VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

JAYASHREE TARAI (1BM24CS407)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
February-May 2025

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"ANALYSIS AND DESIGN OF ALGORITHMS"** carried out by **JAYASHREE TARAI (1BM24CS407),**who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and Design of Algorithms Lab - **(23CS4PCADA)** work prescribed for the said degree.

RAMYA K M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

GITHUB LINK :https://github.com/Jayashreecse/ADA-LAB/tree /main

Course Outcomes:

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

Lab program 1.1:

Write program to obtain the Topological ordering of vertices in a given digraph.

Program full details

Code

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100

int graph[MAX][MAX];
bool visited[MAX];
int stack[MAX];
int top = -1;
int n;

void push(int v) {
   stack[++top] = v;
}
void dfs(int node) {
   visited[node] = true;
   for (int i = 0; i < n; i++) {
      if (graph[node][i] == 1 && !visited[i]) {
         dfs(i);
      }
   }
```

```
    push(node);
}

void topologicalSort() {

    for (int i = 0; i < n; i++) {
        visited[i] = false;
    }

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    printf("Topological Order: ");
    while (top != -1) {
        printf("%d ", stack[top--]);
    }
    printf("\n");
}

int main() {

    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (0 or 1):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
```

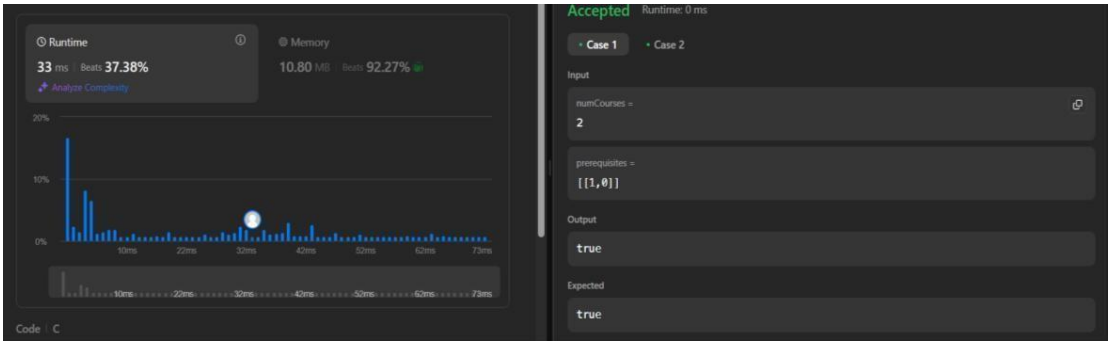
```
            scanf("%d", &graph[i][j]);
        }
    }
    topologicalSort();
    return 0;
}
```

```
Enter number of vertices: 5
Enter the adjacency matrix (0 or 1):
0 1 0 0 0
0 0 1 1 0
0 0 0 1 0
0 0 0 0 0
0 1 0 0 0
Topological Order: 4 0 1 2 3
```

Screenshot of Output

Lab program 1.2:

```
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<vector<int>> graph(numCourses);
        vector<int> indegree(numCourses, 0);
        for (const auto& pre : prerequisites) {
            graph[pre[1]].push_back(pre[0]);
            indegree[pre[0]]++;
        }
        queue<int> q;
        for (int i = 0; i < numCourses; ++i) {
            if (indegree[i] == 0) q.push(i);
        }
        int count = 0;
        while (!q.empty()) {
            int curr = q.front(); q.pop();
            count++;
            for (int next : graph[curr]) {
                indegree[next]--;
                if (indegree[next] == 0) q.push(next);
            }
        }
        return count == numCourses;
    }
};
```



Lab program 2:

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Code

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

void merge(int arr[], int left, int right, int mid) {
   int i, j, k;
   int n1 = mid - left + 1;
   int n2 = right - mid;
   int L[n1], R[n2];

   for(i = 0; i < n1; i++) {
      L[i] = arr[left + i];
   }
   for(j = 0; j < n2; j++) {
      R[j] = arr[mid + 1 + j];
   }

   i = 0;
   j = 0;
```

```
    k = left;

    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 left, int right) {
    if(left < right) {
        int mid = left + (right - left) / 2;
```

```
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, right, mid);
    }
}

void print(int arr[], int size) {
    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}

int main() {
    int n;
    clock_t start, end;


    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    srand(time(NULL));


    for(int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
```

```
    }


    printf("Original Array: ");
    print(arr, n);

    start = clock();


    mergeSort(arr, 0, n - 1);


    end = clock();


    printf("Sorted Array: ");
    print(arr, n);

    printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);




    return 0;
}
```

Screenshot of Output

```
 921 921 921 921 921 921 922 922 922 922 922 922 922 922 922 922 923 923 923 923 923 923 923 924 924 924 924 924 924 924
 924 924 924 925 925 925 925 925 925 925 925 925 926 926 926 926 927 927 927 927 927 927 928 928 928 928 928 928 928 928
 928 928 929 929 929 929 929 929 929 929 929 930 930 930 930 930 930 930 930 930 931 931 931 931 931 931 931 931 931 931
 931 931 932 932 932 932 932 932 932 932 932 933 933 933 933 933 933 933 933 933 933 933 934 934 934 934 934 934 934 934 934
 934 935 935 935 935 935 935 935 935 936 936 936 936 936 936 936 936 936 936 936 936 936 936 937 937 937 937 937 937 937
 937 937 937 937 938 938 938 938 938 938 938 938 939 939 939 939 939 939 939 939 939 939 939 940 940 940 940 940 940 940 940
 941 941 941 941 941 941 941 941 941 941 942 942 942 942 942 942 942 942 942 942 943 943 943 943 943 944 944 944 944 944
 944 944 944 944 944 944 944 945 945 945 945 945 945 945 945 946 946 946 946 947 947 947 947 947 948 948 948 948 948 948
 948 948 948 949 949 949 949 949 949 949 950 950 950 950 950 950 950 950 951 951 951 951 951 951 951 951 951 951 951 952
 952 952 952 952 952 952 952 952 952 953 953 953 953 953 953 953 953 953 953 953 954 954 954 954 954 954 954 954 955 955
 955 955 955 955 955 955 955 956 956 956 956 956 956 956 956 956 956 956 956 956 956 957 957 957 957 957 957 957 957 957
 957 957 957 958 958 958 958 958 958 958 958 959 959 959 959 959 959 959 959 959 959 960 960 960 960 960 960 960 961 961
 961 961 961 961 961 961 961 961 961 962 962 962 962 962 962 962 963 963 963 963 963 963 963 963 963 963 963 963 963 964
 964 964 964 964 964 964 964 964 964 964 965 965 965 965 965 965 965 965 965 965 966 966 966 966 966 966 966 966 966 966
 967 967 967 967 967 967 967 967 967 968 968 968 968 968 968 968 968 968 969 969 969 969 969 969 969 969 969 969 969 970 970
 970 970 970 970 970 970 970 971 971 971 971 971 971 971 971 971 971 971 972 972 972 972 972 972 972 972 972 972 972 972 973
 973 973 973 973 973 973 973 974 974 974 974 974 974 974 974 974 974 974 974 975 975 975 975 975 975 975 975 975 975 975 975
 976 976 976 976 976 976 976 977 977 977 978 978 978 978 978 978 978 978 978 978 979 979 979 979 979 979 979 979 979 979
 979 979 979 979 980 980 980 980 980 980 980 980 980 980 980 980 980 981 981 981 981 981 981 981 981 981 981 981 982 982 982
 982 982 982 983 983 983 983 983 983 983 983 983 983 984 984 984 985 985 985 985 985 985 986 986 986 986 986 986 986 986
 986 987 987 987 987 987 987 987 987 988 988 988 988 988 988 988 988 988 989 989 989 989 989 989 989 989 989 989 989 990 990
 990 990 990 990 990 991 991 991 991 991 991 991 991 991 991 991 991 991 991 992 992 992 992 992 992 992 992 992 992 993 993 993
 993 994 994 994 994 994 994 994 994 994 994 994 994 994 994 995 995 995 995 995 995 995 995 995 995 995 996 996 996 996 996 996
 996 997 997 997 997 997 997 997 997 998 998 998 998 998 998 998 998 998 998 998 998 999 999 999 999 999 999 999 999 999 999
 999 999
Time taken: 1.000000 seconds
```

Lab program 3:

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

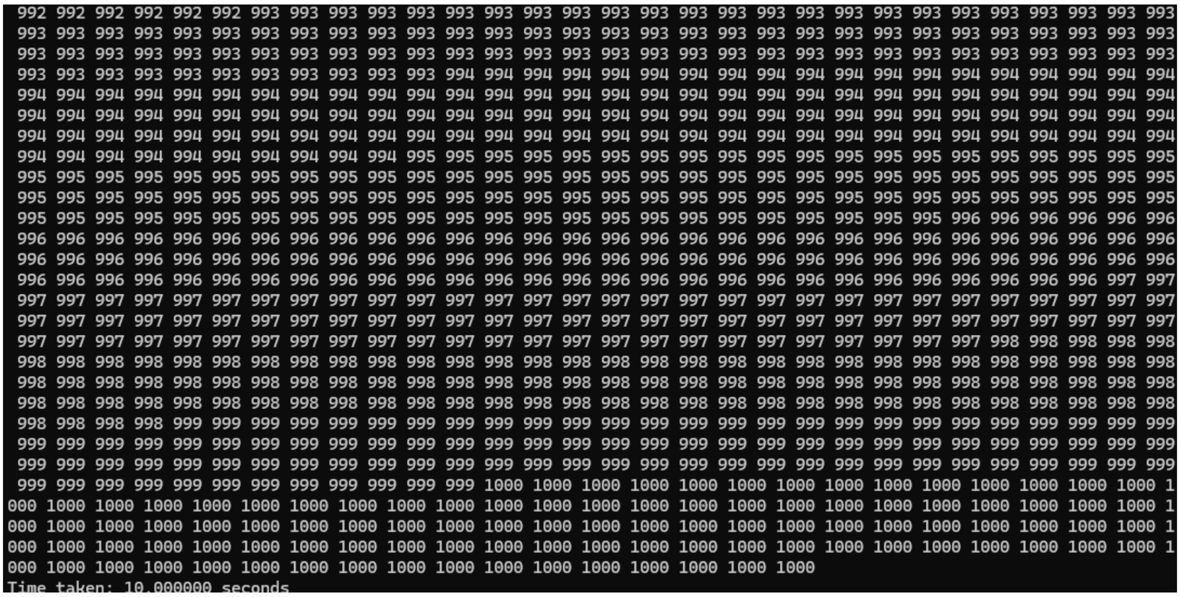
Code

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



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++;


            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
```

```
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    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);
    }
}


void print(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```
int main() {
    int n;
    clock_t start, end;

    printf("Enter the number of elements in the array: ");
    scanf("%d", &n);

    int arr[n];

    srand(time(NULL));

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1001;
    }

    printf("Original Array: ");
    print(arr, n);

    start = clock();

    quickSort(arr, 0, n - 1);
```

```
    end = clock();



    printf("Sorted Array: ");
    print(arr, n);



    printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);





    return 0;
}
```



Screenshot of Outpu

Lab program 4:

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

Code

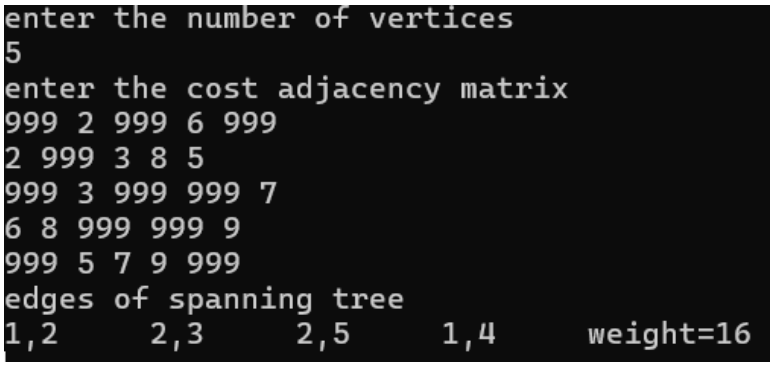
```
#include<stdio.
```

```
#include<conio.h>

  int cost[10][10],vt[10],et[10][10],vis[10],j,n;
  int sum=0;
  int x=1;
  int e=0;
  void prims();

  void main()
  {
    int i;

    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]);
       }
       vis[i]=0;
    }
    prims();
    printf("edges of spanning tree\n");
    for(i=1;i<=e;i++)
    {
          printf("%d,%d\t",et[i][0],et[i][1]);
    }
```

```
    printf("weight=%d\n",sum);
    getch();
}

void prims()
{
   int s,min,m,k,u,v;
   vt[x]=1;
   vis[x]=1;
   for(s=1;s<n;s++)
   {
      j=x;
      min=999;
      while(j>0)
      {
            k=vt[j];
            for(m=2;m<=n;m++)
            {
              if(vis[m]==0)
              {
                    if(cost[k][m]<min)

           j--;
```

```
    }
    vt[++x]=v;
    et[s][0]=u;
    et[s][1]=v;
    e++;
    vis[v]=1;
    sum=sum+min;
  }
}
```

Screenshot of Output



Lab program 5:

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

Code

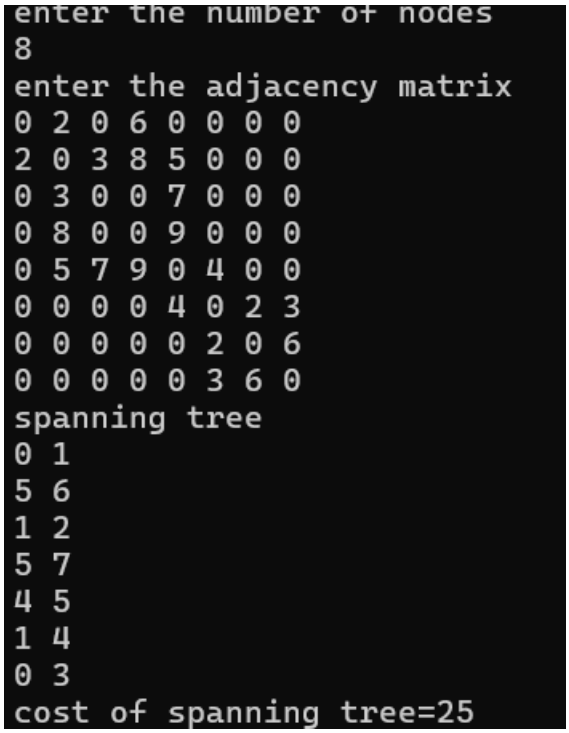
```
#include<stdio.h>
#include<conio.h>


int find(int v,int parent[10])
{
  while(parent[v]!=v)
  {
    v=parent[v];
  }
  return v;
```

```
}

void union1(int i,int j,int parent[10])
{
  if(i<j)
    parent[j]=i;
  else
    parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
  int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
  count=0;
  k=0;
  sum=0;
  for(i=0;i<n;i++)
    parent[i]=i;
  while(count!=n-1)
  {
    min=999;
    for(i=0;i<n;i++)
    {
          for(j=0;j<n;j++)
          {

            if(a[i][j]<min && a[i][j]!=0)
            {
                  min=a[i][j];
                  u=i;
                  v=j;                                .
             }
```

```
            }
        }
        i=find(u,parent);
        j=find(v,parent);
        if(i!=j)
        {
            union1(i,j,parent);
            t[k][0]=u;
            t[k][1]=v;
            k++;
            count++;
            sum=sum+a[u][v];
        }
        a[u][v]=a[v][u]=999;
    }
    if(count==n-1)
    {
        printf("spanning tree\n");
        for(i=0;i<n-1;i++)
        {
            printf("%d %d\n",t[i][0],t[i][1]);
        }
        printf("cost of spanning tree=%d\n",sum);
    }
    else
        printf("spanning tree does not exist\n");
  }


void main()
{
   int n,i,j,a[10][10];
```

```
  clrscr();
  printf("enter the number of nodes\n");
  scanf("%d",&n);
  printf("enter the adjacency matrix\n");
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
          scanf("%d",&a[i][j]);
  kruskal(n,a);
  getch();
}
```

Screenshot of Output



Lab program 6:

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

Code

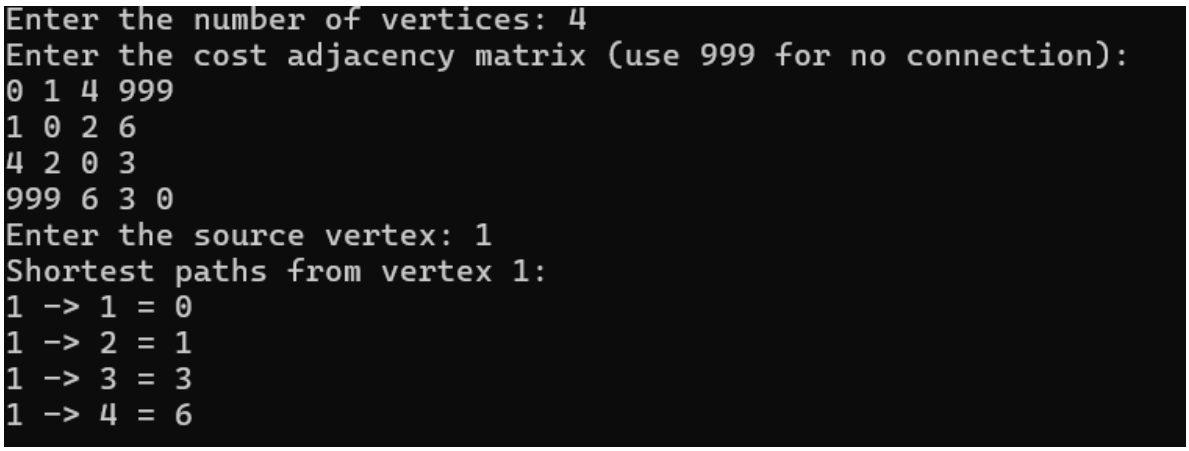
```
#include <stdio.h>
```

```
#define INF 999

void dijkstra(int n, int cost[10][10], int src) {
    int i, j, u, dis[10], vis[10], min;

    // Initialize distances and visited flags
    for (i = 1; i <= n; i++) {
        dis[i] = cost[src][i];
        vis[i] = 0;
    }

    vis[src] = 1;

    for (i = 1; i < n; i++) {
        min = INF;
        u = -1;

        // Find the unvisited vertex with the smallest distance
        for (j = 1; j <= n; j++) {
            if (vis[j] == 0 && dis[j] < min) {
                min = dis[j];
                u = j;
            }
        }

        if (u == -1) break; // All reachable vertices visited

        vis[u] = 1;
```

```
        // Update distances to neighboring vertices
        for (j = 1; j <= n; j++) {
            if (vis[j] == 0 && dis[u] + cost[u][j] < dis[j]) {
                dis[j] = dis[u] + cost[u][j];
            }
        }
    }

    printf("Shortest paths from vertex %d:\n", src);
    for (i = 1; i <= n; i++) {
        if (dis[i] == INF)
            printf("%d -> %d = INF\n", src, i);
        else
            printf("%d -> %d = %d\n", src, i, dis[i]);
    }
}

int main() {
    int src, j, cost[10][10], n, i;

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

    printf("Enter the cost adjacency matrix (use 999 for no connection):\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
        }
```

```
    }

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

    dijkstra(n, cost, src);

    return 0;
}
```

Screenshot of Output



Lab program 7:

Implement Johnson Trotter algorithm to generate permutations.

Code

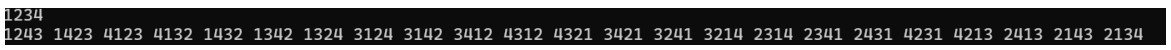
```
#include <stdio.h>


#define LEFT_TO_RIGHT 1
#define RIGHT_TO_LEFT 0
```

```
int searchArr(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++)
        if (a[i] == mobile)
            return i + 1;
    return -1;
}

int getMobile(int a[], int dir[], int n) {
    int mobile_prev = 0, mobile = 0;

    for (int i = 0; i < n; i++) {
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
            if (a[i] > a[i - 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
            if (a[i] > a[i + 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    return mobile;
}

void printOnePerm(int a[], int dir[], int n) {
```

```
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    if (mobile == 0) return;

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            dir[a[i] - 1] = !dir[a[i] - 1]; // toggle direction
        }
    }

    for (int i = 0; i < n; i++)
        printf("%d", a[i]);
    printf(" ");
}

int fact(int n) {
    int res = 1;
    for (int i = 1; i <= n; i++)
```

```
        res = res * i;
    return res;
}

void printPermutation(int n) {
    int a[n], dir[n];

    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d", a[i]);
    }
    printf("\n");

    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

int main() {
    int n = 4;
    printPermutation(n);
    return 0;
}
```

Screenshot of Output



Lab

program 8.1:

Implement Fractional Knapsack using Greedy technique.

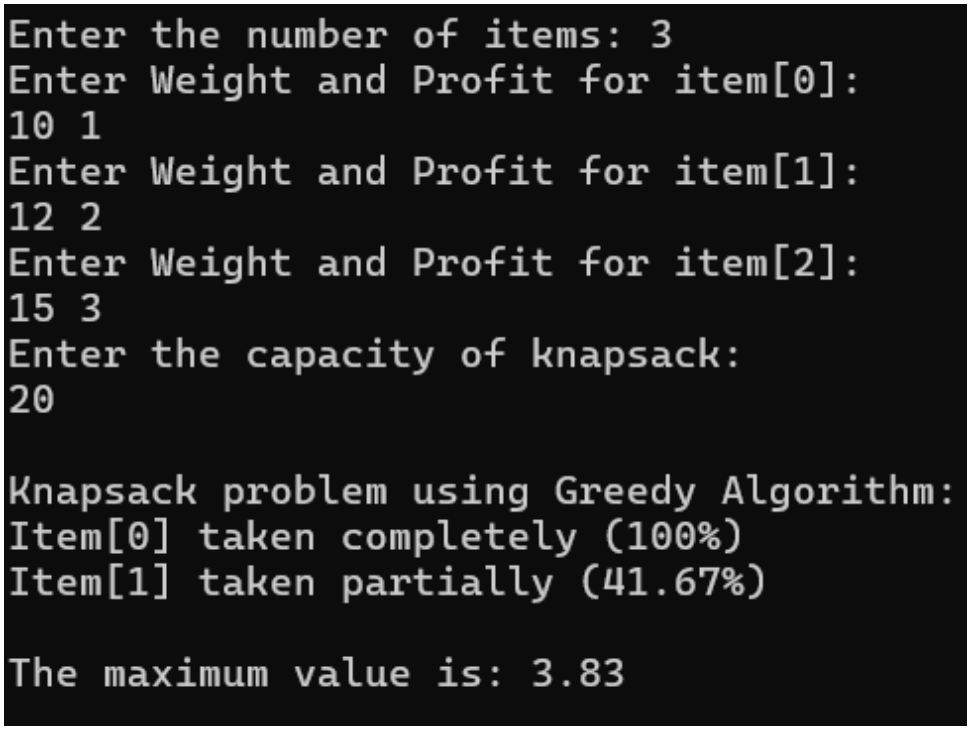
Code

```
#include <stdio.h>


int main() {
    float weight[50], profit[50], ratio[50];
    float Totalvalue = 0.0, temp, capacity, amount;
    int n, i, j;


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


    for (i = 0; i < n; i++) {
        printf("Enter Weight and Profit for item[%d]:\n", i);
        scanf("%f %f", &weight[i], &profit[i]);
    }


    printf("Enter the capacity of knapsack:\n");
    scanf("%f", &capacity);


    // Calculate profit/weight ratio
    for (i = 0; i < n; i++)
        ratio[i] = profit[i] / weight[i];


    // Sort items by descending ratio
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (ratio[i] < ratio[j]) {
```

```
            // Swap ratio
            temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            // Swap weight
            temp = weight[i];
            weight[i] = weight[j];
            weight[j] = temp;

            // Swap profit
            temp = profit[i];
            profit[i] = profit[j];
            profit[j] = temp;
        }
    }
}

printf("\nKnapsack problem using Greedy Algorithm:\n");
for (i = 0; i < n; i++) {
    if (weight[i] <= capacity) {
        // Take full item
        printf("Item[%d] taken completely (100%%)\n", i);
        Totalvalue += profit[i];
        capacity -= weight[i];
    } else {
        // Take fraction of item
        float fraction = capacity / weight[i];
        Totalvalue += profit[i] * fraction;
```

```
            printf("Item[%d] taken partially (%.2f%%)\n", i, fraction * 100);
            break; // Knapsack is now full
        }
    }

    printf("\nThe maximum value is: %.2f\n", Totalvalue);
    return 0;
}
```

Screenshot of Output



Lab program 8.2:

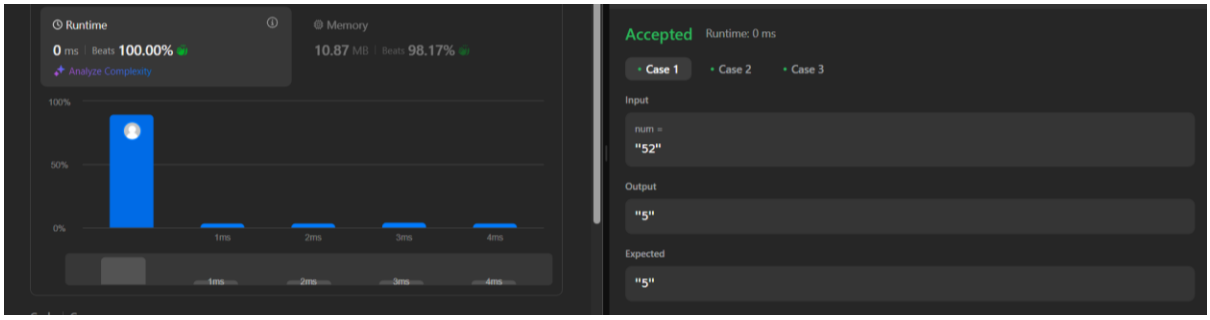
LeetCode Program related to Greedy Technique algorithms

Code

```
char* largestOddNumber(char* num) {
  int len = strlen(num);
```

```
    for (int i = len - 1; i >= 0; i--) {
        if ((num[i] - '0') % 2 == 1) {
            num[i + 1] = '\0'; // Truncate string at that position
            return num; // Return the longest odd-suffix (greedy)
        }
    }

    return ""; // No odd digit found
}
```

Screenshot of Output



Lab program 9.1:

Implement 0/1 Knapsack problem using dynamic programming.

Code

```
#include <stdio.h>

// Function to return the maximum of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve the 0/1 Knapsack problem
```

```
int knapsack(int weight[], int profit[], int n, int capacity) {
    int i, w;
    int K[n + 1][capacity + 1];

    // Build the DP table K[][] bottom up
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (weight[i - 1] <= w)
                K[i][w] = max(profit[i - 1] + K[i - 1][w - weight[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // Optional: Print the items included
    printf("\nItems included:\n");
    w = capacity;
    for (i = n; i > 0 && w > 0; i--) {
        if (K[i][w] != K[i - 1][w]) {
            printf("Item %d (Weight: %d, Profit: %d)\n", i, weight[i - 1], profit[i - 1]);
            w -= weight[i - 1];
        }
    }

    return K[n][capacity];
}
```

```
int main() {
    int n, capacity;
    int weight[50], profit[50];
    int i;

    printf("Enter number of items: ");
    scanf("%d", &n);

    printf("Enter weight and profit for each item:\n");
    for (i = 0; i < n; i++) {
        printf("Item[%d] - Weight Profit: ", i + 1);
        scanf("%d %d", &weight[i], &profit[i]);
    }

    printf("Enter the capacity of knapsack: ");
    scanf("%d", &capacity);

    int maxProfit = knapsack(weight, profit, n, capacity);

    printf("\nMaximum profit: %d\n", maxProfit);
    return 0;
}
```

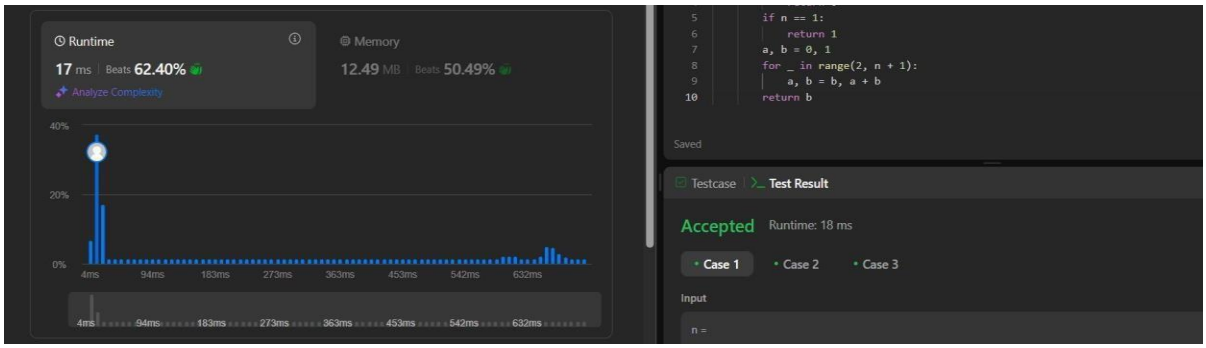
Screenshot of Output

Lab program 9.2:

Code

```
class Solution(object):
    def fib(self, n):
        if n == 0:
            return 0
        if n == 1:
            return 1
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b
```

Screenshot of Output



Lab program 10:

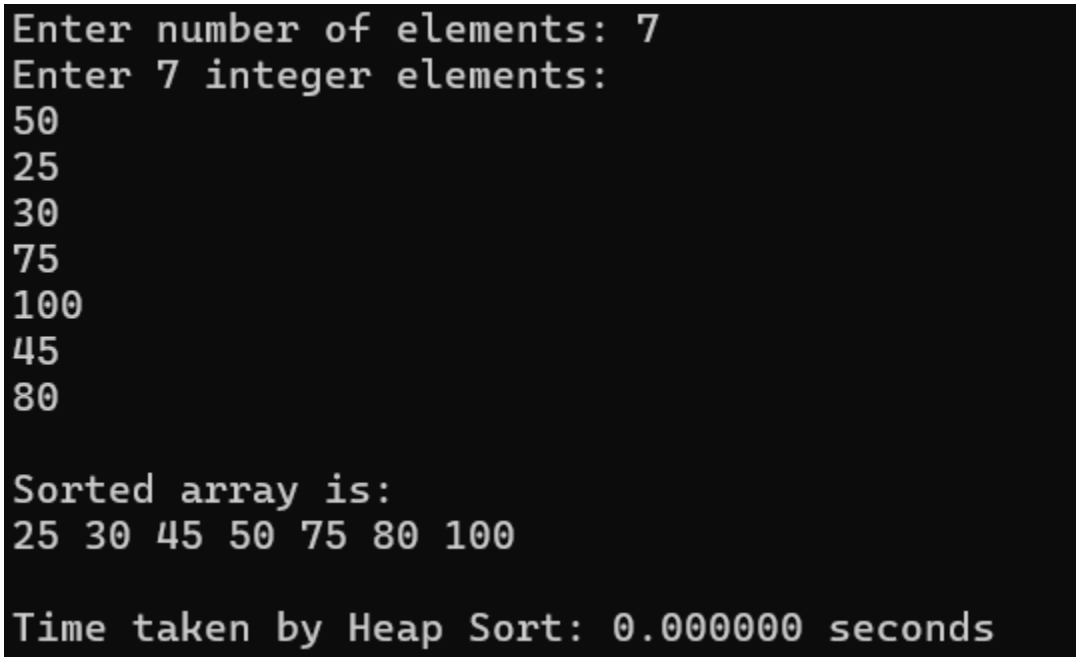
Sort a given set of N integer elements using Heap Sort technique and compute its time taken

Code

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


void heapify(int arr[], int n, int i) {
   int largest = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;


   if (left < n && arr[left] > arr[largest])
      largest = left;


   if (right < n && arr[right] > arr[largest])
      largest = right;


   if (largest != i) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;


      heapify(arr, n, largest);
   }
}

void heapSort(int arr[], int n) {
   for (int i = n / 2 - 1; i >= 0; i--)
      heapify(arr, n, i);

   for (int i = n - 1; i >= 0; i--) {
      int temp = arr[0];
      arr[0] = arr[i];
      arr[i] = temp;


      heapify(arr, i, 0);
   }
}
```

```
int main() {
    int arr[1000], n;
    clock_t start, end;
    double time_taken;

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

    printf("Enter %d integer elements:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    start = clock();

    heapSort(arr, n);

    end = clock();
    time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\nSorted array is:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    printf("\n\nTime taken by Heap Sort: %f seconds\n", time_taken);

    return 0;
}
```

Screenshot of Outpu



Lab program 11.1:

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

Code

```
#include <stdio.h>

#define INF 99999 // Use a large number to represent infinity
#define MAX 100

void floydWarshall(int graph[MAX][MAX], int n) {
    int dist[MAX][MAX];
    int i, j, k;

    // Initialize the solution matrix same as input graph
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            dist[i][j] = graph[i][j];

    // Floyd-Warshall algorithm
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the final shortest distance matrix
```

```
    printf("\nAll-Pairs Shortest Paths (Floyd-Warshall):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", dist[i][j]);
        }
        printf("\n");
    }
}

int main() {
    int graph[MAX][MAX], n;

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

    printf("Enter the adjacency matrix (use 99999 for no direct path):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall(graph, n);

    return 0;
}
```

Screenshot of Output

```
Enter number of vertices: 4
Enter the adjacency matrix (use 99999 for no direct path):
0 4 3 9
99 0 1 99
99 990 99999
5 2 6 0
2 99 99999 99999

All-Pairs Shortest Paths (Floyd-Warshall):
  0   4   3   8
  8   0   1   6
  7  11   5   5
  2   6   0   2
```

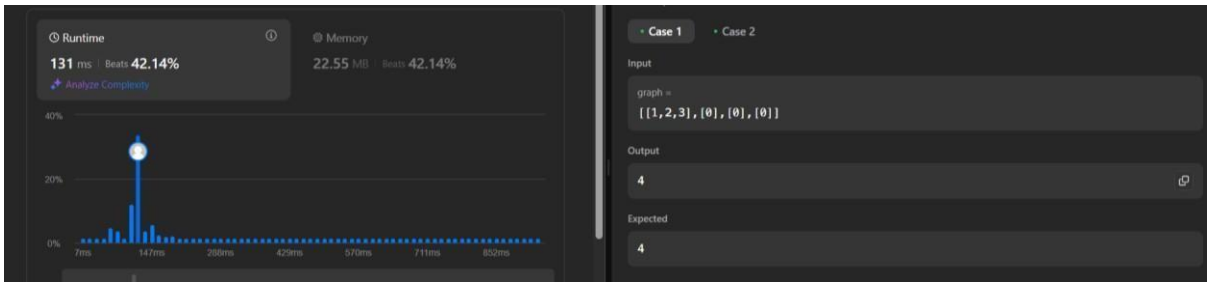
Lab program 11.2:

LeetCode Program related to shortest distance calculation

Code

```
class Solution
    def shortestPathLength(self, graph: List[List[int]]) -> int:
        n=len(graph)
        queue=deque([(i,1<<i) for i in range(n)])
        seen=set(queue)
        ans=0
        while queue:
            for _ in range(len(queue)):
                u,m=queue.popleft()
                if m==(1<<n)-1:
                    return ans
                for v in graph[u]:
                    if (v,m|1<<v) not in seen:
                        queue.append((v,m|1<<v))
                        seen.add((v,m|1<<v))
            ans+=1
```

Screenshot of Output



Lab program 12:

Implement “N-Queens Problem” using Backtracking.

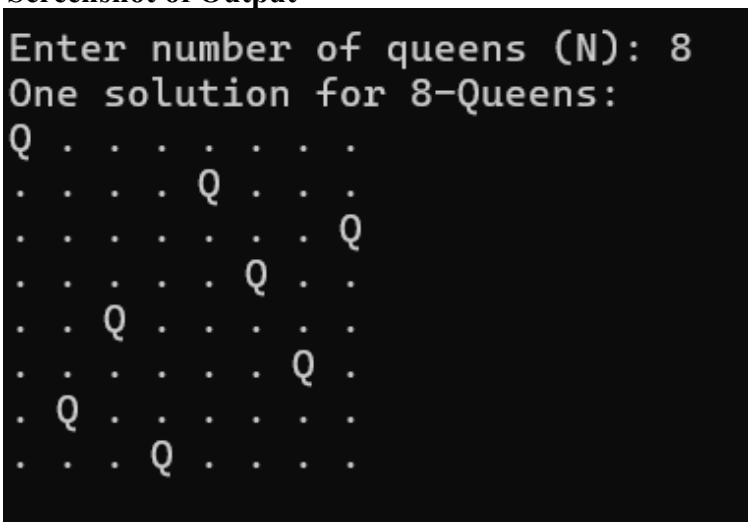
Code

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

#define MAX 20

int board[MAX];
int found = 0;

// Function to print one solution
void printSolution(int n) {
    printf("One solution for %d-Queens:\n", n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (board[i] == j)
                printf("Q ");
            else
```

```
            printf(". ");
        }
        printf("\n");
    }
    found = 1;
}

// Check if placing queen at (k, i) is safe
int isSafe(int k, int i) {
    for (int j = 1; j < k; j++) {
        if (board[j] == i || fabs(board[j] - i) == fabs(j - k))
            return 0;
    }
    return 1;
}

// Recursive backtracking to find one solution
void nQueens(int k, int n) {
    for (int i = 1; i <= n && !found; i++) {
        if (isSafe(k, i)) {
            board[k] = i;
            if (k == n)
                printSolution(n);
            else
                nQueens(k + 1, n);
        }
    }
}
```

```
int main() {
    int n;
    printf("Enter number of queens (N): ");
    scanf("%d", &n);

    if (n < 1 || n > MAX) {
        printf("Please enter N between 1 and %d.\n", MAX);
        return 1;
    }

    nQueens(1, n);

    if (!found)
        printf("No solution exists for N = %d\n", n);

    return 0;
}
```

Screenshot of Output



21/8/2025

LAB PROGRAM-1

Sort a given set of N integer elements Using Merge sort Technique and Compute its time taken. Run the Program for different values of N and Record the time take to sort

Merge Sort → Divide (divide until single elements)
conquer (Each sub array is sorted)
combine (Merge the sub-arrays together)

Time Complexity → O(n log n)
Space Complexity : O(n)
Recurrance relationship :

$$T(n) \begin{cases} O(1) & \text{if } n=1 \\ 2T(n/2) + O(n) & \text{if } n>1 \end{cases}$$

Program

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

void merge(int arr[], int left, int mid, int right){
    int n1 = mid - left + 1
    int n2 = right - mid;
    int leftArr[n1], rightArr[n2];

    for(int i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for(int i = 0; i < n2; i++)
        rightArr[i] = arr[mid + 1 + i];
    int i = 0, j = 0, k = left;
```

```
while (i<n1 && j<n2){
    if (leftArr[i] <= rightArr[j]) {
        arr[k] = leftArr[i];
        i++;
    } else{
        arr[k] = rightArr[j];
        j++;
    }
    k++;
}

while (i<n1) {
    arr[k] = leftArr[i];
    i++;
    k++;
}

while (j<n2) {
    arr[k] = rightArr[j];
    j++;
    k++;
}

void mergeSort(int arr[], int left, int right) {
    if (left<right) {
    int mid = left + (right - left) /2
    mergesort(arr, left, mid);
    mergeSort(arr, mid+1, right);

    merge(arr, left, mid, right);
}
```

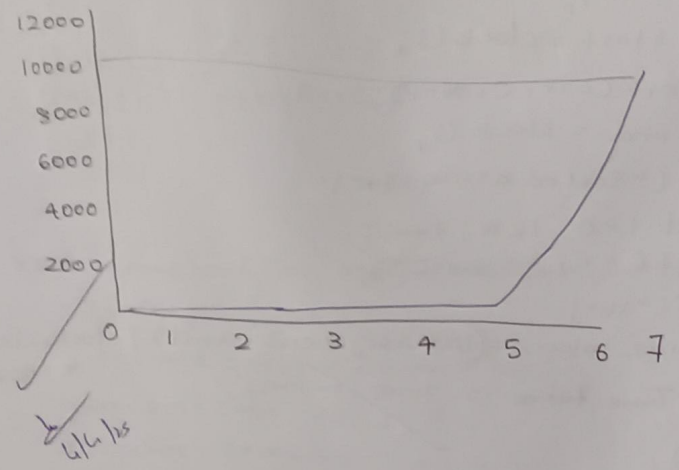
```
int main() {
    int N;
    printf("Enter the no of elements (N): ");
    scanf("%d", &N);

    int arr[N]

    srand(time(NULL));
    for(int i=0; i<N; i++) {
        arr[i] = rand() %1000
    }
    printf("Original orginal array: \n");
    for(int i=0; i<N; i++) {
        printf("%d", arr[i]);
    }
    printf("\n");
    clock_t start = clock();
    mergesort(arr, 0, N-1);
    clock_t end = clock();
    printf("sorted array: \n");
    for(int i=0; i<N; i++) {
        printf("%d", arr[i]);
    printf("\n")
double time_taken = ((double)(end-start) / clocks_per_sec)
printf("Time taken is. time_taken);          * 1000
    return 0;
    }
```

Output

Enter the no of elements:

Input size N	Time taken to sort (milliseconds)
10	0.00s
50	0.0s
100	0.6s
500	1.02
1000	3.2
5000	18.8
10000	43.2

Chart



4/4/25

Quick Sort

(Running the program for different values of N)

→ Quicksort → takes an pivot element arrange such a way left side of pivot has less than element & Right side of the pivot has greater elements

Algorithm

```
Quick sort (array, leftmostindex, rightmost index)
if (leftmostindex < rightmost index)
    pivotelement ← partition (array, leftmost_index,
                              rightmost index)

    Quicksort (array, leftmostindex, pivotindex -1)
    Quick sort (array, pivotindex, rightmost index)

Partition (array, leftmost, rightmost index)
    Set rightmostindex as pivotIndex
    StoreIndex ← leftmostIndex - 1
    for i ← leftmost index + 1 to rightmostindex
    if element [i] < pivot element
        Swap element [i] and element [store index]
        Storeindex++
        Swap pivotElement and element [store IndexH]
    return storeIndex + 1
```

Trace the Algorithm

Initial Array : [8, 7, 2, 1, 0, 9, 6]
(0 … 6)

1) First ~~Partition~~ ~~Algo~~ Quick sort Algo

Quicksort (array, 0, 6)

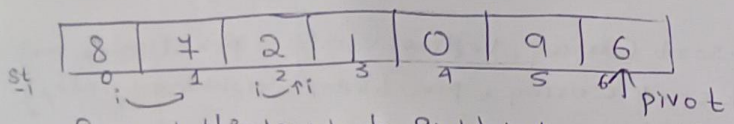
→ checks for low < high (0 < 6) → true

Partition (array, 0, 6) → goes to the Partition Block

Partition (array, 0, 6)

Rightmost element as → pivot → 6

Store index is → left index − 1 → 0 − 1 = −1



for i = leftindex + 1 to Right index

for → i = 0 : element [i] < pivot element

8 < 6 , no swap

for → i = 1 : element [i] < pivot element

7 < 6 , no swap

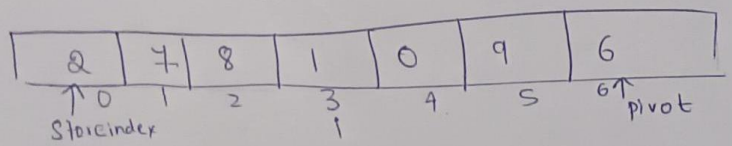
for → i = 2 : element [i] < pivot element

2 < 6 Yes

Swap [i] & storeindex

Swap [2] & Swap [8]

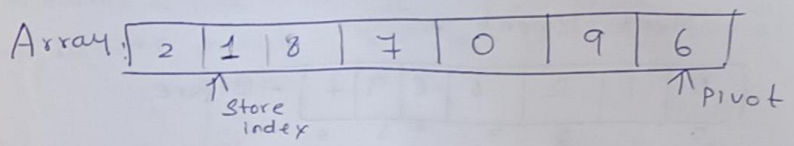
Store index = 0



element [i] < pivot element

for i = ~~3~~ : 1 < 6 Yes

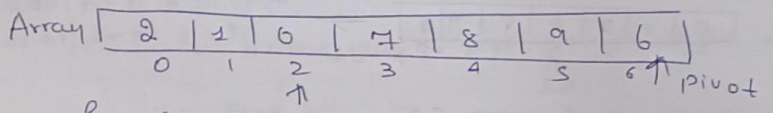
Swap array [1] and array [3]



i = 4 0 < 6

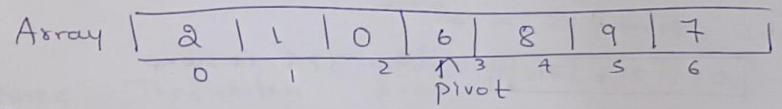
Swap array [2] & array [4]

Storeindex + + → 2

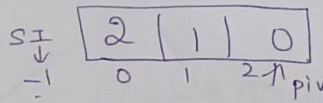


for i = 5 9 > 6 no swap

Swap the pivot (6) with arr [3]



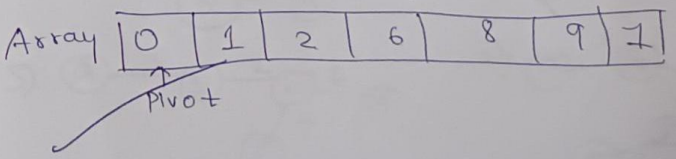
Recursive Call on left sub array (quick sort (array, 0, 2)



for i = 0 2 > 0 no swap

i = 1 1 > 0 no swap

Swap pivot array [2] with array [0]



Step 3 Rewrcive call on Right Subarray

Quick sort (array, 1, 2))

1 pivot element → 2

0	1	2	6	8	9	7
0	1	2	3	4	5	6

→ Before

sto ↑0 pivot ↑2

for = 1 i<2 swap array[1] with array[1] no change

SI → 1

Step 4

0	1	2	6	8	9	7
0	1	2	3	4	5	6

SI ↑3 pivot ↑6 →

for i = 4 8>7 no swap

i = 5 9>7 no swap

swap pivot array[4] with array[6]

Array

0	1	2	6	7	9	8
0	1	2	3	4	5	6

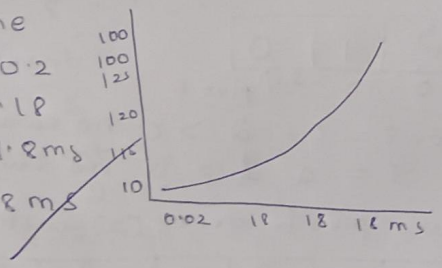
pivot ↑4

Step 5 → piv → 8, 5] → 4

0	1	2	6	7	8	9

i=5 9>8 no swap

n input	time
10	0.02
100	0.18
1000	1.8ms
10000	18 ms



Prism Algorithm

→ Minimum Spanning tree

→ forms a tree that has every Vertex

→ has the minimum sum of weight among all trees

Pseudocode / Algorithm

T = ϕ

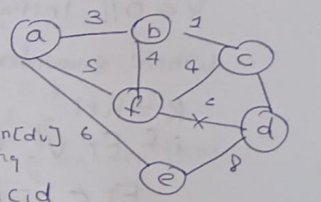
U = {1}

while (U ≠ V)

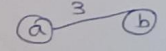
let (u, v) be the lowest cost edge u∈U + v∈V-U

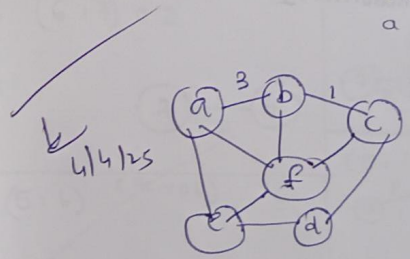
T = T ∪ {(u, v)}

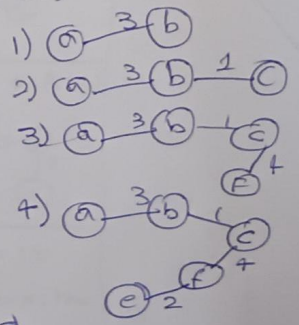
U = U ∪ {v}



Source	Tree vertex	d[v] = min[dv] & Remaining
a	a(-10)	b|f|e|c|d
		a(f,5), a(b,3)
		a(d,∞) a(a,6)
		a(c,6)



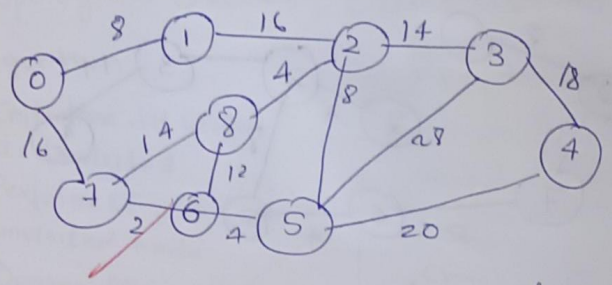




Kruskal's Algorithm

Algorithm
// I/P Weighted Connected Graph
G ∈ (V, E)
// O/P ET edges of MST
Sort E in non decreasing order of the edge weight
w(e_i1) ≤ ≤ .. ≤ w(e_i|E|)
ET ← ϕ ; ecounter ← 0
// initialize set of tree edges & its size
k ← 0 // Initialize no. of Procesed edges
while ecounter < |V| − 1 do
k = k + 1
if ET ∪ {e_ik} is Cycyclic
ET ← ET ∪ {e_ik}
ecounter ← ecounter + 1
return ET

Example / tracing of problem

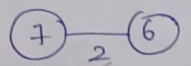


Step 1 Identify all edges in order of weight (small to big)

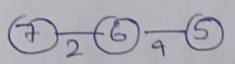
(6,7) = 2 ✓
(5,6) = 4 ✓
(8,2) = 4 ✓
(2,5) = 8 ✓
(0,1) = 8 ✓
(6,8) = 12 ✗
(7,8) = 14 ✗
(2,3) = 14 ✓
(1,2) = 16 ✗
(0,7) = 16
(3,4) = 18 ✓
(5,4) = 20 ✗
(5,3) = 28 ✗

Step 2 Construct the Graph from least one

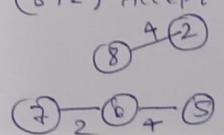
(6,7) = 2 Accept



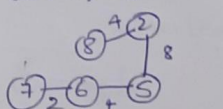
(5,6) (Accept)



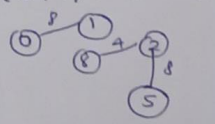
(8,2) Accept



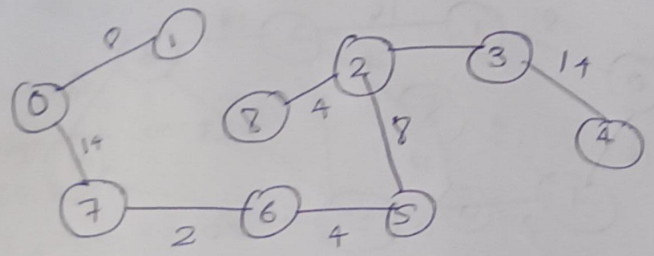
(2,5) = 8 Accept



(0,1) = 8 Accept



(6,8) = 12
×
As it forms the cycle we cannot take it



Minimal spanning tree

$8 + 14 + 2 + 4 + 8 + 4 + 14 =$

Topological Ordering (DFS - based)

Algorithm

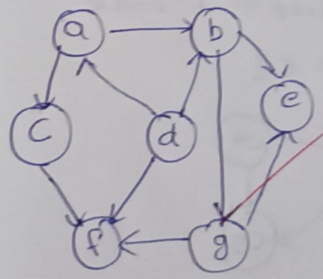
1) Initialize all vertices as unvisited
2) Perform DFS on each unvisited node
3) During DFS, after visiting all neighbours of a node, push it to a stack
4) After the DFS is complete the stack will contain the vertices in topological order

→ for each neighbour u of v
  if not visited[u]:
    DFS(u, visited, stack)
  Stack.push(v)

Pseudocode

```
TopologicalSort(graph G)
  Create a visited[] array and initialize all to false
  Create an empty stack S
  for each vertex v in G:
    if not visited[v]:
      DFS(v, visited, S)
  while stack S is not empty
    print S.pop()

DFS(v, visited, stack):
  visited[v] = true
  for each neighbour u of v:
    if not visited[u]:
      DFS(u, visited, stack)
  visited[v] = true
```

Tracing with an Example

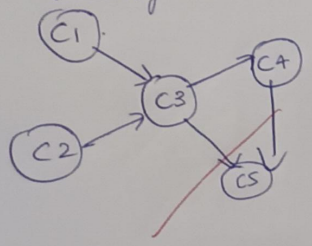


Stack Adjcent Vertex Node Visited Stackpop

Topological sorting based on Source Removal Method

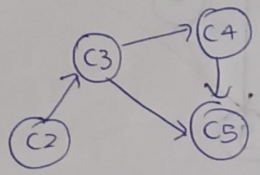
```
function Topological order (G)
  for i = 1 to n
    Indegree [i] = 0
    for j = 1 to n
    Indegree [i] = Indegree [i] + A [j] [i]
for i = 1 to n
choose j with Indegree [j] = 0
  enumerate j
    Indegree [j] = -1
    for k = 1 to n
      if A [j] [k] == 1
    Indegree [k] = Indegree [k] - 1
```

Tracing



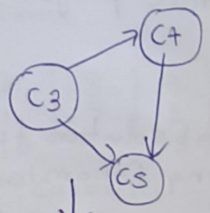
→ whose incomming is 0 Delete that node

Delete C1

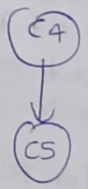


Again check whose incomming node vertex is 0

Delete C2



Delete C3



Delete C4

The topological order is C1, C2, C3, C4, C5

Implement 0/1 knapsack problem Using Dynamic programming

Algorithm

Aim To find optimal solution for the knapsack problem Using Dynamic Programming

Input n → no of objects to be selected
m → weight/capacity of knapsack
w → weights of all objects
p → profits of all objects

Output → Optimal solution for the ni of objects

```
for i = 0 to n do
  for j = 0 to m do
    if (i = 0 or j = 0)
      V[i,j] = 0
    else if (w[i] > j)
      V[i,j] = V[i-1,j]
    else
      V[i,j] = max (V[i-1,j],
               V[i-1, j-w[i]] + p[i])
    endif
  endfor
endfor
```

Max(V[i-1, j], V[i-1, j - wi] + pi

Tracing

W = 5 capacity n = 4 items

items	1	2	3	4
weight	2	1	3	1
profit	12	10	20	15

V[i,j] =
- 0 → if i=j=0
- V[i-1,j] → if wi > j
- max(V[i-1,j], V[i-1, j-wi] + pi) → if wi ≤ j

j → M capacity

items	0	1	2	3	4	5
i 0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
n 4	0	10	15	30	37	

Profit table

item 1 → weight 2
we can store it in 1 capacity → NO

item 1 → In capacity 2, 3, 4, 5 can be store as its weights → 2
all other, are profit 12 capacity

item 2 → weight → (1) Profit → 10

capacity 1 can store

capacity 2 which is already there 12 either we can store item 1 → 2 weigh → 12 V(choose the maximum) item 2 → 1 weigh → 10

w[i] > j
w[1] > 2
V[i,j] = V[i-1,j] = V[1,12]

↳ satisfies 3rd conditon

w[i] < j capacity
item [1] < 2

max [V[2-1, 12], V[2-1], 2[1] + 10]
[1,12] 1 1+10 =

Item 2 weight - 1 profit = 10

Already there 12

item 1 → weight 2 (12) profit

item 2 → weight 1 (10)

or choose max

both weight we can do 2+1 = 3

capacity 3

= 12+10 = 22 ✓

capacity 3

$w[i] < j$ $V[i,j] = \max(V[i-1,j], V[i-1, j-w[i] + p[i]])$

$w[2] < 3$ $V[2,3] = \max(V[2-1,3], V[2-1, 3-1+10])$

$1 < 3$ $V(1,3)$ 12, $V[1,2]$ 0, $2+10$

v

item 2 capacity = 4

item 1 → (w 2) → 12 choose max

item 2 → (1) → 10

Item 1,2 → (2+1) → 22 ✓ 10+22

item 2 capacity = 5

item 1 → w 2, p 12

item 2 → (1) 10

Item 1,2 → 3 → 22 ✓

Item 3 weight → 3 profit 20

~~item~~ capacity 1

weight = 3 cannot be filled

take the privious

$w[i] > j$

$w[3] > j$

$3 > 1$

$V[i,j] = V[i-1,j]$

$V[3,1] = V[3-1,j]$

$= V[2,1]$

$= 10$

capacity 2

weight = 3 cannot be filled

take the privous

$w[i] > j$

$w[3] > j$

$3 > 2$

$V[i,j] = V[i-1,j]$

$V[3,2] = V[3-1,2]$

$= V[2,2]$ table

$= 12$

capacity 3

w[i] w[3] = 3

i3 → wei → 3 → 20

i1, i2 → weight 3 → 22 ✓

choose max

$w[i] < j$

$w[3] < 3$

$V[3,3] = \max[$

$V[i-1,j], V[i-1], j - w[i] + p[i])$

$V[3-1,3]$ $V[2,3]$ (22), $V[3-1]$ $V[2,0]$ 0

$3 - [3 + 20]$

$3 - 23 = 20$

capacity 4

where we have options

either we can store

item 2 & item 3 → 3+1 = 4 j

20+10 = 30

previous (20) - (30)

choose max 30

capacity 5

previous 32

i3 + i1 =

3 + 2 = 5 weight

20 + 12 = 32

22 or 32 ✓

Item 4 - weight → 2 Profit 15

capacity 1

NO

weight = 2

$w[i] > j$

$w[4] > 1$

$4 > 1$

$V[i,j] = V[i-1,j]$

$V[3,1] = V[3-1,1]$

$= 10$ //

capacity 2

weight = 2 yes

compare w profit

item 1 → 2 → 12

item 4 → w 2 prof 15

choose the max

15

capacity 3

previous 22

item 4 + item 2

w[2] w=1 =3

15 + 10 = 25

choose max

25

capacity 4

previous 30

14 + i1

1) 2 + 2 = 4

15 + 12 = 27

2) i3 + i2

we 3 + 1 = 4

pro 10 + 20 = 30 ✓

choose max

capacity 5

previous 32

1) i4 + i1 + i2

2 + 2 + 1 = 5

15 + 12 + 10 = 37 ✓ choose max

37 //

2) i3 + i4

3 + 2 = 5

20 + 15 = 35

3) i3 + i1

3 + 2 = 5

20 + 12 = 22

$V[4,5] = \max[V[i-1,j]\ V[i-1]$

$j - w[i] + p[i])$

→ max[v[32],

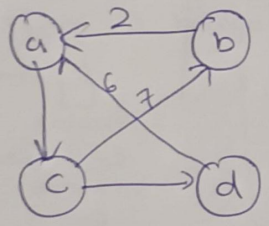
Implement All pair shortest path problem Using Floyd Algo

Algorithm

I/P : weighted matrix w of Graph
O/P : distance matrix of shortest path

```
D ← W
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      D[i,j] = min { D[i], D[i,k] + D[k,j] }
Return D
```

Tracing



step 1 find the cost matrix put diagonal = 0

$$R0 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & \infty & 0 \end{bmatrix} \end{array}$$

step 2: Consider shortest path

$$R1 \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ \infty & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

Next for b

through A i = 1, j = 1, K = 1

$D[i,j] \leftarrow \min\{[i+k] + D[k+j]\}$

b, c = min { ∞, ba + ac } = 2 + 3 = 5
b, d = min { ∞, 2 + ∞ } = ∞
c, b = min { 7, ∞ + 3 } = 7
c, d = min { 1, ∞ + ∞ } = 1
d, b = min { ∞, 6 + ∞ } = ∞
d, c = min { ∞, 6 + 3 } = 9

$$R2 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & \infty & 3 & \infty \\ 2 & 0 & 5 & \infty \\ 9 & 7 & 0 & 1 \\ 6 & \infty & 9 & 0 \end{bmatrix} \end{array}$$

for c i for next

through B i = 2, j = 2, k = 2

a, c = min { 3, ∞ } = 3
a, d = min { ∞, ∞ } = ∞
c, a = min { ∞, 7 + 2 } = 9
c, d = min { 1, 7 + ∞ } = 1
d, a = min { 6, ∞ + 2 } = 6
d, c = min { 9, ∞ + 5 } = 9

$$R3 = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 10 & 3 & 4 \\ 2 & 0 & 5 & 6 \\ 9 & 7 & 0 & 7 \\ 6 & 16 & 9 & 0 \end{bmatrix} \end{array}$$

Next for d

through C i = 3, j = 3, K = 3

a, b = min { ∞, 3 + 7 } = 10
a, d = min { ∞, 3 + 1 } = 4
b, a = min { 2, 5 + 9 } = 2
b, d = min { ∞, 5 + 1 } = 6
d, a = min { 6, 9 + 3 } = 6
d, b = min { ∞, 9 + 7 } = 6

$R^4 =$

	a	b	c	d
a	0	10	3	4
b	2	10	5	6
c	7	7	0	1
d	6	6	9	0

→ final shortest distance

through d i = 4 j = 4 , k = 4

a , b = min { 10, 4+6 } = 10

a , c = min { 3, 4+9 } = 3

b , a = min { 2, 6+6 } = 2

b , c = min { 5, 6+9 } = 5

c , a = min { 9, 1+6 } = 7

c - b = min { 7, 1+6 } = 7

1) From a Given Vertex in a weighted Connected Graph, find shortest paths to other vertices using Dijkstra algorithm

2) Implement Fractional knapsack Using Greedy technique

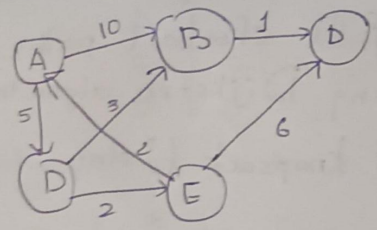
(2) Fractional Knapsack

1) Dijkstra Algorithm

Algorithm
n = no of vertices, w = cost adjacency matrix with

```
for i = 0 to n-1 do
    d[i] = Const [source[i]]
    S[i] = 0
endfor
S[source] = 1 // visited
for i = 1 to n-1 do
for u ∈ d[u]
 d[u] is min
for [j = 0 to n-1] do
 add u ∈ V-S
    add u to S ie s[u] = 1
  for every V ∈ V-S do [i.e for V = 0 to n-1]
   if d[j] + w[u,v] < d[v]
      d[v] = d[u] + w[u,v]
  endif
  endfor
    endfor
```

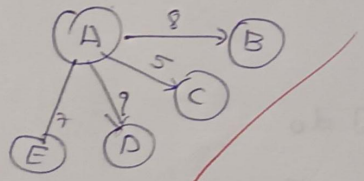
Tracing



Source = A

Source	V = V - S	d[v] min(d[v], d[u] + w[u][v])
A	B C D E	d[B] min (∞, 0+10) = 10
		d[C] min (∞, 0+5) = 5
		d[D] min (∞, 0+∞) = ∞
		d[E] min (∞, 0+∞) = ∞ (min)
A, C	B D E	d[B] min (10, 5+3) = 8
		d[D] min (∞, 5+∞) = ∞
		d[E] min (∞, 5+2) = 7 (min)
A C E	B D	d[B] min (8, 7+∞) = 8 - min
		d[D] min (∞, 7+6) = 13
A C E B	D	d[D] min (13, 8+1) = 9 - min

A C E B D



2) Fractional Knapsack

Algorithm

1. Input n items with profits and weights, and knapsack capacity W
2. Calculate profit/weight ratio for each item
3. Sort items in descending order of each
4. Take as much of each item possible
   * if it fits, take whole item
   * if not, take fraction that fits
5) Stop when knapsack is full

Tracing

Step 1 → divide profit/weight

Example

Obj	1	2	3	4	5	6	7
P	10	5	15	7	6	18	3
W	2	3	5	7	1	4	1
P/W	5	1.6	3	1	6	4.5	3

Obj	profit	weight
5	6	1
1	10	2
6	18	4
3	15	2
7	3	1
2	$5 \times \frac{2}{3}$	$3 \times \frac{2}{3}$ ⇒ = 15 kgs

$= 1.3$

$= 53.3$

Implement Johnson Trotter algorithm to Generate permutation

Leetcode problems

1) Count of Range Sum (Merge sort

```
# include <stdlib.h>
int mergesort (long *sum, int left, int right, int
               lower, int upper) {
    if (right - left <= 1) return 0;
    int mid = (left + right) / 2;
    int count = mergesort (sum, left, mid, lower, upper) +
                mersort (sum, mid, right, lower, upper);
    int j = mid, k = mid, t = mid;
    int n = right - left;
    long *cache = (long *) malloc (n * sizeof (long));
    int r = 0
    for (int i = left; i < mid; i++) {
        while (k < right && sum[k] - sum[i] < lower) k++;
        while (j < right && sum[j] <= upper) j++;
        count += j - k;
    while (t < right cache [r++] = sum[t++]
    for (int i = 0; i < r; i++) sum [left + i] = cache[i]
    free (cache)
```

Case 1 case 2

[-2, 5, -1]

lower -2 upper 2

2) Kth largest element in an Array

```
# include <stdlib.h>
void heapify (int *heap, int heapsize, int i) {
    int smallest = i;
    int left = 2 * i + 1
    int right = 2 * i + 2;
if (left < heapsize && heap [left] < heap [smallest])
        smallest = left;
if (right < heapsize && heap [right] < heap [smallest])
        smallest = right;
if (smallest != i) {
    int temp = heap [i];
    heap [smallest] = temp;
    heapify (heap, heapsize, smallest);
}
}
void buildminheap (int * heap, int heapsize) {
    for (int i = heapsize / 2 - 1; i >= 0; i --) {
        heap [i] = nums [i];
}
buildminheap (heap, k);
for (int i = k; i < numsize; i++) {
    if (nums[i] > heap[0]) {
        heap[0] = nums[i];
        heapify (heap, k, 0)
}
int result = heap[0];
free (heap);
return result;
}
```

3) Course schedule

```
#include<stdbool.h>
bool dfs(int course, int graph[][2000], int* size, int* state)
{ if (state[course]==1) return false; // cycle found
  if (state[course]==2) return true;

state[course] = 1;
for(int i=0; i<size[course]; i++){
  if (!dfs(graph[course][i], graph, size, state))
      return false;
}
state[course]=2
return true
}

bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int* prequisitescolsize)
{ int graph[2000][2000] = {0};
  int size[2000]={0};
  int state[2000]={0};
for(int i=0; i<prequisitiessize; i++){
    int a = prequisites[i][0];
    int b = prequisites[i][1];
    graph[b][size[b]++]=a;
}
for(int i=0;i<numCourses; i++){
  if(state[i]==0 && !dfs(i,graph,size,state))
      return false;
}
  return true;
}
```

4) Pizza with 3n slices

```
#include<stdlib.h>
#include<string.h>

int max(int a, int b){return a>b?a:b;}
int helper(int* slices, int start, int end, int n){
  int len = end-start+1;
  int dp[501][251];
memset(dp,0,sizeof(dp));
for(int i=0; i<len; i++){
  for(int j=1; j<=k; j++){
    if(i==0)
      dp[i][j] = (j==1)?slices[start+i]:0;
    else if(i==1)
      dp[i][j] = max(dp[i-1][j], (j==1?slices
        [start+1]:0));
    else
      dp[i][j] = max(dp[i-1][j], dp[i-2][j-1]+
        slices[start+i]);
  }}
return dp[len-1][k];
}
```

5) Number of Ways to Arrive at Destination

```
#include<limits.h>
#include<stdbool.h>
#define MOD 1000007
#define MaxRd 201

int countPaths(int n, int** roads, int roadsize,
    int *roadcolsize){
  int graph[MaxN][MaxN];
  for(int i=0; i<N; i++)
    for(int j=0; j<n; j++)
      graph[i][j] = -1;
  // Build Graph
  for(int i=0; i<roadsize; i++){
    int u=road[i][0], v=roads[i][1], t=roads[i][2];
    graph[u][v] = t;
    graph[v][u] = t;
}
```

```
long long dist [MaxN];
int ways[Maxn] = {0};
bool visited [MaxN] = {false};

for(int i=0; i<n; i++) dist[i] = (long-max);
  dist[0] = 0
  ways[0] = 1;

for (int i=0; i<n; i++){
  int u = -1
  for(int j=0; j<n; j++)
    if(!visited[j] && (u==-1 || dist[j]<dist[u]))
      u=j;
  visited[u] = true
  for(int v=0; v<n; v++){
    if(graph[u][v] == -1 || visited[v])
      continue;
    long long newDist = dist[u] + graph[u][v]
      if(newDist < dist[v]){
        dist[v] = newDist;
        ways[v] = ways[u];
      }else if(newDist == dist[v]){
        ways[v] = (ways[v] + ways[u])
}}}
return ways[n-1]
```

6) Maximum Units on a truck

```
#include<stdlib.h>
int cmp(const void *a, const void *b){
    int *boxA = *(int**)a;
    int *boxB = *(int**)b;
    return boxB[1] = box[1];

int maximumUnits(int **boxtypes, int boxTypesize, int *box
                                                  colsize
    qsort(boxtypes, boxtypesize, sizeof(int*), cmp);
  int totalUnits = 0;
for(int i=0; i< boxtypesize && truck size >0; i++){
    int boxestotake = boxtype[i][0];
  int boxunitsPerBox = boxtype[i][1];
    int take = boxestotake < trucksize
    totalUnits += take * unitsPerBox;
      trucksize -= size
  } return totalUnits
}
```

23/05/2025

Implement N Queens Problem Using Backtracking

Algorithm

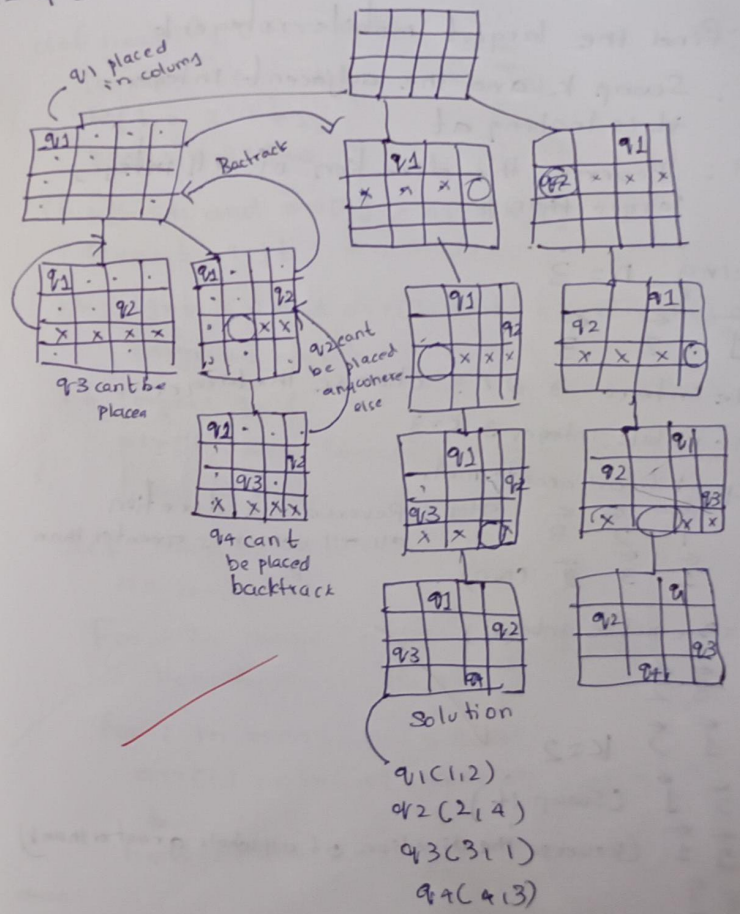
```
// Initialize the Board
int board[N][N] = {0};
bool SolveNQUtil (int board[N][N] , int col)
    if (col >= N) return true;
    for (int i=0; i<N; i++)
    // check
        if (issafe (board , i , col))
    // If Yes then  board[i][col] = 1;
    // Recursively place queen in the column
        if (SolveNQUtil (board , col+1)) return true;
        board[i][col] = 0
bool isSafe (int board[N][N] , int row, int col)
    for (int i=0; i<col; i++) if (board[row][i])
            return false;
    for (int i=row, j=col; i>=0 && j>0; i--, j--)
    if (board[i][j]) return false
bool solveNQ()
    SolveNQUtil(board, 0)
```

Algorithm

1) start in the leftmost column
2) If all Queens are placed, return true
3) Try placing the Queen in all rows one by one for the current column

NO Queen should be placed horizontally / vertically / diagonally

(n = 4 = 4 Queens



Johnson trotter Algorithm

// Algorithm

// Implements johnson - trotter

// IP : A +ve int n

// o/p A list of operation of the set 1 to n

Step 1 : Initialize the first permutation with < 1 < 2 < n

Step 2 : Find the largest mobile integer k

Step 3 : Swap k and the adjacent integer it is looking at

Step 4 : Reverse the direction of all integers larger than k

Tracing n = 3

← ← ←

1 2 3

mobile integer → 0 1 3 choose the larger

1) largest mobile integer 3 k = 3

2) Swap k & adjacent mobile integer

← ← ← Step 4 Reverse the direction all all which is greater than k

1 2 3

← ←

1 3 2 (NP).

K = 3 (mobile integer)

← ← ←

1 3 2

← ← ←

3 1 2 k = 2

← ← ←

3 2 1 (Swap it)

→ ← ←

3 2 1 (Reverse the direction of all whols greater than k)

← → ←

2 3 1

← ← →

2 1 3

Sort a Given set of N integer Elements using HeapSort technique and Compute its time taken

→ Bottom up Approach is better always bcz its faster & optimal O(n) where the Top-down Time complexity O(nlogn)

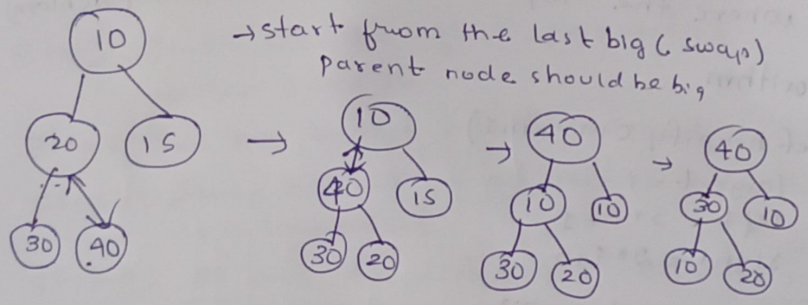
Algorithm

```
def heapify (arr, n, i)
    largest = i
    left = 2*i + 1
    right = 2*i + 2

if left < n and arr[i] < arr[left]
    largest = left

if right < n and arr[largest] < arr[right]:
    largest = right

if largest != i:
    arr[i], arr[largest] = arr[largest], arr[i]
    heapify (arr, n, largest)

def heap_sort (arr):
    n = len (arr)

For i in range (n//2 - 1, -1, -1):
    heapify (arr, n, i)

for i in range (n-1, 0, -1):
    arr[i], arr[0] = arr[0], arr[i]
    heapify (arr, i, 0)
```

Tracing

Array : 10, 20, 15, 30, 40

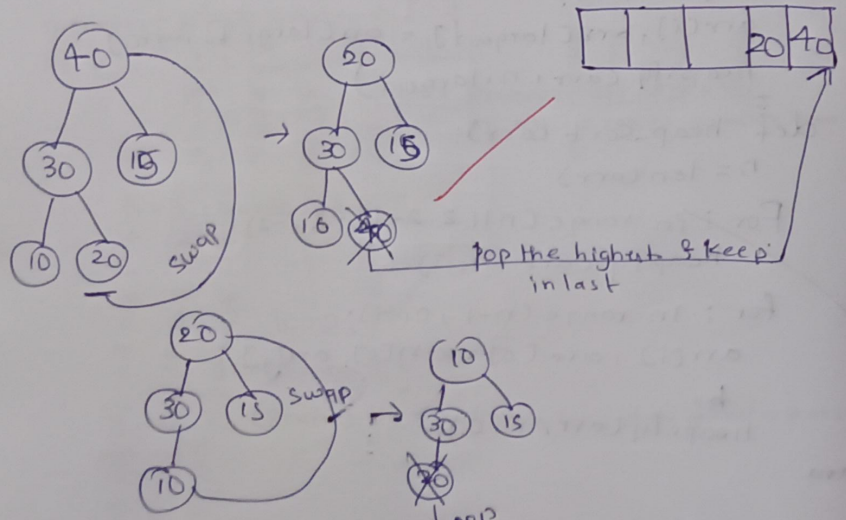
Step 1 : Get the Max heap Using Bottom Up approch

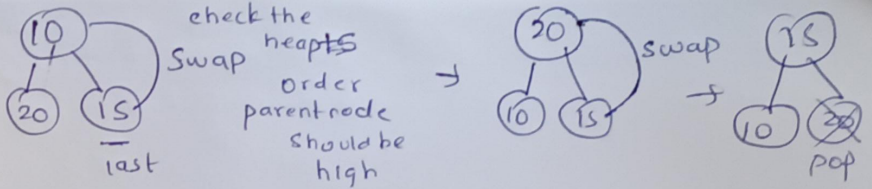


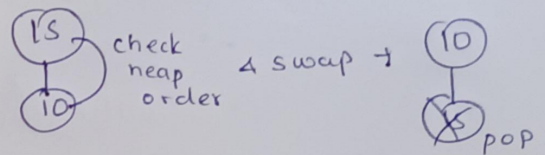
Step 2 . Apply the deletion Process

1) Consider the last & first element Swap & pop the highest element

2) check the order of heap







Sorted list of Element

1	2	3	4	5
10	15	20	30	40

23/5/25