Experinment-1a: Develop a program and measure the running time for Binary Search with Divide and Conquer

Aim: To develop a program and measure the running time for Binary Search with Divide and Conquer

Description:

Source Code:

```
#include<stdio.h>
int binary_search(int A[], int key, int len) {
  int low = 0;
  int high = len -1;
  while (low <= high) {
    int mid = low + ((high - low) / 2);
    if (A[mid] == key) {
      return mid; }
    if (key < A[mid]) {
      high = mid - 1;}
    else {
      low = mid + 1;} }
  return -1;}
int main() {
  int a[10]={1,3,5,7,9,11,13,15,17,21};
  int key = 3;
  int position = binary_search(a, key, 10);
  if (position == -1){
    printf("Not found");
    return 0; }
  printf("Found it at %d", position);
  return 0;}
```

Output:

```
Found it at 1
```

Running time:

Experinment-2: Develop a program and measure the running time for Merge Sort with Divide and Conquer

Aim: To Develop a program and measure the running time for Merge Sort with Divide and Conquer
Description:

Source Code:

```
#include <stdio.h>
void merge(int arr[], int l, int m, int r)
{
   int i, j, k;
   int n1 = m - l + 1;
   int n2 =  r - m;
   int L[n1], R[n2];
   for (i = 0; i < n1; i++)
      L[i] = arr[l + i];
   for (j = 0; j < n2; j++)
      R[j] = arr[m + 1+ j];
   i = 0; // Initial index of first subarray
   j = 0; // Initial index of second subarray
   k = l; // Initial index of merged subarray
   while (i < n1 && j < n2)
   {  if (L[i] <= R[j]) {
         arr[k] = L[i];
         i++; }
      else
      { arr[k] = R[j];
         j++; }
      k++; }
    while (i < n1)
   { arr[k] = L[i];
      i++;
      k++; }
   while (j < n2)
   { arr[k] = R[j];
      j++;
      k++; }}
void mergeSort(int arr[], int l, int r)
{if (l < r) {
      int m = l+(r-l)/2;
      mergeSort(arr, l, m);
      mergeSort(arr, m+1, r);
      merge(arr, l, m, r); } }
void printArray(int A[], int size)
{int i;
   for (i=0; i < size; i++)
      printf("%d ", A[i]);
   printf("\n"); }
int main()
{
```

```
    int arr[] = {70, 50, 30, 10, 20, 40,60};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    printf("Given array is \n");
    printArray(arr, arr_size);
    mergeSort(arr, 0, arr_size - 1);
    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0; }
```

Output:

Given array is

70 50 30 10 20 40
60

Sorted array is

10 20 30 40 50 60
70

Running time:

Experinment- 3: . Develop a program and measure the running time for Quick Sort with Divide and Conquer

Aim: To . Develop a program and measure the running time for Quick Sort with Divide and Conquer

Description:

Source Code:

```
#include <stdio.h>
void swap(int* a, int* b)
{int t = *a;
        *a = *b;
        *b = t;}
int partition(int arr[], int low, int high)
{       int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
                if (arr[j] < pivot) {i++;
                        swap(&arr[i], &arr[j]); }}
        swap(&arr[i + 1], &arr[high]);
        return (i + 1);}
void quickSort(int arr[], int low, int high)
{if (low < high) {
                int pi = partition(arr, low, high);
                quickSort(arr, low, pi - 1);
                quickSort(arr, pi + 1, high); }}
int main()
{int arr[] = { 10, 7, 8, 9, 1, 5 };
        int N = sizeof(arr) / sizeof(arr[0]);
        quickSort(arr, 0, N - 1);
        printf("Sorted array: \n");
        for (int i = 0; i < N; i++)
                printf("%d ", arr[i]);
        return 0;}
```

Output:

Sorted array:

1 5 7 8 9 10

Running time:

Experinment-4: Develop a program and measure the running time for estimating minimum-cost spanning Trees with Greedy Method

Aim: To develop a program and measure the running time for estimating minimum-cost spanning Trees with Greedy Method.

Description:

Source Code:

Prim's Algorithm:

```
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
#define V 5
int minKey(int key[], bool mstSet[])
{       int min = INT_MAX, min_index;
for (int v = 0; v < V; v++)
        if (mstSet[v] == false && key[v] < min)
                min = key[v], min_index = v;
return min_index;}
int printMST(int parent[], int graph[V][V])
{       printf("Edge \tWeight\n");
for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i,
                graph[i][parent[i]]);}
void primMST(int graph[V][V])
{       int parent[V];
int key[V];
bool mstSet[V];
for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;
key[0] = 0;
parent[0] = -1;
for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet);
        mstSet[u] = true;
        for (int v = 0; v < V; v++)
                if (graph[u][v] && mstSet[v] == false
                        && graph[u][v] < key[v])
                        parent[v] = u, key[v] = graph[u][v];        }
printMST(parent, graph);}
int main()
{       int graph[V][V] = { { 0, 2, 0, 6, 0 },
                                        { 2, 0, 3, 8, 5 },
                                        { 0, 3, 0, 0, 7 },
                                        { 6, 8, 0, 0, 9 },
                                        { 0, 5, 7, 9, 0 } };
primMST(graph);
return 0;}
```

Output:

Edge Weight

0 - 1 2

1 - 2 3

0 - 3 6

1 - 4 5

Running time:

Kruskal's
Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
int comparator(const void* p1, const void* p2)
{
const int(*x)[3] = p1;
const int(*y)[3] = p2;
return (*x)[2] - (*y)[2];}
void makeSet(int parent[], int rank[], int n)
{
for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;       }}
int findParent(int parent[], int component)
{
if (parent[component] == component)
        return component;
return parent[component]= findParent(parent, parent[component]);}
void unionSet(int u, int v, int parent[], int rank[], int n)
{       u = findParent(parent, u);
v = findParent(parent, v);
if (rank[u] < rank[v]) {
        parent[u] = v;}
else if (rank[u] > rank[v]) {
        parent[v] = u;}
else {parent[v] = u;
        rank[u]++;       }}
void kruskalAlgo(int n, int edge[n][3])
{       qsort(edge, n, sizeof(edge[0]), comparator);
int parent[n];
int rank[n];
makeSet(parent, rank, n);
int minCost = 0;
printf("Following are the edges in the constructed MST\n");
for (int i = 0; i < n; i++) {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];

if (v1 != v2) {
                unionSet(v1, v2, parent, rank, n);
```

```
                minCost += wt;
                printf("%d -- %d == %d\n", edge[i][0],
                        edge[i][1], wt); }          }
printf("Minimum Cost Spanning Tree: %d\n", minCost);}
int main()
{
int edge[5][3] = { { 0, 1, 10 },
                                { 0, 2, 6 },
                                { 0, 3, 5 },
                                { 1, 3, 15 },
                                { 2, 3, 4 } };
        kruskalAlgo(5, edge);
return 0;}
```

Output:

Following are the
edges in the
constructed MST

2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost
Spanning Tree: 19

Running time:

Experinment-5: Develop a program and measure the running time for estimating Single Source Shortest Paths with Greedy Method.

Aim: To Develop a program and measure the running time for estimating Single Source Shortest Paths with Greedy Method.

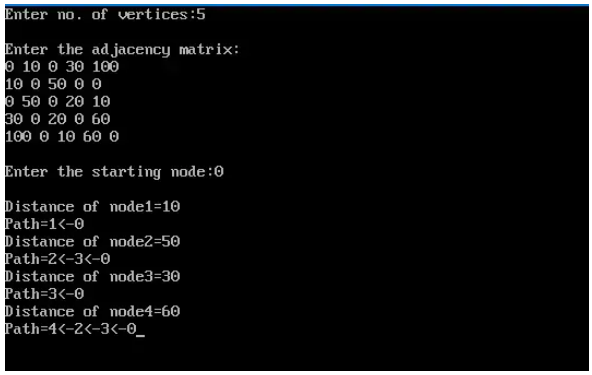
Description:

Source Code:

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10
void dijkstra(int G[MAX][MAX],int n,int startnode);
int main()
{ clrscr();
   int G[MAX][MAX],i,j,n,u;
   printf("Enter no. of vertices:");
   scanf("%d",&n);
   printf("\nEnter the adjacency matrix:\n");
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   scanf("%d",&G[i][j]);
   printf("\nEnter the starting node:");
   scanf("%d",&u);
   dijkstra(G,n,u);
   getch();
   return 0;}
void dijkstra(int G[MAX][MAX],int n,int startnode)
{ int cost[MAX][MAX],distance[MAX],pred[MAX];
   int visited[MAX],count,mindistance,nextnode,i,j;
   for(i=0;i<n;i++)
   for(j=0;j<n;j++)
   { if(G[i][j]==0)
   cost[i][j]=INFINITY;
   else
   cost[i][j]=G[i][j];  }
   for(i=0;i<n;i++)
   {
   distance[i]=cost[startnode][i];
   pred[i]=startnode;
   visited[i]=0; }
   distance[startnode]=0;
   visited[startnode]=1;
   count=1;
   while(count<n-1) {
    mindistance=INFINITY;
   for(i=0;i<n;i++)
   if(distance[i]<mindistance&&!visited[i]) {
   mindistance=distance[i];
```

```
 nextnode=i; }
 visited[nextnode]=1;
 for(i=0;i<n;i++)
 if(!visited[i])
 if(mindistance+cost[nextnode][i]<distance[i]) {
  distance[i]=mindistance+cost[nextnode][i];
  pred[i]=nextnode; }
count++;}
 for(i=0;i<n;i++)
 if(i!=startnode) {
 printf("\nDistance of node%d=%d",i,distance[i]);
 printf("\nPath=%d",i);
 j=i;
 do
 {
   j=pred[j];
   printf("<-%d",j);
 }while(j!=startnode); }}
```

Output:



Running time:

Experinment-6: Develop a program and measure the running time for optimal Binary search trees with Dynamic Programming

Aim: To develop a program and measure the running time for optimal Binary search trees with Dynamic Programming

Description:

Source Code:

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
void main()
{      char ele[MAX][MAX];
int w[MAX][MAX], c[MAX][MAX], r[MAX][MAX], p[MAX], q[MAX];
int temp=0, root, min, min1, n;
int i,j,k,b;
clrscr();
printf("Enter the number of elements:");
scanf("%d",&n);
printf("\n");
for(i=1; i <= n; i++)     {
       printf("Enter the Element of %d:",i);
       scanf("%d",&p[i]);
}
printf("\n");
for(i=0; i <= n; i++){
printf("Enter the Probability of %d:",i);
       scanf("%d",&q[i]); }
printf("W\t\tC\t\tR\n");
for(i=0; i <= n; i++)
{for(j=0; j <= n; j++)
       {if(i == j)
              {w[i][j] = q[i];
                     c[i][j] = 0;
                     r[i][j] = 0;
                     printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]:
%d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]); }}}
printf("\n");
for(b=0; b < n; b++)
{for(i=0,j=b+1; j < n+1 && i < n+1; j++,i++)
       {if(i!=j && i < j)
              {w[i][j] = p[j] + q[j] + w[i][j-1];
                     min = 30000;
                     for(k = i+1; k <= j; k++)
                     {min1 = c[i][k-1] + c[k][j] + w[i][j];
                            if(min > min1)
                            {min = min1;
                                   temp = k;} }
                     c[i][j] = min;
                     r[i][j] = temp;}
```

```
            printf("W[%d][%d]: %d\tC[%d][%d]: %d\tR[%d][%d]:
%d\n",i,j,w[i][j],i,j,c[i][j],i,j,r[i][j]);}
      printf("\n");      }
printf("Minimum cost = %d\n",c[0][n]);
root = r[0][n];
printf("Root  = %d \n",root);
getch();
}
```

Output:

```
Enter the number of elements:6

Enter the Element of 1:10
Enter the Element of 2:3
Enter the Element of 3:9
Enter the Element of 4:2
Enter the Element of 5:0
Enter the Element of 6:10

Enter the Probability of 0:5
Enter the Probability of 1:6
Enter the Probability of 2:4
Enter the Probability of 3:4
Enter the Probability of 4:3
Enter the Probability of 5:8
Enter the Probability of 6:0
W           C           R
W[0][0]: 5  C[0][0]: 0  R[0][0]: 0
W[1][1]: 6  C[1][1]: 0  R[1][1]: 0
W[2][2]: 4  C[2][2]: 0  R[2][2]: 0
W[3][3]: 4  C[3][3]: 0  R[3][3]: 0
W[4][4]: 3  C[4][4]: 0  R[4][4]: 0
W[5][5]: 8  C[5][5]: 0  R[5][5]: 0
W[6][6]: 0  C[6][6]: 0  R[6][6]: 0

W[0][1]: 21 C[0][1]: 21 R[0][1]: 1
W[1][2]: 13 C[1][2]: 13 R[1][2]: 2
W[2][3]: 17 C[2][3]: 17 R[2][3]: 3
W[3][4]: 9  C[3][4]: 9  R[3][4]: 4
W[4][5]: 11 C[4][5]: 11 R[4][5]: 5
W[5][6]: 18 C[5][6]: 18 R[5][6]: 6

W[0][2]: 28 C[0][2]: 41 R[0][2]: 1
W[1][3]: 26 C[1][3]: 39 R[1][3]: 3
W[2][4]: 22 C[2][4]: 31 R[2][4]: 3
W[3][5]: 17 C[3][5]: 26 R[3][5]: 5
W[4][6]: 21 C[4][6]: 32 R[4][6]: 6

W[0][3]: 41 C[0][3]: 79 R[0][3]: 2
W[1][4]: 31 C[1][4]: 53 R[1][4]: 3
W[2][5]: 30 C[2][5]: 56 R[2][5]: 3
W[3][6]: 27 C[3][6]: 53 R[3][6]: 6
```

```
W[0][4]: 46 C[0][4]: 96 R[0][4]: 3
W[1][5]: 39 C[1][5]: 78 R[1][5]: 3
W[2][6]: 40 C[2][6]: 89 R[2][6]: 4

W[0][5]: 54 C[0][5]: 121      R[0][5]: 3
W[1][6]: 49 C[1][6]: 115      R[1][6]: 3

W[0][6]: 64 C[0][6]: 158      R[0][6]: 3

Minimum cost = 158
Root  = 3

Running time:
```

Experinment-7: Develop a program and measure the running time for identifying solution for traveling salesperson problem with Dynamic Programming

Aim: To Develop a program and measure the running time for identifying solution for traveling salesperson problem with Dynamic Programming

Description:

Source Code:

```
#include <stdio.h>
int tsp_g[10][10] = {
  {12, 30, 33, 10, 45},
  {56, 22, 9, 15, 18},
  {29, 13, 8, 5, 12},
  {33, 28, 16, 10, 3},
  {1, 4, 30, 24, 20}
};
int visited[10], n, cost = 0;
void travellingsalesman(int c){
  int k, adj_vertex = 999;
  int min = 999;
  visited[c] = 1;
  printf("%d ", c + 1);
  for(k = 0; k < n; k++) {
    if((tsp_g[c][k] != 0) && (visited[k] == 0)) {
      if(tsp_g[c][k] < min) {
        min = tsp_g[c][k];
      }
      adj_vertex = k;
    }
  }
  if(min != 999) {
    cost = cost + min;
  }
  if(adj_vertex == 999) {
    adj_vertex = 0;
    printf("%d", adj_vertex + 1);
    cost = cost + tsp_g[c][adj_vertex];
    return;
  }
  travellingsalesman(adj_vertex);
}
int main(){
  int i, j;
  n = 5;
  for(i = 0; i < n; i++) {
    visited[i] = 0;
  }
  printf("\n\nShortest Path:\t");
```

```
    travellingsalesman(0);
    printf("\n\nMinimum Cost: \t");
    printf("%d\n", cost);
    return 0;
}
```

Output:

Shortest Path: 1 5 4 3 2 1

Minimum Cost: 99

Running time:

Experinment-8: Develop a program and measure the running time for identifying solution for 8-Queens problem with Backtracking

Aim: To Develop a program and measure the running time for identifying solution for 8-Queens problem with Backtracking

Description:

Source Code:

```
#define N 8
#include <stdbool.h>
#include <stdio.h>
void printSolution(int board[N][N])
{for (int i = 0; i < N; i++) {
		for (int j = 0; j < N; j++) {
			if(board[i][j])
				printf("1 ");
			else
				printf("0 ");
		}
		printf("\n");
	}
}
bool isSafe(int board[N][N], int row, int col)
{
	int i, j;

	// Check this row on left side
	for (i = 0; i < col; i++)
		if (board[row][i])
			return false;

	// Check upper diagonal on left side
	for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
		if (board[i][j])
			return false;

	// Check lower diagonal on left side
	for (i = row, j = col; j >= 0 && i < N; i++, j--)
		if (board[i][j])
			return false;

	return true;
}
bool solveNQUtil(int board[N][N], int col)
{	if (col >= N)
		return true;
	for (int i = 0; i < N; i++) {
```

```
                if (isSafe(board, i, col)) {
                        board[i][col] = 1;
                        if (solveNQUtil(board, col + 1))
                                return true;
                        board[i][col] = 0; // BACKTRACK
                }
        }
        return false;
}
bool solveNQ()
{
        int board[N][N] = { { 0, 0, 0, 0, 0, 0, 0, 0},
                                                { 0, 0, 0, 0, 0, 0, 0, 0},
                                                { 0, 0, 0, 0, 0, 0, 0, 0},
                                                { 0, 0, 0, 0, 0, 0, 0, 0},
                                                { 0, 0, 0, 0, 0, 0, 0, 0} ,
                                                { 0, 0, 0, 0, 0, 0, 0, 0} ,
                                                 { 0, 0, 0, 0, 0, 0, 0, 0} ,
                                                { 0, 0, 0, 0, 0, 0, 0, 0}  };

        if (solveNQUtil(board, 0) == false) {
                printf("Solution does not exist");
                return false;
        }

        printSolution(board);
        return true;
}
int main()
{
        solveNQ();
        return 0;
}
```

Output:

```
 1,0,0,0,0,0,0,0
 0,0,0,0,0,0,1,0
0,0,0,0,1,0,0,0
 0,0,0,0,0,0,0,1
0,1,0,0,0,0,0,0
 0,0,0,0,0,1,0,0
1,0,0,0,0,0,0,0
 0,0,1,0,0,0,0,0
```

Running time:

Experinment-9: Develop a program and measure the running time for Graph Coloring with Backtracking

Aim: To Develop a program and measure the running time for Graph Coloring with Backtracking
Description:

Source Code:

```
#include <stdbool.h>
#include <stdio.h>
#define V 4
void printSolution(int color[]);
bool isSafe(bool graph[V][V], int color[])
{
// check for every edge
for (int i = 0; i < V; i++)
        for (int j = i + 1; j < V; j++)
                if (graph[i][j] && color[j] == color[i])
                        return false;
return true;
}
bool graphColoring(bool graph[V][V], int m, int i,
                        int color[V])
{
// if current index reached end
if (i == V) {
        // if coloring is safe
        if (isSafe(graph, color)) {
                // Print the solution
                printSolution(color);
                return true;
        }
        return false;
}

// Assign each color from 1 to m
for (int j = 1; j <= m; j++) {
        color[i] = j;
        if (graphColoring(graph, m, i + 1, color))
                return true;

        color[i] = 0;
}

return false;
}
void printSolution(int color[])
{
printf("Solution Exists:"
        " Following are the assigned colors \n");
```

```
for (int i = 0; i < V; i++)
        printf(" %d ", color[i]);
printf("\n");
}

// Driver code
int main()
{
test whether it is 3 colorable
(3)---(2)
| / |
| / |
| / |
(0)---(1)
*/
bool graph[V][V] = {
        { 0, 1, 1, 1 },
        { 1, 0, 1, 0 },
        { 1, 1, 0, 1 },
        { 1, 0, 1, 0 },
};
int m = 3; // Number of colors
int color[V];
for (int i = 0; i < V; i++)
        color[i] = 0;

// Function call
if (!graphColoring(graph, m, 0, color))
        printf("Solution does not exist");

return 0;
}
```

Output:

Solution Exists: Following are the assigned colors
1 2 3 2

Running time:

Experinment-10: Develop a program and measure the running time to generate solution of Hamiltonian Cycle problem with Backtracking

Aim: To Develop a program and measure the running time to generate solution of Hamiltonian Cycle problem with Backtracking

Description:

Source Code:

```
#include<stdio.h>
#define V 5

void printSolution(int path[]);
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{
if (graph [ path[pos-1] ][ v ] == 0)
        return false;
for (int i = 0; i < pos; i++)
        if (path[i] == v)
                return false;

return true;
}
bool hamCycleUtil(bool graph[V][V], int path[], int pos)
{
if (pos == V)
{       if ( graph[ path[pos-1] ][ path[0] ] == 1 )
        return true;
        else
        return false;
}
for (int v = 1; v < V; v++){
        if (isSafe(v, graph, path, pos))
        {
                path[pos] = v;
                if (hamCycleUtil (graph, path, pos+1) == true)
                        return true;
                path[pos] = -1;
        }
}
return false;
}
bool hamCycle(bool graph[V][V])
{
int *path = new int[V];
for (int i = 0; i < V; i++)
        path[i] = -1;
path[0] = 0;
if ( hamCycleUtil(graph, path, 1) == false )
{
```

```
        printf("\nSolution does not exist");
        return false;
}

printSolution(path);
return true;
}
void printSolution(int path[])
{
printf ("Solution Exists:"
                " Following is one Hamiltonian Cycle \n");
for (int i = 0; i < V; i++)
        printf(" %d ", path[i]);
printf(" %d ", path[0]);
printf("\n");
}
int main()
{
(0)--(1)--(2)
| / \ |
| / \ |
| /     \ |
(3)-------(4) */
bool graph1[V][V] = {{0, 1, 0, 1, 0},
                     {1, 0, 1, 1, 1},
                     {0, 1, 0, 0, 1},
                     {1, 1, 0, 0, 1},
                     {0, 1, 1, 1, 0},
                     };
hamCycle(graph1);
(0)--(1)--(2)
| / \ |
| / \ |
| /     \ |
(3)      (4) */
bool graph2[V][V] = {{0, 1, 0, 1, 0},
                     {1, 0, 1, 1, 1},
                     {0, 1, 0, 0, 1},
                     {1, 1, 0, 0, 0},
                     {0, 1, 1, 0, 0},
                     };
        hamCycle(graph2);

return 0;
}
```

Output:

```
Solution Exists: Following is one Hamiltonian Cycle
0 1 2 4 3 0
Solution does not exist
Running time:
```

Experinment-11: Develop a program and measure the running time running time to generate solution of Knapsack problem with Backtracking

Aim: To Develop a program and measure the running time running time to generate solution of Knapsack problem with Backtracking

Description:

Source Code:

```
#include <stdio.h>
int max(int a, int b) { return (a > b) ? a : 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 profit[] = { 60, 100, 120 };
int weight[] = { 10, 20, 30 };
int W = 50;
int n = sizeof(profit) / sizeof(profit[0]);
printf("%d", knapSack(W, weight, profit, n));
return 0;
}
```

Output: 220

Running time: