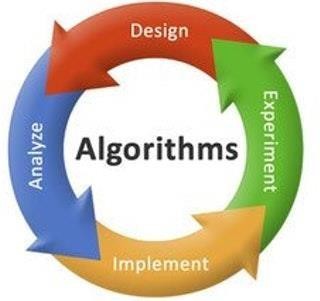
**T. JOHN INSTITUTE TECHNOLOGY**

**Department of Computer Science & Engineering**

**B.E - IV Semester 2023-2024**

ANALYSIS AND DESIGN OF ALGORITHMS MANUAL (BCSL404)



Name :

USN :

Batch: Section :

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Contents** | **Page no.** |
| 1 | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. | 5 |
| 2 | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm. | 6 |
| 3 | 1. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. 2. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm. | 7 |
| 4 | Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm. | 10 |
| 5 | Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph. | 11 |
| 6 | Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method. | 12 |
| 7 | Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method. | 14 |
| 8 | Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,. ,sn} of n  positive integers whose sum is equal to a given positive integer d. | 16 |
| 9 | Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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. | 17 |
| 10 | Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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. | 18 |
| 11 | Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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. | 20 |
| 12 | Design and implement C/C++ Program for N Queen's problem using Backtracking. | 21 |
| 13 | Viva-Questions. | 23 |

Syllabus

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ANALYSIS AND DESIGN OF ALGORITHM LABORATORY**  **[As per Choice Based Credit System (CBCS) scheme]** | | | | |
| **Course Code** | | BCSL404 | **CIE Marks** | 50 |
| **Teaching Hours/Week (L:T:P: S)** | | 0:0:2:0 | **SEE Marks** | 50 |
| **Credits** | | 01 | **Exam Hours** | 02 |
|  | | | | |
| **Course objectives:** This course will enable students to   * To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges. * To apply diverse design strategies for effective problem-solving. * To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks. | | | | |
| 1 | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm. | | | |
| 2 | Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm. | | | |
| 3 | 1. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. 2. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm. | | | |
| 4 | Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm. | | | |
| 5 | Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph. | | | |
| 6 | Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method. | | | |
| 7 | Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method. | | | |
| 8 | Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,. ,sn} of n  positive integers whose sum is equal to a given positive integer d. | | | |
| 9 | Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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. | | | |
| 10 | Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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. | | | |
| 11 | Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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. | | | |
| 12 | Design and implement C/C++ Program for N Queen's problem using Backtracking. | | | |

|  |
| --- |
| **Course outcomes (Course Skill Set):** |
| At the end of the course the student will be able to:   1. Develop programs to solve computational problems using suitable algorithm design strategy. 2. Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical). 3. Make use of suitable integrated development tools to develop programs 4. Choose appropriate algorithm design techniques to develop solution to the computational and complex problems. 5. Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences. |
| Conduct of Practical Examination: |
| **Assessment Details (both CIE and SEE):**  The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together.  **Continuous Internal Evaluation (CIE):**  CIE marks for the practical course are 50 Marks.  The split-up of CIE marks for record/ journal and test are in the ratio 60:40.   * Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session. * Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks. * Total marks scored by the students are scaled down to 30 marks (60% of maximum marks). * Weightage to be given for neatness and submission of record/write-up on time. * Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus. * In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce. * The suitable rubrics can be designed to evaluate each student’s performance and learning ability. * The marks scored shall be scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student. |

## Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

# include <stdio.h> int a,b,i,j,u,v,n;

int visited[10],edge=1, min, mincost=0, cost[10][10]; void kruskal(int n, int cost[10][10])

{

printf("The edge of the spanning tree are \n"); while(edge<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;

}

while(visited[u]) u=visited[u]; while(visited[v]) v=visited[v]; if(u!=v)

{

edge++;

printf("\n Edge (%d,%d)=%d", a,b,min); mincost=mincost+min;

visited[v]=u;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n The minimum cost =%d", mincost);

}

int main( )

{

printf("Enter the number of vertices:\n"); scanf("%d", &n);

printf("Enter the cost adjacency matrix:\n"); for(i=1;i<=n;i++)

for(j=1;j<=n;j++) scanf("%d", &cost[i][j]); kruskal(n,cost);

return 0;

}

# OUTPUT:

Enter the number of vertices**:**

4

Enter the cost adjacency matrix:

0 10 999 40 10

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 10 0 20 999 |  | 1 |  |  | 2 |
| 999 20 0 30 |  |  |  | 20 |  |
| 40 999 30 0 | 40 |  |  |  |  |
| The edge of the spanning tree is |  |  | 30 |  |  |
| Edge (1, 2) =10  Edge (2, 3) =20 |  | 4 |  |  | 3 |
| Edge (3, 4) =30 |  |  |  |  |  |
| The minimum cost =60 |  |  |  |  |  |

## Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

# include<stdio.h> int n,a[10][10]; void min\_spa\_tree()

{

int i,j,u,v,min,sum,k;

int t[10][2],p[10],d[10],s[10],source; min=9999;

source=0; for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(a[i][j]!=0 && a[i][j]<=min)

{

min=a[i][j]; source=i;

}

}

}

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

{

d[i]=a[source][i]; s[i]=0;

p[i]=source;

}

s[source]=1; sum=0; k=1;

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

{

min=9999; u=-1;

for(j=1;j<=n;j++)

{

if(s[j]==0)

{

if(d[j]<=min)

{

min=d[j]; u=j;

}

}

}

t[k][1]=u;

t[k][2]=p[u]; k++;

sum+=a[u][p[u]]; s[u]=1;

for(v=1;v<=n;v++)

{

if(s[v]==0 && a[u][v]<d[v])

{

d[v]=a[u][v]; p[v]=u;

}

}

}

if(sum>=9999)

printf("spanning tree does not exits\n"); else

{

printf("spanning tree exists and min spanning tree is \n"); for(i=1;i<n;i++)

{

printf("%d", t[i][1]); printf(" "); printf("%d", t[i][2]); printf("\n");

}

printf("The cost of the spanning tree is %d", sum); printf("\n");

}

}

int main( )

{

int i,j;

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

printf("Enter the cost adjacency matrix\n"); for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d", &a[i][j]);

}

}

min\_spa\_tree( ); return 0;

}

# OUTPUT:

Enter the no of nodes 4

Enter the cost adjacency matrix

0 10 999 40

10 0 20 999

999 20 0 30

40 999 30 0

Spanning tree exists and min spanning tree is

1 2

3 2

4 3

The cost of the spanning tree is 70

1. **a) Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.**

**b) Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm**

1. **Floyd's algorithm:**

#include <stdio.h>

#define INF 99999 // Infinity

// Function to implement Floyd's algorithm void floydWarshall(int graph[][4], int V) {

int dist[V][V], i, j, k;

// Initialize the distance matrix same as input graph matrix

for (i = 0; i < V; i++)

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

// Calculate shortest paths using Floyd-Warshall algorithm for (k = 0; k < V; k++) {

// Pick all vertices as source one by one for (i = 0; i < V; i++) {

// Pick all vertices as destination for the

// above picked source for (j = 0; j < V; j++) {

// If vertex k is on the shortest path from

// i to j, then update the value of dist[i][j] if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

}



}

}

// Print the shortest distance matrix

printf("The following matrix shows the shortest distances" " between every pair of vertices:\n");

for (i = 0; i < V; i++) { for (j = 0; j < V; j++) {

if (dist[i][j] == INF)

printf("%7s", "INF"); else

printf("%7d", dist[i][j]);

}

printf("\n");

}

}

int main() {

// Number of vertices in the graph int V = 4;

// Graph representation as an adjacency matrix int graph[4][4] = {{0, 5, INF, 10},

{INF, 0, 3, INF},

{INF, INF, 0, 1},

{INF, INF, INF, 0}};

// Call the function floydWarshall(graph, V); return 0;

}

## Warshal’s algorithm:

#include<stdio.h> int n,i,j;

int a[10][10];

int p[10][10];

int write\_data()

{

printf("the path matrix is shown below\n"); for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

printf("%d",p[i][j]); printf(" ");

}

printf("\n");

}

return 0;

}

void path\_matrix()

{

int i,j,k; for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++) p[i][j]=a[i][j];

}

for(k=1;k<=n;k++)

{

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

{

for(j=1;j<=n;j++)

{

if(p[i][k]==1&&p[k][j]==1) p[i][j]=1;

}

}

}

}

int main()

{

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

printf("enter the adjacency matrix\n"); for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

scanf("%d",&a[i][j]);

}

}

path\_matrix(); write\_data(); return 0;

}

# OUTPUT

Enter the no. of nodes 4

1

2

4

3

Enter the adjacency matrix 0 1 0 0

0 0 1 0

1 0 0 1

0 0 0 0

The path matrix is shown below 1 1 1 1

1 1 1 1

1 1 1 1

0 0 0 0

## Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

#include<stdio.h>

void dijstra(int n, int v, int cost[10][10], int d[])

{

int count,u,i,j,s[10],min; for(i=1;i<=n;i++)

{

s[i]=0;

d[i]=cost[v][i];

} s[v]=1;

d[v]=1;

count=2;

while(count<=n)

{

min=999; for(j=1;j<=n;j++) if(d[j]<min && s[j]==0)

{

min=d[j]; u=j;

} s[u]=1;

count++; for(j=1;j<=n;j++)

if((d[u]+cost[u][j]<d[j])&&s[j]==0) d[j]=d[u]+cost[u][j];

}

}

int main( )

{

int n,i,j,v,c[10][10],d[10]; printf("Enter the no of nodes"); scanf("%d", &n);

printf("Enter the cost adjacency matrix\n"); for(i=1;i<=n;i++)

for(j=1;j<=n;j++) scanf("%d", &c[i][j]);

printf("Enter the source vertex"); scanf("%d", &v);

dijstra(n,v,c,d); printf("The shortest path"); for(j=1;j<=n;j++)

if(j!=v)

printf("\n%d->%d=%d", v,j,d[j]); return 0;

}

# OUTPUT:

Enter the number of nodes 3

1

10

3

20

2

Enter the cost adjacency Matrix 0 10 999

30

999 0 20

30 999 0

Enter the source vertex 1

The shortest path 1-->2=10

1-->3=30

## Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

void find\_indegree(int n,int a[10][10],int indegree[])

{

int i,j,sum; for(j=1;j<=n;j++)

{

sum=0; for(i=1;i<=n;i++) sum=sum+a[i][j]; indegree[j]=sum;

}

}

void topological(int n,int a[10][10])

{

int i,k,u,v,top,t[10],indegree[10],s[10]; find\_indegree(n,a,indegree);

top=-1; k=1;

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

if(indegree[i]==0)

s[++top]=i;

while(top!=-1)

{

u=s[top--]; t[k++]=u; for(v=1;v<=n;v++)

{

if(a[u][v]==1)

{

indegree[v]--; if(indegree[v]==0) s[++top]=v;

}

}

}

printf("THE TOPOLOGICAL SEQUENCE IS \n");

for(i=1;i<=n;i++) printf("%d\t",t[i]);

}

int main()

{

int i,j,n,a[10][10];

printf("ENTER THE NUMBER OF NODES\n");

scanf("%d",&n);

printf("ENTER THE ADJACENCY MATRIX\n");

for(i=1;i<=n;i++) for(j=1;j<=n;j++) scanf("%d",&a[i][j]); topological(n,a);

return 0;

}

# OUTPUT:

ENTER THE NUMBER OF NODES

3

ENTER THE ADJACENCY MATRIX 0 1 1

0 0 0

0 1 0

THE TOPOLOGICAL SEQUENCE IS

1 3 2

## Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

**#include**<stdio.h> **#include**<stdlib.h>

**int** w[10], p[10], v[10][10];

**int Max**(**int** a,**int** b)

{

**if**(a>b) **return** a; **else return** b;

}

## int knapsack(int n,int c)

{

**int** i,j;

**for**(i=0;i<=n;i++)

{

**for**(j=0;j<=c;j++)

{

**if**(i==0||j==0) v[i][j]=0;

**else if**(j-w[i]>=0)

v[i][j]=Max(v[i-1][j],p[i]+v[i-1][j-w[i]]);

## else

v[i][j]=v[i-1][j];

}

}

**return** v[n][c];

}

## void optimalsubset(int n,int c)

{

**int** i,j;

**for**(i=n,j=c;i>=1 && j>0;i--)

{

**if**(v[i][j]!=v[i-1][j])

{

**printf**("item %d\n",i); j=j-w[i];

}

}

}

**int main**( )

{

**int** n,c,mp,i,j;

**printf**("enter the number of items \n");

**scanf**("%d",&n);

**printf**("enter the weights of each item\n");

**for**(i=1;i<=n;i++) **scanf**("%d",&w[i]);

**printf**("enter the profit of each item\n"); **for**(i=1;i<=n;i++) **scanf**("%d",&profits[i]);

**printf**("enter the knapsack capacity\n");

**scanf**("%d",&c); mp=knapsack(n,c);

**printf**("solution of the knapsack \n");

**for**(i=0;i<=n;i++)

{

**for**(j=0;j<=c;j++) **printf**("%d\t",v[i][j]); **printf**("\n");

}

**printf**("the maximal value is %d \n",mp); **printf**("the items of optimal subset are\n"); optimalsubset(n,c);

**return** 0;

}

# OUTPUT:

enter the number of items 4

enter the weights of each item 2

1

3

2

enter the profit of each item 12

10

20

15

Enter the knapsack capacity 5

Solution of the knapsack

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 maximal value is 37

The items of optimal subset are Item 4

Item 2

Item 1

## Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

**Discrete Knapsack**

#include <stdio.h>

// Structure to represent an item struct Item {

int weight; int value;

};

// Function to solve the discrete Knapsack problem using greedy approximation method void discreteKnapsack(struct Item items[], int n, int capacity) {

// Calculate value-to-weight ratio for each item float ratios[n];

for (int i = 0; i < n; i++) {

ratios[i] = (float) items[i].value / items[i].weight;

}

// Sort items based on value-to-weight ratio in descending order for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) { if (ratios[i] < ratios[j]) {

float tempRatio = ratios[i]; ratios[i] = ratios[j]; ratios[j] = tempRatio;

struct Item tempItem = items[i]; items[i] = items[j];

items[j] = tempItem;

}

}

}

// Fill knapsack

int totalWeight = 0; float totalValue = 0.0;

printf("Items selected:\n");

for (int i = 0; i < n && totalWeight < capacity; i++) { if (items[i].weight <= capacity - totalWeight) {

totalWeight += items[i].weight; totalValue += items[i].value;

printf("Item with weight %d and value %d\n", items[i].weight, items[i].value);

} else {

int remainingWeight = capacity - totalWeight;

totalValue += items[i].value \* ((float) remainingWeight / items[i].weight);

printf("Item with weight %d and value %d (fractional part)\n", items[i].weight, items[i].value); break;

}

}

printf("Total value: %.2f\n", totalValue);

}

int main() {

// Example items

struct Item items[] = {{10, 60}, {20, 100}, {30, 120}}; int n = sizeof(items) / sizeof(items[0]);

int capacity = 50;

// Solve discrete Knapsack problem discreteKnapsack(items, n, capacity);

return 0;

}

## continuous Knapsack problem

#include <stdio.h>

// Structure to represent an item struct Item {

int weight; int value;

};

// Function to solve the continuous Knapsack problem using greedy approximation method void continuousKnapsack(struct Item items[], int n, int capacity) {

// Calculate value-to-weight ratio for each item float ratios[n];

for (int i = 0; i < n; i++) {

ratios[i] = (float) items[i].value / items[i].weight;

}

// Sort items based on value-to-weight ratio in descending order for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) { if (ratios[i] < ratios[j]) {

float tempRatio = ratios[i]; ratios[i] = ratios[j]; ratios[j] = tempRatio;

struct Item tempItem = items[i]; items[i] = items[j];

items[j] = tempItem;

}

}

}

// Fill knapsack

int totalWeight = 0; float totalValue = 0.0;

printf("Items selected:\n");

for (int i = 0; i < n && totalWeight < capacity; i++) { if (items[i].weight <= capacity - totalWeight) {

totalWeight += items[i].weight; totalValue += items[i].value;

printf("Item with weight %d and value %d\n", items[i].weight, items[i].value);

} else {

int remainingWeight = capacity - totalWeight;

totalValue += items[i].value \* ((float) remainingWeight / items[i].weight);

printf("Item with weight %d and value %d (fractional part)\n", items[i].weight, items[i].value); break;

}

}

printf("Total value: %.2f\n", totalValue);

}

int main()

{

struct Item items[] = {{10, 60}, {20, 100}, {30, 120}}; int n = sizeof(items) / sizeof(items[0]);

int capacity = 50;

// Solve continuous Knapsack problem continuousKnapsack(items, n, capacity);

return 0;

}

## Find a subset of a given set S = {sl, 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>

int w[20], d,n,count,x[10], i,k; void sumsubsets(int s, int k, int r)

{

x[k]=1;

if(s+w[k]==d)

{

}

else

printf("\n subset %d\t", ++count); for(i=0;i<=k;i++)

if(x[i])

printf("%d ", w[i]);

if(s+w[k]+w[k+1]<=d) sumsubsets(s+w[k],k+1,r-w[k]);

if((s+r-w[k]>=d) && (s+w[k+1]<=d))

{

x[k]=0;

sumsubsets(s,k+1,r-w[k]);

}

}

int main( )

{

int sum=0;

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

printf("Enter the elements :"); for(i=0;i<n;i++)

scanf("%d", &w[i]); printf("Enter the sum :\n"); scanf("%d", &d); for(i=0;i<n;i++)

x[i]=0;

for(i=0;i<n;i++) sum+=w[i]; if(sum<d || w[0]>d)

printf("\n no subsets is possible"); else

sumsubsets(0,0,sum); return 0;

}

# OUTPUT:

Enter the no. of elements: 4 Enter the elements: 2 3 1 4 Enter the sum: 5

Subset 1 >2 3

Subset 2 >1 4

Enter the no. of elements4 Enter the elements: 2 3 1 4 Enter the sum: 15

No subsets is possible

## Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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 <stdlib.h> #include <time.h>

// Function to perform selection sort void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) { int min\_idx = i;

for (int j = i + 1; j < n; j++) { if (arr[j] < arr[min\_idx]) {

min\_idx = j;

}

}

// Swap the found minimum element with the first element int temp = arr[i];

arr[i] = arr[min\_idx]; arr[min\_idx] = temp;

}

}

// Function to generate random integers and store them in a file void generateRandomNumbersToFile(int n, const char\* filename) {

FILE \*file = fopen(filename, "w"); if (file == NULL) {

printf("Error opening file.\n"); exit(1);

}

srand(time(NULL)); // Seed for random number generation for (int i = 0; i < n; i++) {

fprintf(file, "%d\n", rand());

}

fclose(file);

}

// Function to read integers from a file and store them in an array void readIntegersFromFile(int arr[], int n, const char\* filename) {

FILE \*file = fopen(filename, "r"); if (file == NULL) {

printf("Error opening file.\n"); exit(1);

}

for (int i = 0; i < n; i++) { fscanf(file, "%d", &arr[i]);

}

fclose(file);

}

int main() {

FILE \*timeFile = fopen("time\_vs\_n.csv", "w"); if (timeFile == NULL) {

printf("Error opening file.\n"); exit(1);

}

fprintf(timeFile, "n, time\_taken\_ms\n");

for (int n = 5000; n <= 10000; n += 1000) {

// Generate random numbers and store in a file generateRandomNumbersToFile(n, "random\_numbers.txt");

// Read integers from the file int arr[n];

readIntegersFromFile(arr, n, "random\_numbers.txt");

// Measure the time taken to sort clock\_t start = clock(); selectionSort(arr, n);

clock\_t end = clock();

double time\_taken = ((double)(end - start)) \* 1000 / CLOCKS\_PER\_SEC;

// Record the time taken

fprintf(timeFile, "%d, %lf\n", n, time\_taken);

}

fclose(timeFile);

printf("Time taken for sorting recorded in time\_vs\_n.csv\n"); return 0;

}

# OUTPUT:

Enter the size 5

The sorted array is

15

77

83

86

93

Time taken 1.000000

## Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. 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<time.h> #include<stdlib.h> #define maxsize 3000

void quicksort(int a[],int l,int h)

{

int j; if(l<h)

{

usleep(500000); j=partition(a,l,h); quicksort(a,l,j-1); quicksort(a,j+1,h);

}

}

int partition(int a[],int l,int h)

{

int i,j,k,t;

k=a[l]; i=l+1; j=h; while(1)

{

while(l<h && k>=a[i]) i++;

while(k<a[j])

j--;

if(i<j)

{

}

else

{

}

}

}

t=a[i]; a[i]=a[j]; a[j]=t;

t=a[l]; a[l]=a[j]; a[j]=t; return j;

int main()

{

int a[maxsize],i,n; double runtime=0;

printf("Enter the size \n"); scanf("%d",&n);

srand(1);// generates random numbers for(i=1;i<=n;i++)

a[i]=rand()%100;

time\_t start=time(NULL); quicksort(a,1,n);

time\_t end=time(NULL); runtime=difftime(end,start); printf("The sorted array is"); for(i=1;i<=n;i++) printf("\n%d\n", a[i]);

printf("Time taken %f\n",runtime); return 0;

}

# OUTPUT:

Enter the size 5

The sorted array is 15

77

83

86

93

Time taken 1.000000

## Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. 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<time.h> #include<stdlib.h> #define maxsize 3000

void merge(int a[],int low,int mid,int high)

{

int i,j,k,p;

int b[maxsize]; i=k=low; j=mid+1;

while(i<=mid && j<=high)

{

if(a[i]<=a[j])

{

}

else

{

}

k=k+1;

}

b[k]=a[i]; i=i+1;

b[k]=a[j]; j=j+1;

if(i>mid) for(p=j;p<=high;p++)

{

}

else

b[k]=a[p]; k=k+1;

for(p=i;p<=mid;p++)

{

b[k]=a[p]; k=k+1;

}

for(i=low;i<=high;i++) a[i]=b[i];

}

void mergesort(int a[],int low, int high)

{

int mid; if(low<high)

{

usleep(50000); mid=(low+high)/2; mergesort(a,low,mid-1); mergesort(a,mid+1,high); merge(a,low,mid,high);

}

}

int main()

{

int a[maxsize],i,n; double runtime=0;

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

srand(1); for(i=1;i<=n;i++) a[i]=rand()%100;

printf("The randomly generated numbers are:\n"); for(i=1;i<=n;i++)

printf("%d\n", a[i]);

{

time\_t start=time(NULL); mergesort(a,1,n);

time\_t end=time(NULL); runtime=difftime(end,start);

}

printf("The sorted elements are :\n"); for(i=1;i<=n;i++)

printf("%d\n", a[i]);

printf("Time taken to sort=%f\n",runtime ); return 0;

}

# OUTPUT:

Enter the number of elements to sort 5

The randomly generated numbers are: 83

86

77

15

93

The sorted elements are :

15

83

86

77

93

Time taken to sort=0.001

## Design and implement C/C++ Program for N Queen's problem using Backtracking.

#include<stdio.h> #include<stdlib.h> int x[10],n;

int display()

{

int i,j; for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++) if(x[i]==j)

printf("\tQ");

else printf("\t x");

printf("\n\n");

}

printf("\n\n"); return 0;

}

int place(int k, int i)

{

int j;

for(j=1;j<=k-1;j++) if((x[j]==i)||(abs(x[j]-i)==abs(j-k))) return 0;

return 1;

}

int nqueen(int k)

{

int i; for(i=1;i<=n;i++) if(place(k,i))

{

x[k]=i; if(k==n) display(); nqueen(k+1);

}

}

int main()

{

printf("enter the no. of queens\n"); scanf("%d",&n); if(n==1||n==2||n==3)

printf(" nqueen solution doesnot exist\n"); else

printf("solution for nqueen\n"); nqueen(1);

}

# OUTPUT:

Enter the no. of queens 4

solution for nqueen

|  |  |  |  |
| --- | --- | --- | --- |
| x | Q | x | x |
| x | x | x | Q |
| Q | x | x | x |
| x | x | Q | x |
| x | x | Q | x |
| Q | x | x | x |

|  |  |  |  |
| --- | --- | --- | --- |
| x | x | x | Q |
| x | Q | x | x |

# VIVA-QUESTIONS

## What is an algorithm?

An algorithm is a sequence of unambiguous instructions for solving a problem. i.e., for obtaining a required output for any legitimate input in a finite amount of time

1. **What are important problem types? (or) Enumerate some important types of problems. 1**. Sorting 2. Searching 3. Numerical problems 4. Geometric problems 5. Combinatorial Problems 6. Graph Problems 7. String processing Problems

## Name some basic Efficiency classes

1. Constant 2. Logarithmic 3. Linear 4. nlogn 5. Quadratic 6. Cubic 7. Exponential 8. Factorial

## What are algorithm design techniques?

Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing. General design techniques are: (i) Brute force

1. divide and conquer
2. decrease and conquer
3. transform and conquer
4. greedy technique
5. dynamic programming
6. backtracking
7. branch and bound

## How is algorithm‟s time efficiency measured?

Time efficiency indicates how fast the algorithm runs. An algorithm’s time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm’s innermost loop.

## How is the efficiency of the algorithm defined?

The efficiency of an algorithm is defined with the components. (i) Time efficiency -indicates how fast the algorithm runs (ii) Space efficiency -indicates how much extra memory the algorithm needs.

## What are the characteristics of an algorithm?

Every algorithm should have the following five characteristics (i) Input (ii) Output (iii) Definiteness (iv) Effectiveness (v) Termination Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input.

## Write general plan for analyzing non-recursive algorithms.

* 1. Decide on parameter indicating an input’s size.
  2. Identify the algorithm’s basic operation
  3. Checking the no.of times basic operation executed depends on size of input.
  4. Set up sum expressing the no.of times the basic operation is executed. depends on some additional property,then best,worst,avg.cases need to be investigated (establishing order of growth)

## Define the terms: pseudo code, flow chart.

A pseudo code is a mixture of a natural language and programming language like constructs. A pseudocode is usually more precise than natural language. A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm’s steps.

## write general plan for analyzing recursive algorithms.

* 1. Decide on parameter indicating an input’s size.
  2. Identify the algorithm’s basic operation iii. Checking the no.of times basic operation executed depends on size of input.if it depends on some additional property,then best,worst,avg.cases need to be investigated of timesthe basic operation is executed

1. Set up the recurrence relation,with an appropriate initial condition,for the number
2. Solve recurrence (establishing order of growth)

## Define the divide and conquer method.

Given a function to compute on ‘n’ inputs the divide-and-comquer strategy suggests splitting the inputs in to’k’ distinct susbsets, 1<k<n, yielding ‘k’ subproblems. The subproblems must be solved recursively, and then a method must be found to combine subsolutions into a solution of the whole.

## What is Merge sort?

Merge sort is divide and conquer strategy that works by dividing an input array in to two halves,sorting them recursively and then merging the two sorted halves to get the original array sorted.

## What is general divide and conquer recurrence?

Time efficiency T(n)of many divide and conquer algorithms satisfies the equation T(n)=a.T(n/b)+f(n).This is the general recurrence relation.

## Describe the recurrence relation for merge sort?

If the time for the merging operation is proportional to n, then the computing time of merge sort is described by the recurrence relation

T(n) = a n = 1, a a constant 2T (n/2) + n n >1, c a constant

## The relation between order of growth of functions

O(1) < O(log n) < O(n) < O(n \* log n) < O(n2 ) < O(n3 ) < O(2n )

## Asymptotic notations

Big Oh(Worst Case), Big Theta (Average Case), Big Omega(Best Case)

## Explain the greedy method

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given ‘n’ inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one candevice an algorithm that works in stages considering one input at a time.

## Define feasible and optimal solution.

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution. A feasible solution either maximizes or minimizes the given objective functionis called as optimal solution.

## What are the constraints of knapsack problem?

To maximize ∑pixi

The constraint is : ∑wixi ≥ m and 0 ≤ xi ≤ 1 1≤ i ≤ n

where m is the bag capacity, n is the number of objects and for each object i wi and pi are the weight and profit of object respectively.

## Specify the algorithms used for constructing Minimum cost spanning tree

a) Prim’s Algorithm b) Kruskal’s Algorithm

## State single source shortest path algorithm (Dijkstra‟s algorithm).

For a given vertex called the source in a weigted connected graph,find shotrtest paths to all its other vertices.Dijikstra’s algorithm applies to graph with non-negative weights only.

## State efficiency of prim‟s algorithm.

O(|v|2 ) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY) O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

## State Kruskal Algorithm

The algorithm looks at a MST for a weighted connected graph as an acyclic subgraph with |v|-1 edges for which the sum of edge weights is the smallest.

## . State efficiency of Dijkstra‟s algorithm.

O(|v|2 ) (WEIGHT MATRIX AND PRIORITY QUEUE AS UNORDERED ARRAY) O(|E| LOG|V|) (ADJACENCY LIST AND PRIORITY QUEUE AS MIN-HEAP)

## Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned into K>=2 disjoint sets Vi,1<=i<=k.The multi stage graph problem is to find a minimum cost paths from s(source ) to t(sink) Two approach(forward and backward)

## Define All pair shortest path problem

Given a weighted connected graph, all pair shortest path problem asks to find the lengths of shortest paths from each vertex to all other vertices.

## Define floyd‟s algorithm.

To find all pair shortest path.

## State the time efficiency of floyd‟s algorithm

O(n3 )

## What is binary search?

It is an efficient method of finding out a required item from a given list, provided the list is in order. The process is:

1. First the middle item of the sorted list is found.
2. Compare the item with this element.
3. If they are equal search is complete. 4. If the middle element is greater than the item being searched, this process is repeated in the upper half of the list. 5. If the middle element is lesser than the item being searched, this process is repeated in the lower half of the list.

## What is the another name for Quicksort?

Partition Exchange Sort

## What is DFS traversal?

Consider an arbitrary vertex v and mark it as visited on each iteration, proceed to an unvisited vertex w adjacent to v. we can explore this new vertex depending upon its adjacent information. This process continues until dead end is encountered.(no adjacent unvisited vertex is available). At the dead end the algorithm backs up by one edge and continues the process of visiting unvisited vertices. This process continues until we reach the starting vertex.

1. **What are the various applications of BFS & DFS tree traversal technique? Application of BFS.** Checking connectivity and checking acyclicity of a graph. Check whether there is only one root in the BFS forest or not. If there is only one root in the BFS forest, then it is connected graph otherwise disconnected graph. For checking cycle presence, we can take advantage of the graph's representation in the form of a BFS forest, if the latter vertex does not have cross edge, then the graph is acyclic.

## Application of DFS

To check whether given graph is connected or not. To check whether the graph is cyclic or not.

To find the spanning tree. Topological sorting

## What is shortest path?

- It is the path which is having shortest length among all possible paths.

## what is state space tree?

Constructing a tree of choices being made and processing is known as state-spacetree. Root represents an initial state before the search for solution begins.

## What are the requirements that are needed for performing Backtracking?

Complex set of constraints. They are:

* 1. Explicit constraints.
  2. Implicit constraints.