph is connected");
else
printf("\nGraph is not connected");
getch();

end=clock();
timespent=(double)(end-beg)/CLOCKS_PER_SEC;
printf(" The time spent %f\n",timespent);
}
```

7. TRAVERSAL TECHNIQUES:BREATH FIRST SEARCH

AIM:

Write a program in C to implement the following traversal techniques Depth First Search

ALGORITHM:

1. Start
2. Declare the matrix,i ,j,n.
3. Declare the function bfs..
4. in the function bfs if there is an unvisited node x the visit(x)
5. pop the values which are visited until the queue is not empty.
6. if the visited nodes have unvisited neighbours visit y.
7. push the non visited neighbours int the queue.
8. in function main get the input of the matrix
9. include the time
10. stop

PROGRAM:

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

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
for(i=1;i<=n;i++)
if(a[v][i]&&!visited[i])
q[++r]=i;
if(f<=r)
{
visited[q[f]]=1;
bfs(q[f++]);
}
}
void main()
{
```

```
int v;
clock_t beg,end;
double timespent;
beg=clock();

clrscr();
printf("\nEnter the no of vertices: ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
q[i]=0;
visited[i]=0;
}
printf("\n Enter graph data in matrix form :\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
}
printf("\nEnter the starting vertices: ");
scanf("%d",&v);
bfs(v);
printf("\nThe node which are reachable are :\n");
for(i=1;i<=n;i++)
{
if(visited[i])
printf("%d\t",i);
else
printf("\nBfs is not possible");
}
getch();
end=clock();
timespent=(double)(end-beg)/CLOCKS_PER_SEC;
printf(" The time spent %f\n",timespent);
}
```

8. KNAPSACK PROBLEM USING GREEDY METHOD

AIM:

Write a program in C to solve Knapsack Problem using Greedy Method

ALGORITHM:

Step1: Start the program.
Step2: Declare the variable.
Step3: Using the get function read the number of items, capacity of the bag,
Weight of the item and value of the items.
Step4: Find the small weight with high value using the find function.

Step5: Find the optimal solution using the function findop ().
Step6: Display the optimal solution for the items.
Step7: Stop the process.

PROGRAM:

```
#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("\nEnter 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();
}
```

9. 8-QUEENS PROBLEM USING BACKTRACKING

AIM:

Write a program in C to solve 8-Queens Problem using Backtracking.

ALGORITHM:

Step1: Start the process.
Step2: Enter the no. of queens.
Step3: Using the queen function place all the queens.
Step4: Check the position to place the queen is free or not.
Step5: Check finally if all the queens are placed.
Step6: Check for all the constraints, check the rows and columns
Step7: If no clash then all the queens are placed in correct position.
Step8: stop.

PROGRAM:

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

int board[20];
int count;
void main()
{
int n,i,j;
void queen(int row, int n);
clrscr();
printf("\n\t Program for queen's using backtracking");
printf("Enter number of queen's ");
scanf("%d",&n);
queen(1,n);                          //trace using backtrack
getch();
 }
```

```
void print_board(int n)
{
int i,j;
printf("\n\n solution %d:\n\n",++count);
//number of solution
for(i=1;i<=n;i++)
{
printf("\t%d",i);
}
for(i=1;i<=n;i++)
{
printf("\n\n%d",i);
for(j=1;j<=n;j++)       //for n*n board
{
if(board[i]==j)
printf("\tQ");   //Queen at i,j position
else
printf("\t-");   //empty slot
}
}
printf("\n Press any key to continue.....");
getch();
}

int place(int row ,int column)
{
int i;
for(i=1;i<=row-1;i++)
{
//checking for column and diagonal conflicts
if(board[i]==column)
return 0;
else
if(abs(board[i]-column)==abs(i-row))
return 0;
}
//no conflicts hence Queen can be placed
return 1;
}


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
print_board(n);    //printing the board configuration
else                    //try queen with next position
queen(row+1,n);
}
```

```
}
}
```