

RNS INSTITUTE OF TECHNOLOGY

(AICTE Approved, VTU Affiliated and NAAC 'A+' Grade Accredited)

(UG programs – CSE, ECE, ISE, EIE and EEE are Accredited by NBA up to 30.6.2025)

Channasandra, Dr. Vishnuvardhan Road, Bengaluru-560098



Department of Information Science & Engineering

ANALYSIS & DESIGN OF ALGORITHMS

LABORATORY MANUAL

IV Semester

BCSL404

Faculty-in-charge

Dr. Suresh L.

Dr. Sunitha K.

Ms. Varshitha M. K.

Mr. Janardhan Singh K.

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Information Science and Engineering



VISION of the College

Building RNSIT into a World - Class Institution

MISSION of the College

To impart high quality education in Engineering, Technology and Management with a difference, enabling students to excel in their career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to *achieve distinctions in various walks of life* leading to outstanding contributions.
2. Imparting value based, need based, and choice based and skill based professional education to the aspiring youth and *carving them into disciplined, World class Professionals* with social responsibility.
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students.
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development.
5. Providing freedom of action and choice for all the Stake holders with better visibility.

VISION of the Department

Building Information Technology Professionals by Imparting Quality Education and Inculcating Key Competencies

MISSION of the Department

- Provide strong fundamentals through learner centric approach
- Instill technical, interpersonal, interdisciplinary skills and logical thinking for holistic development
- Train to excel in higher education, research, and innovation with global perspective
- Develop leadership and entrepreneurship qualities with societal responsibilities

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

ISE Graduates within four years of graduation will have

- **PEO1:** Acquired the fundamentals of computers and applied knowledge of Information Science & Engineering and continue to develop their technical competencies by problem solving using programming.
- **PEO2:** Ability to formulate problems, attained the Proficiency to develop system/application software in a scalable and robust manner with various platforms, tools and frameworks to provide cost effective solutions.
- **PEO3:** Obtained the capacity to investigate the necessities of the software Product, adapt to technological advancement, promote collaboration and interdisciplinary activities, Protecting Environment and developing Comprehensive leadership.
- **PEO4:** Enabled to be employed and provide innovative solutions to real-world problems across different domains.
- **PEO5:** Possessed communication skills, ability to work in teams, professional ethics, social responsibility, entrepreneur and management, to achieve higher career goals, and pursue higher studies.

PROGRAM OUTCOMES(POs)

Engineering Graduates will be able to:

- **PO1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems
- **PO2: Problem analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
- **PO3: Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.
- **PO4: Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.
- **PO6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess Societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the

knowledge of, and need for sustainable development.

- **PO8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9: Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10: Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11: Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES(PSOs)

ISE Graduates will have

- **PSO1 – Problem Solving Abilities:** Ability to demonstrate the fundamental and theoretical concepts, analyze the real time problems and develop customized software solutions by applying the knowledge of mathematics and algorithmic techniques.
- **PSO2 – Applied Engineering Skills:** Enable creative thinking, Ability to apply standard practices and strategies, technical skills in software design, development, integration of systems and management for improving the security, reliability and survivability of the infrastructure.
- **PSO3 – General Expertise and Higher Learning:** Ability to exchange knowledge effectively demonstrate the ability of team work, documentation skills, professional ethics, entrepreneurial skills and continuing higher education in the field of Information technology.

RNS INSTITUTE OF TECHNOLOGY

Dr. VISHNUVARDHAN ROAD, CHANNASANDRA, BENGALURU -560 098

Department of Information Science and Engineering

Analysis and Design of Algorithms Lab

Subject Code: BCSL404

Hours/Week: 2P

Total Hours: 24

Exam Hours: 02

Scheme	NEP (22 SCHEME)
I.A. Marks	50
Credits	01

Course objectives

- 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.

Course Outcomes

At the end of the course the student will be able to:

CO1	Develop programs to solve computational problems using suitable algorithm design strategy.
CO2	Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).
CO3	Make use of suitable integrated development tools to develop programs.
CO4	Choose appropriate algorithm design techniques to develop solution to the computational and complex problems.
CO5	Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences.

CO mapping to PO/PSOs

CO / PO & PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO 1	2	3	3	2	3							1	3	2	
CO2	3	3	3	2	3							2	3	3	
CO3	3	3	3	2	3							2	3	3	
CO4	3	3	3	3	3							2	2	3	
CO5	3	3	3	3	3							1	2	3	

Data Structures Laboratory Evaluation Rubrics

Subject Code: BSCL404

Hours/Week: 2P

Total Hours: 24

I.A. Marks : 50

Exam Hours: 02

Exam Marks: 50

Lab Write-up and Execution Rubrics (Max: 24 marks)

		Above Average	Average	Below Average
a.	Understanding of problem and approach to solve. (8 Marks)	Able to analyze the given problem and efficiently implement using suitable high-level language instructions. (8-6)	Able to analyze the problem and moderate understanding of high-level language instructions. (5-3)	Poor understanding of high-level language instructions or No program write-up. (2-0)
b.	Execution and Viva (5 questions) (8 Marks)	Program executed for varied inputs with valid results and able to answer all five questions appropriately. (8-6)	Program is executed for some inputs and able to answer three-two questions. (5-3)	Program has compilation errors or no Execution and not answered any questions. (2-0)
c.	Results and Documentation (8 Marks)	Program and results obtained are legibly written / documented. (8-6)	Program and results obtained is acceptably documented. (5-3)	No Proper results and poor documentation. (2-0)

LAB Internal Assessment rubrics (Max: 16 marks)

		Above Average	Average	Below Average
a.	Write-up (3 Marks)	Able to write the complete code. (3)	Able to write the code with few errors. (2-1)	Unable to write. (0)
b.	Execution (10 Marks)	Executed successfully for all the inputs given. (10-7)	Obtained partially correct results. (6-3)	Program has compilation errors or No Execution. (2-0)
c.	Viva (5 questions) (3 Marks)	Able to answer all five questions correctly. (3)	Able to answer three-two questions. (2-1)	Not answered any. (0)

Mini-project Assessment rubrics (Max: 10 marks)

		Above Average	Average	Below Average
a.	Mini-project (10 Marks)	Able to develop a mini-project completely and given presentation along with the team members. (10-7)	Able to develop a mini-project partially and given presentation along with the team members. (6-3)	Not able to develop mini-project or absent for the presentation. (2-0)

FACULTY INCHARGE

Dr. Suresh L.

Dr. Sunitha K.

Ms. Varshitha M.K.

Mr. Janardhan Singh K.

HoD, Dept of ISE

Lesson Planning / Schedule of Experiments

Sl. No	List of Programs	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.	8
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.	10
3	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.	12
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.	15
5	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.	17
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.	19
7	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.	21
8	Design and implement C/C++ Program to find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .	23
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.	25
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.	27
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.	29
12	Design and implement C/C++ Program for N Queen's problem using Backtracking.	31

1. 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 cost[10][10],n;

void kruskal()
{
    int par[n];
    int a=0,b=0,u=0,v=0,min, mincost = 0, ne = 0;
    for(int i=0;i<n;i++)
        par[i]=-1;

    printf("the minimum spanning tree edges are...");
    while(ne < n-1)
    {
        //Find the least cost edge
        min = 999;
        for(int i=0;i<n;i++)
            for(int j=0;j<n;j++)
                if(cost[i][j] < min)
                {
                    min=cost[i][j];
                    a=u=i;
                    b=v=j;
                }

        //Check if edge select cause cyclicity?
        while(par[u]!=-1)
            u=par[u];

        while(par[v]!=-1)
            v=par[v];

        if(u!=v)
        {
            printf("From vertex %d to vertex %d and the cost = %d\n",a,b,min);
            mincost+=min;
            par[v]=u;
            ne++;
        }
        //edge included in MST should not be considered for next iteration
        cost[a][b]=cost[b][a]=999;
    }
    printf("Cost of MST = %d", mincost);
}

void main()
{
    printf("Enter the no. of vertices:");
    scanf("%d",&n);
    printf("Enter the cost matrix\n");
```

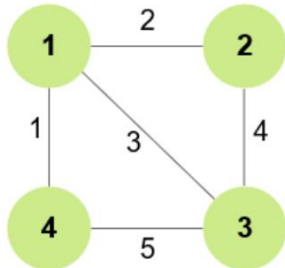


```

for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
        scanf("%d",&cost[i][j]);
kruskal();
}

```

Sample Input and Output:



Enter the no. of vertices:4

Enter the cost matrix

999 2 3 1

2 999 4 999

3 4 999 5

1 999 5 999

the minimum spanning tree edges are...

From vertex 0 to vertex 3 and the cost = 1

From vertex 0 to vertex 1 and the cost = 2

From vertex 0 to vertex 2 and the cost = 3

Cost of MST = 6

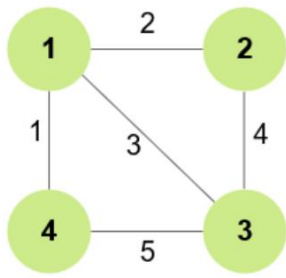
2. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
#include<stdio.h>
int cost[10][10],n;

void prim()
{
    int vt[10]={0};
    int a=0,b=0,min, mincost = 0, ne = 0;
    //start from the first vertex
    vt[0] = 1;
    while(ne < n-1)
    {
        //Find the nearest neighbour
        min = 999;
        for (int i = 0; i<n; i++)
        {
            if(vt[i]==1)
                for(int j = 0;j <n; j++)
                    if(cost[i][j] < min && vt[j]==0)
                    {
                        min = cost[i][j];
                        a = i;
                        b = j;
                    }
        }
        //Include nearest neighbour 'b' into MST
        printf("Edge from vertex %d to vertex %d and the cost %d\n",a,b,min);
        vt[b] = 1;
        ne++;
        mincost += min;
        cost[a][b] = cost[b][a] = 999;
    }
    printf("minimum spanning tree cost is %d",mincost);
}

void main()
{
    printf("Enter the no. of vertices: ");
    scanf("%d",&n);
    printf("Enter the cost matrix\n");
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    prim();
}
```

Sample Input and Output:



Enter the no. of vertices: 4

Enter the cost matrix

999 2 3 1

2 999 4 999

3 4 999 5

1 999 5 999

Edge from vertex 0 to vertex 3 and the cost 1

Edge from vertex 0 to vertex 1 and the cost 2

Edge from vertex 0 to vertex 2 and the cost 3

minimum spanning tree cost is 6

3a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
#include<stdio.h>

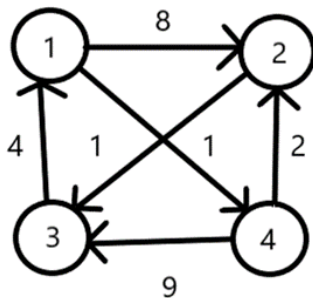
int min(int a, int b)
{
    return(a < b ? a : b);
}

void floyd(int D[][10],int n)
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                D[i][j]=min(D[i][j],D[i][k]+D[k][j]);
}

int main()
{
    int n, cost[10][10];
    printf("Enter no. of Vertices: ");
    scanf("%d",&n);
    printf("Enter the cost matrix\n");
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            scanf("%d",&cost[i][j]);
    floyd(cost,n);

    printf("All pair shortest path\n");
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            printf("%d ",cost[i][j]);
        printf("\n");
    }
}
```

Sample Input and Output:



Enter no. of Vertices: 4
Enter the cost matrix
999 8 4 999
999 999 1 999

4 999 999 999
999 2 9 999
All pair shortest path
8 8 4 999
5 13 1 999
4 12 8 999
7 2 3 999

Dept. of ISE, RNSIT

3b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

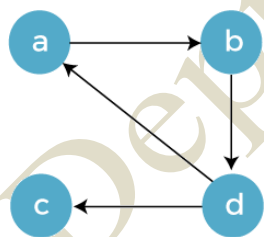
```
#include<stdio.h>

void warshal(int A[][10],int n)
{
    for(int k=1;k<=n;k++)
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                A[i][j]=A[i][j] || (A[i][k] && A[k][j]);
}

void main()
{
    int n, adj[10][10];
    printf("Enter no. of Vertices: ");
    scanf("%d",&n);
    printf("Enter the adjacency matrix\n");
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            scanf("%d",&adj[i][j]);
    warshal(adj,n);

    printf("Transitive closure of the given graph is\n");
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
            printf("%d ",adj[i][j]);
        printf("\n");
    }
}
```

Sample Input and Output:



```
Enter no. of Vertices: 4
Enter the adjacency matrix
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive closure of the given graph is
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

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.

```
#include<stdio.h>

int cost[10][10],n,dist[10];

int minm(int m, int n)
{
    return(( m < n) ? m: n);
}

void dijkstra(int source)
{
    int s[10]={0};
    int min, w=0;

    for(int i=0;i<n;i++)
        dist[i]=cost[source][i];

    //Initialize dist from source to source as 0
    dist[source] = 0;

    //mark source vertex - estimated for its shortest path
    s[source] = 1;
    for(int i=0; i < n-1; i++)
    {
        //Find the nearest neighbour vertex
        min = 999;
        for(int j = 0; j < n; j++)
        {
            if ((s[j] == 0 ) && (min > dist[j]))
            {
                min = dist[j];
                w = j;
            }
        }
        s[w]=1;
        //Update the shortest path of neighbour of w
        for(int v=0;v<n;v++)
        {
            if(s[v]==0 && cost[w][v]!=999)
            {
                dist[v]= minm(dist[v],dist[w]+cost[w][v]);
            }
        }
    }
}

int main()
{
```

```
int source;
```

```
printf("Enter the no.of vertices:");
```

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

```
printf("Enter the cost matrix\n");
```

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

```
    for(int j=0;j<n;j++)
```

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

```
printf("Enter the source vertex:");
```

```
scanf("%d",&source);
```

```
dijkstra(source);
```

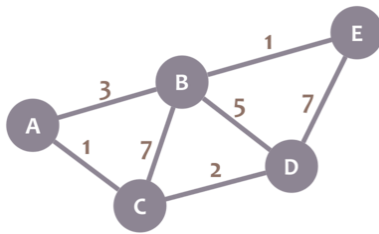
```
printf("the shortest distance is...");
```

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

```
    printf("Cost from %d to %d is %d\n",source,i,dist[i]);
```

```
}
```

Sample Input and Output:



Enter the no.of vertices:5

Enter the cost matrix

0 3 1 999 999

3 0 7 5 1

1 7 0 2 999

999 5 2 0 7

999 1 999 7 0

Enter the source vertex:0

the shortest distance is...Cost from 0 to 0 is 0

Cost from 0 to 1 is 3

Cost from 0 to 2 is 1

Cost from 0 to 3 is 3

Cost from 0 to 4 is 4

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

//Solved using Source removal Method

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

```
int cost[10][10],n,colsum[10];
```

```
void cal_colsum()
```

```
{
    for(int j=0;j<n;j++)
    {
        colsum[j]=0;
        for(int i=0;i<n;i++)
            colsum[j]+=cost[i][j];
    }
}
```

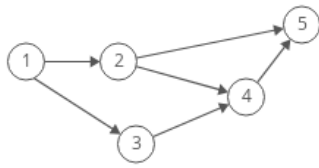
```
void source_removal()
```

```
{
    int i,j,k,select[10]={0};
    printf("Topological ordering is:");
    for(i=0;i<n;i++)
    {
        //Calculate the outdegree for each vertices
        cal_colsum();
        for(j=0;j<n;j++)
        {
            if(select[j]==0 && colsum[j]==0)//source vertex
                break;
        }
        printf("%d ",j);
        select[j]=1;
        //Remove source vertex j from cost matrix
        for(k=0;k<n;k++)
            cost[j][k]=0;
    }
}
```

```
void main()
```

```
{
    printf("Enter no. of Vertices: ");
    scanf("%d",&n);
    printf("Enter the cost matrix\n");
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%d",&cost[i][j]);
    source_removal();
}
```

Sample Input and Output:



Enter no. of Vertices: 5

Enter the cost matrix

0 1 1 0 0

0 0 0 1 1

0 0 0 1 0

0 0 0 0 1

0 0 0 0 0

Topological ordering is: 0 1 2 3 4

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

```
#include<stdio.h>
int n,m,p[10],w[10];

int max(int a, int b)
{
    return(a>b?a:b);
}

void knapsack_DP()
{
    int V[10][10],i,j;
    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])//weight of the item is larger than capacity
                V[i][j]=V[i-1][j];
            else
                V[i][j]=max(V[i-1][j],p[i]+V[i-1][j-w[i]]);//maximization

    for(i=0;i<=n;i++)
    {
        for(j=0;j<=m;j++)
            printf("%d ",V[i][j]);
        printf("\n");
    }
    /* tracking back the optimal solution vector */
    printf("Items included are:");
    while(n > 0)
    {
        if(V[n][m] != V[n-1][m])
        {
            printf("%d ",n);
            m = m - w[n];
        }
        n--;
    }
}

int main()
{
    int i;
    printf("Enter the no. of items: ");
    scanf("%d",&n);
    printf("Enter the weights of n items: ");
    for(i=1;i<=n;i++)
        scanf("%d",&w[i]);
    printf("Enter the prices of n items: ");
    for(i=1;i<=n;i++)
```

```
scanf("%d",&p[i]);
printf("Enter the capacity of Knapsack: ");
scanf("%d",&m);
knapsack_DP();
}
```

Sample Input and Output:

Enter the no. of items: 4

Enter the weights of n items: 7 3 4 5

Enter the prices of n items: 42 12 40 25

Enter the capacity of Knapsack: 10

0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 42 42 42 0

0 0 0 12 12 12 12 42 42 42 0

0 0 0 12 40 40 40 52 52 52 0

0 0 0 12 40 40 40 52 52 65 65

Items included are: 4 3

Dept. of ISE, RNSIT

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

```
#include<stdio.h>

int n,m,p[10],w[10];
void greedy_knapsack()
{
    float max, profit=0;
    int k=0,i,j;
    printf("item included is :");
    for(i=0;i<n;i++)
    {
        max=0;
        //choose the item which has highest price to weight ratio
        for(j=0;j<n;j++)
        {
            if(((float)p[j])/w[j] > max)
            {
                k=j;
                max=((float)p[j])/w[j];
            }
        }
        //kth element has highest price to weight ratio
        if(w[k] <= m )
        {
            printf("%d",k);
            m = m - w[k];
            profit=profit+p[k];
            p[k]=0;
        }
        else
            break;//unable fit item k into knapsack
    }
    printf("Discrete Knapsack profit = %f\n",profit);
    printf("Continuous Knapsack also includes item %d with portion: %f\n", k, (float)m/w[k]);
    profit = profit + ((float)m)/w[k] * p[k];
    printf("Continuous Knapsack profit = %f\n",profit);
}

int main()
{
    int i;
    printf("Enter the no. of items: ");
    scanf("%d",&n);
    printf("Enter the weights of n items: ");
    for(i=0;i<n;i++)
        scanf("%d",&w[i]);
    printf("Enter the prices of n items: ");
    for(i=0;i<n;i++)
        scanf("%d",&p[i]);
    printf("Enter the capacity of Knapsack: ");
```

```
scanf("%d",&m);  
greedy_knapsack();  
}
```

Sample Input and Output:

Enter the no. of items: 4

Enter the weights of n items: 2 1 3 2

Enter the prices of n items: 12 10 20 15

Enter the capacity of Knapsack: 5

item included is :1 3

Discrete Knapsack profit = 25.000000

Continuous Knapsack also includes item 2 with portion: 0.666667

Continuous Knapsack profit = 38.333332

Dept. of ISE, RNSIT

8. Design and implement C/C++ Program to find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d .

```
#include<stdio.h>

int x[10], w[10], count, d;
void sum_of_subsets(int s, int k, int rem)
{
    x[k] = 1;
    if( s + w[k] == d)
    {
        //if subset found
        printf("subset = %d\n", ++count);
        for(int i=0 ; i <= k ; i++)
            if ( x[i] == 1)
                printf("%d ",w[i]);
        printf("\n");
    }
    else if ( s + w[k] + w[k+1] <= d )//left tree evaluation
        sum_of_subsets(s+w[k], k+1, rem-w[k]);

    if( ( s+rem-w[k] >= d) && ( s + w[k+1]) <= d)//right tree evaluation
    {
        x[k] = 0;
        sum_of_subsets(s,k+1,rem-w[k]);
    }
}

int main()
{
    int sum = 0,n;
    printf("enter no of elements:");
    scanf("%d",&n);
    printf("enter the elements in increasing order:");
    for( int i = 0; i < n ; i++)
    {
        scanf("%d",&w[i]);
        sum=sum+w[i];
    }
    printf("eneter the sum:");
    scanf("%d",&d);

    if ( ( sum < d ) || ( w[0] > d ) )
        printf("No subset possible\n");
    else
        sum_of_subsets(0,0,sum);
}
```

Sample Input and Output:

```
enter no of elements:5
enter the elements in increasing order:1 2 3 4 5
eneter the sum:10
subset = 1
```

1 2 3 4
subset = 2
1 4 5
subset = 3
2 3 5

Dept. of ISE, RNSIT

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.

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

int a[10000],n,count;

void selection_sort()
{
    for(int i=0;i<n-1;i++)
    {
        int min = i;
        for(int j=i+1;j<n;j++)
        {
            count++;
            if(a[j]<a[min])
                min=j;
        }
        int temp=a[i];
        a[i]=a[min];
        a[min]=temp;
    }
}

int main()
{
    printf("Enter the number of elements in an array:");
    scanf("%d",&n);
    printf("All the elements:");
    srand(time(0));
    for(int i=0;i<n;i++)
    {
        a[i]=rand();
        printf("%d ",a[i]);
    }
    selection_sort();
    printf("\nAfter sorting\n");
    for(int i=0;i<n;i++)
        printf("%d ", a[i]);
    printf("\nNumber of basic operations = %d\n",count);
}
```

Sample Input and Output:

Enter the number of elements in an array:5

All the elements:

24152 32742 28304 4804 22274

After sorting

4804 22274 24152 28304 32742

Number of basic operations = 10

Enter the number of elements in an array:10

All the elements:

24243 6017 4212 23217 16170 24802 1085 24280 9847 6392

After sorting

1085 4212 6017 6392 9847 16170 23217 24243 24280 24802

Number of basic operations = 45

Dept. of ISE, RNSIT

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.

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

int count=0;
int partition(int a[], int low,int high)
{
    int pivot=a[low],temp,i=low+1,j=high;
    while(1)
    {
        //Traverse i from left to right, segregating element of left group
        while(i<=high && a[i]<=pivot)//a[i]<=pivot used for avoiding multiple duplicates
        {
            i++; count++;
        }
        //Traverse j from right to left, segregating element of right group
        while(j>0 && a[j]>pivot)
        {
            j--; count++;
        }
        count+=2;
        //If grouping is incomplete
        if(i<j)
        {
            temp = a[i];
            a[i] = a[j];
            a[j] =temp;
        }
        else if(i>j)//If grouping is completed
        {
            temp = a[low];
            a[low] = a[j];
            a[j] = temp;
            return j;
        }
        else //Duplicate of Pivot found
            return j;
    }
}

void quicksort(int a[],int low, int high)
{
    int s;
    if(low<high)
    {
        //partition to place pivot element in between left and right group
        s = partition(a,low,high);
```

```

        quicksort(a,low,s-1);
        quicksort(a,s+1,high);
    }
}
int main()
{
    int a[10000],n;
    printf("Enter the number of elements in an array:");
    scanf("%d",&n);
    printf("All the elements:");
    srand(time(0));
    for(int i=0;i<n;i++)
    {
        a[i]=rand();
        printf("%d ",a[i]);
    }
    quicksort(a,0,n-1);
    printf("\nAfter sorting\n");
    for(int i=0;i<n;i++)
        printf("%d ", a[i]);
    printf("\nNumber of basic operations = %d\n",count);
}

```

Sample Input and Output:

Enter the number of elements in an array:5

All the elements:

24442 6310 12583 16519 22767

After sorting

6310 12583 16519 22767 24442

Number of basic operations = 18

Enter the number of elements in an array:10

All the elements:

24530 1605 3396 10868 6349 9906 12836 28823 21075 22418

After sorting

1605 3396 6349 9906 10868 12836 21075 22418 24530 28823

Number of basic operations = 44

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.

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

int count=0;
void merge(int a[], int low,int mid,int high)
{
    int i,j,k,c[10000];

    i=low, j=mid+1, k=0;
    while((i<=mid) && (j<=high))
    {
        count++;
        //choose the least element and store in Temporary array 'C'
        if(a[i]<a[j])
            c[k++]=a[i++];
        else
            c[k++]=a[j++];
    }
    //Copy the remaining array elements from any one of sub-array
    while(i<=mid)
        c[k++]=a[i++];
    while(j<=high)
        c[k++]=a[j++];
    for(i=low,j=0;j<k;i++, j++)
        a[i]=c[j];
}

void merge_sort(int a[], int low, int high)
{
    int mid;
    if(low < high)
    {
        //Divide the given array into 2 parts
        mid=(low+high)/2;
        merge_sort(a,low,mid);
        merge_sort(a,mid+1,high);
        merge(a,low,mid,high);
    }
}

int main()
{
    int a[10000],n,i;
    printf("Enter the number of elements in an array:");
    scanf("%d",&n);
    printf("All the elements:");
```

```

srand(time(0));
for(i=0;i<n;i++)
{
    a[i]=rand();
    printf("%d ",a[i]);
}
merge_sort(a,0,n-1);
printf("\nAfter sorting\n");
for(i=0;i<n;i++)
    printf("%d ", a[i]);
printf("\nNumber of basic operations = %d\n",count);
}

```

Sample Input and Output:

Enter the number of elements in an array:5

All the elements:

24759 329 8704 24132 7473

After sorting

329 7473 8704 24132 24759

Number of basic operations = 8

Enter the number of elements in an array:10

All the elements:

24854 17121 2477 1072 11684 5437 26057 1167 17322 3583

After sorting

1072 1167 2477 3583 5437 11684 17121 17322 24854 26057

Number of basic operations = 22

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

```
#include<stdio.h>
#include<math.h>    //for abs() function

int place(int x[],int k)
{
    for(int i=1;i<k;i++)
    {
        if( (x[i] == x[k]) || ( abs(x[i]-x[k]) == abs(i-k)) )
            return 0;
    }
    return 1; //feasible
}

int nqueens(int n)
{
    int x[10], k, count=0;

    k=1; // select the first queen
    x[k]=0; //no positions allocated
    while(k != 0) // until all queens are present
    {
        x[k]++; // place the kth queen in next column
        while((x[k] <= n) && (!place(x,k)))
            x[k]++; // check for the next column to place queen

        if(x[k] <= n)
        {
            if(k == n) // all queens are placed
            {
                printf("\nSolution %d\n",++count);
                for(int i=1;i <= n;i++)
                {
                    for(int j=1;j <= n;j++)
                        printf("%c ",j==x[i]?'Q':'X');
                    printf("\n");
                }
            }
            else
            {
                ++k; //select the next queen
                x[k]=0; // start from the next column
            }
        }
        else
        {
            k--; // backtrack
        }
    }
    return count;
}

void main()
```

```

{
    int n;
    printf("Enter the size of chessboard: ");
    scanf("%d",&n);
    printf("\nThe number of possibilities are %d",nqueens(n));
}

```

Sample Input and Output:

1. Enter the size of chessboard: 4

Solution 1

```

X Q X X
X X X Q
Q X X X
X X Q X

```

Solution 2

```

X X Q X
Q X X X
X X X Q
X Q X X

```

The number of possibilities are 2

2. Enter the size of chessboard: 3

The number of possibilities are 0

3. Enter the size of chessboard: 1

Solution 1

```

Q

```

The number of possibilities are 1