VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014,Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Dharunya Balavelavan (1BM23CS090)

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 Dharunya Balavelavan **(1BM23CS090)**, who is a 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.

Shrushti C S
Assistant Professor
Department of CSE
Bengaluru BMSCE,

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

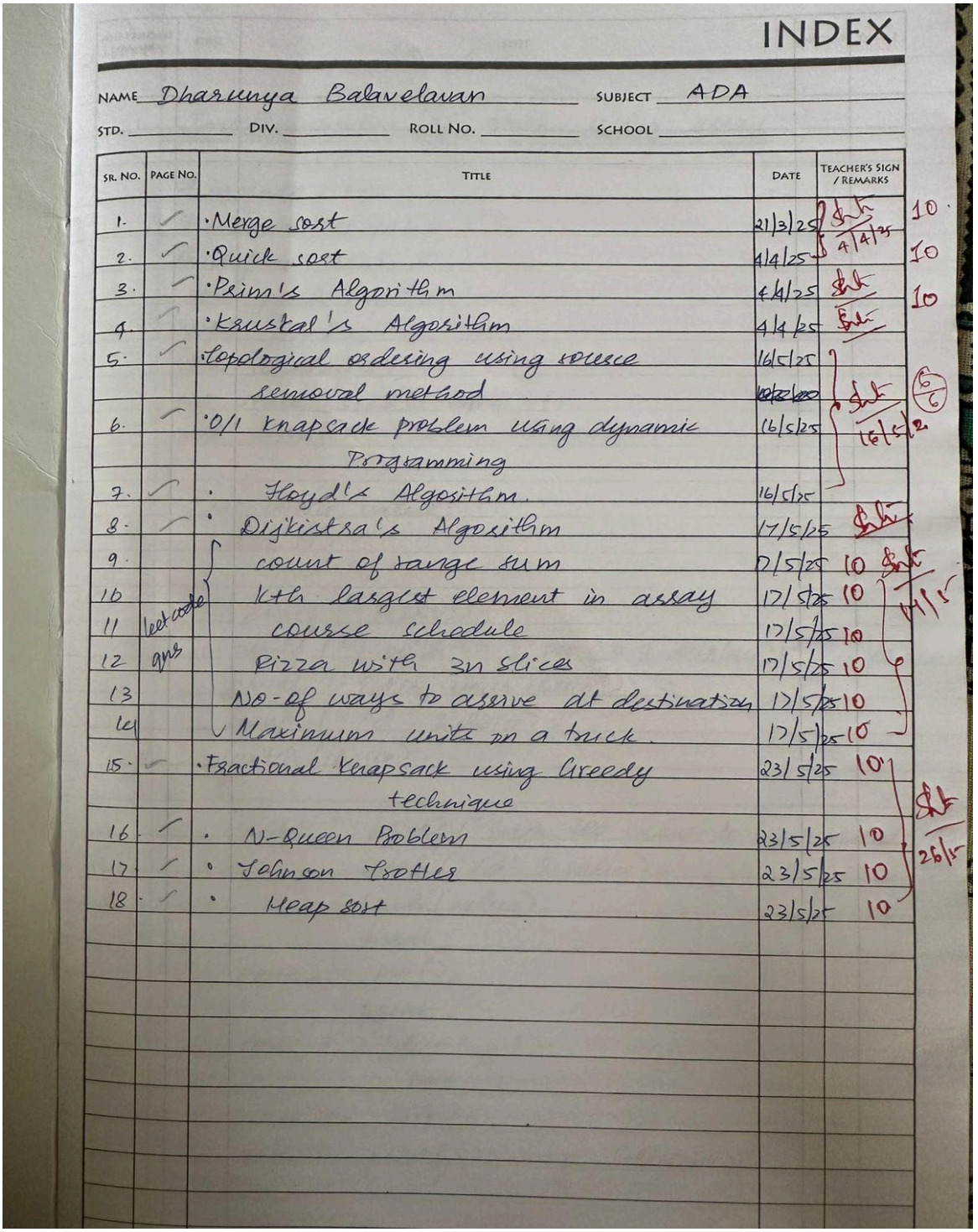
Index Sheet

Sl. No.	Experiment Title	Page No.
1	Write a program to obtain the Topological ordering of vertices in a given digraph. LeetCode Program related to Topological sorting	5
2	Implement Johnson Trotter algorithm to generate permutations.	9
3	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. LeetCode Program related to sorting.	11
4	Sort a given set of N integer elements using Quick Sort technique and compute its time taken. LeetCode Program related to sorting.	15
5	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	19
6	Implement 0/1 Knapsack problem using dynamic programming. LeetCode Program related to Knapsack problem or Dynamic Programming.	22
7	Implement All Pair Shortest paths problem using Floyd's algorithm. LeetCode Program related to shortest distance calculation.	25
8	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	29

9	Implement Fractional Knapsack using Greedy technique. LeetCode Program related to Greedy Technique algorithms.	33
10	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	37
11	Implement "N-Queens Problem" using Backtracking.	40

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.



INDEX

NAME Dharunya Balavelavan SUBJECT ADA

STD. _____ DIV. _____ ROLL NO. _____ SCHOOL _____

SR. NO.	PAGE NO.	TITLE	DATE	TEACHER'S SIGN / REMARKS
1.	✓	• Merge sort	21/3/25	Sk 4/4/25 10
2.	✓	• Quick sort	4/4/25	10
3.	✓	• Prim's Algorithm	4/4/25	Sk 10
4.	✓	• Kruskal's Algorithm	4/4/25	Sk
5.	✓	• topological ordering using source removal method	16/5/25	Sk 16/5 (6/6)
6.	✓	• 0/1 knapsack problem using dynamic Programming	16/5/25	
7.	✓	• Floyd's Algorithm.	16/5/25	
8.	✓	• Dijkistra's Algorithm	17/5/25	Sk
9.		count of range sum	17/5/25 10	Sk 17/5
10		Kth largest element in assay	17/5/25 10	
11	leetcode	course schedule	17/5/25 10	
12	qns	Pizza with 3n slices	17/5/25 10	
13		No-of ways to arrive at destination	17/5/25 10	
14		Maximum units on a truck.	17/5/25 10	
15.	✓	• Fractional knapsack using Greedy technique	23/5/25 10	Sk 26/5
16.	✓	• N-Queen Problem	23/5/25 10	
17.	✓	• Johnson Trotter	23/5/25 10	
18.	✓	• Heap sort	23/5/25 10	

GITHUB LINK:

https://github.com/Dharunya21/ADA-

Lab program 1:

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

Code:

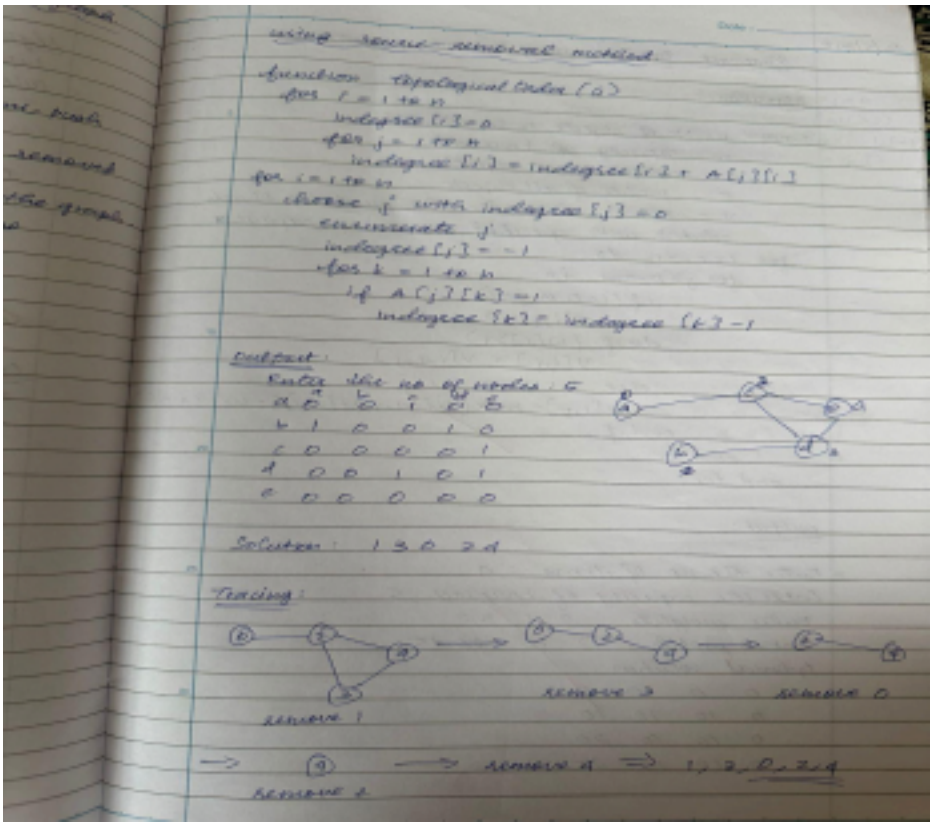
```
#include <stdio.h>
#define MAX 100
int adj[MAX][MAX], n, visited[MAX], stack[MAX], top = -1;
void dfs(int v) {
    visited[v] = 1;
    for (int i = 0; i < n; i++)
        if (adj[v][i] && !visited[i])
            dfs(i);
    stack[++top] = v;
}
void topologicalSort() {
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i);
    while (top >= 0)
        printf("%d ", stack[top--]);
}
int main() {
    int edges, u, v;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &n, &edges);
    for (int i = 0; i < edges; i++) {
        scanf("%d %d", &u, &v);
```

```
        adj[u][v] = 1;
    }
    printf("Topological Order: ");
    topologicalSort();
    return 0;
}
```

Result:

```
Enter number of vertices and edges: 6 7
0 1
0 2
2 1
2 3
3 4
3 5
4 5
Topological Order: 0 2 3 4 5 1
```

Algorithm and Tracing Screenshot:



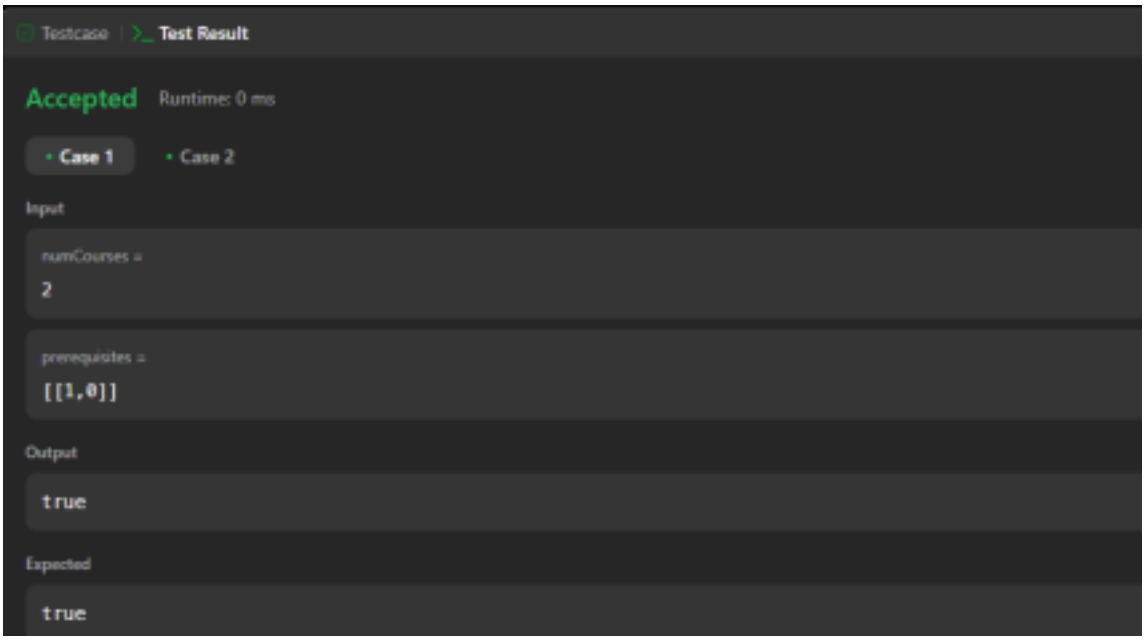
LEETCODE: Course Schedule

Code:

```
typedef struct Node {
    int course;
    struct Node *next;
} Node;
void addEdge(Node **graph, int src, int dest) {
    Node *newNode = (Node *)malloc(sizeof(Node));
    newNode->course = dest;
    newNode->next = graph[src];
    graph[src] = newNode;
}
bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize,
int* prerequisitesColSize) {
    Node **graph = (Node **)calloc(numCourses, sizeof(Node *));
    int *indegree = (int *)calloc(numCourses, sizeof(int));
    for (int i = 0; i < prerequisitesSize; i++) {
        int course = prerequisites[i][0];
        int pre = prerequisites[i][1];
        addEdge(graph, pre, course);
        indegree[course]++;
    }
    int *queue = (int *)malloc(numCourses * sizeof(int));
    int front = 0, rear = 0;
    for (int i = 0; i < numCourses; i++) {
        if (indegree[i] == 0) {
            queue[rear++] = i;
        }
    }
    int count = 0;
    while (front < rear) {
int current = queue[front++];
count++;
```

```
        Node *temp = graph[current];
        while (temp != NULL) {
            indegree[temp->course]--;
            if (indegree[temp->course] == 0) {
                queue[rear++] = temp->course;
            }
            temp = temp->next;
        }
    }

    for (int i = 0; i < numCourses; i++) {
        Node *temp = graph[i];
        while (temp != NULL) {
            Node *next = temp->next;
            free(temp);
            temp = next;
        }
    }
    free(graph);
    free(indegree);
    free(queue);

    return count == numCourses;
}
```

Result:



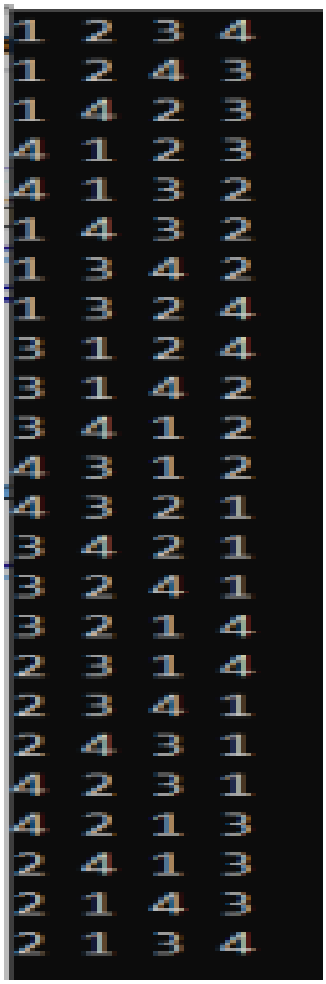
Lab program 2 :

Implement Johnson Trotter algorithm to generate permutations. Code:

```
#include <stdio.h>
#include <stdlib.h>
void swap(int* a, int* b) {
int temp = *a;
*a = *b;
*b = temp;
}
void generatePermutations(int arr[], int start, int end) {
if (start == end) {
for (int i = 0; i <= end; i++) {
printf("%d ", arr[i]);
}
printf("\n");
} else {
for (int i = start; i <= end; i++) {
swap(&arr[start], &arr[i]);
generatePermutations(arr, start + 1, end);
```

```
swap(&arr[start], &arr[i]); // backtrack

}

}

}

int main() {

int n;

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

int* arr = (int*)malloc(n * sizeof(int));

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

scanf("%d", &arr[i]); }

generatePermutations(arr, 0, n - 1);

free(arr);

return 0;

}
```

Result:



Lab program 3 :

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> // for exit

#include <time.h> // for clock timing

void split(int[], int, int);

void combine(int[], int, int, int);

int main() {

 int a[15000], n, i, j, ch, temp;

 clock_t start, end;

 while (1) {

 printf("\n1: For manual entry of N value and array elements");

 printf("\n2: To display time taken for sorting number of elements N in the range 500 to 14500");
```

```
printf("\n3: To exit");

printf("\nEnter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("\nEnter the number of elements: ");

scanf("%d", &n);

printf("\nEnter array elements:\n");

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

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

}

start = clock();
split(a, 0, n - 1);

end = clock();

 printf("\nSorted array is:\n");

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

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

printf("\nTime taken to sort %d numbers is %f Secs\n", n, ((double)(end - start)) /
CLOCKS_PER_SEC);

break;

case 2:

n = 500;

while (n <= 14500) {

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

a[i] = n - i; // Reverse order to simulate worst case

start = clock();

split(a, 0, n - 1);

// Dummy loop to create consistent delay

for (j = 0; j < 500000; j++) {
```

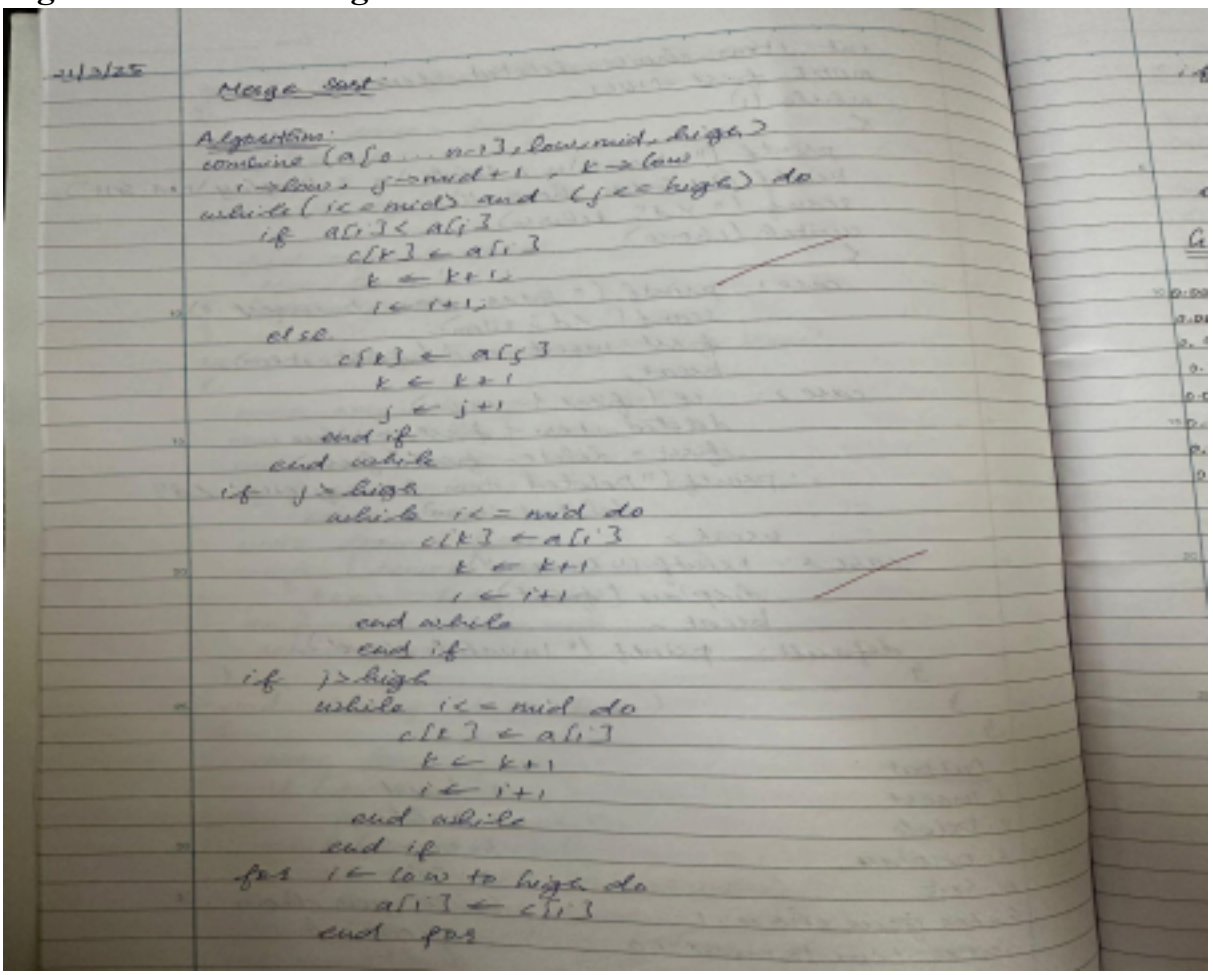
```
temp = 38 / 600;

}

end = clock();

printf("\nTime taken to sort %d numbers is %f Secs", n, ((double)(end - start)) /
CLOCKS_PER_SEC);

n = n + 1000;

}

break;

case 3:

exit(0);
}

} return 0;

}


// Recursive function to divide the
array void split(int a[], int low, int

high) {

int mid;

if (low < high) {

mid = (low + high) / 2;

split(a, low, mid);

split(a, mid + 1, high);

combine(a, low, mid, high);

}

}

// Function to merge two sorted halves void

combine(int a[], int low, int mid, int high) { int

c[15000], i, j, k;

i = k = low;

j = mid + 1;
```

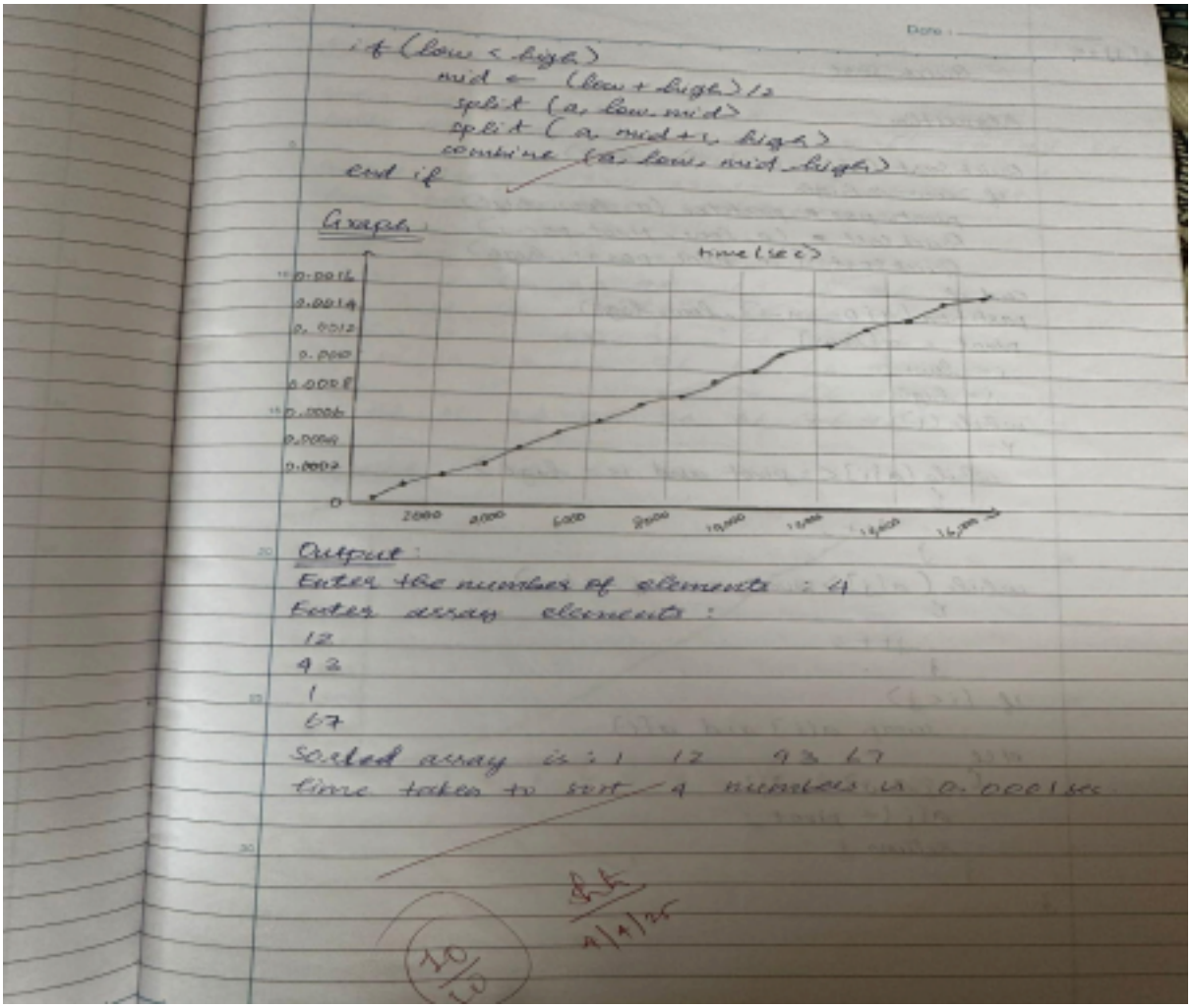
```
while (i <= mid && j <= high) {
if (a[i] < a[j]) {
c[k++] = a[i++];
} else {
c[k++] = a[j++];
}
}
while (i <= mid) {
c[k++] = a[i++];
}
while (j <= high) {
c[k++] = a[j++];
}
for (i = low; i <= high; i++) {
a[i] = c[i];
}
```

Result:

```
1: For manual entry of N value and array elements
2: To display time taken for sorting number of elements N in the range 500
    to 14500
3: To exit
Enter your choice: 2

Time taken to sort 500 numbers is 0.000524 Secs
Time taken to sort 1500 numbers is 0.000593 Secs
Time taken to sort 2500 numbers is 0.000587 Secs
Time taken to sort 3500 numbers is 0.000526 Secs
Time taken to sort 4500 numbers is 0.000597 Secs
Time taken to sort 5500 numbers is 0.000642 Secs
Time taken to sort 6500 numbers is 0.000698 Secs
Time taken to sort 7500 numbers is 0.000743 Secs
Time taken to sort 8500 numbers is 0.000821 Secs
Time taken to sort 9500 numbers is 0.000863 Secs
Time taken to sort 10500 numbers is 0.000918 Secs
Time taken to sort 11500 numbers is 0.000992 Secs
Time taken to sort 12500 numbers is 0.001044 Secs
Time taken to sort 13500 numbers is 0.001105 Secs
Time taken to sort 14500 numbers is 0.001276 Secs
```

Algorithm and Tracing :





Lab program 4 :

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

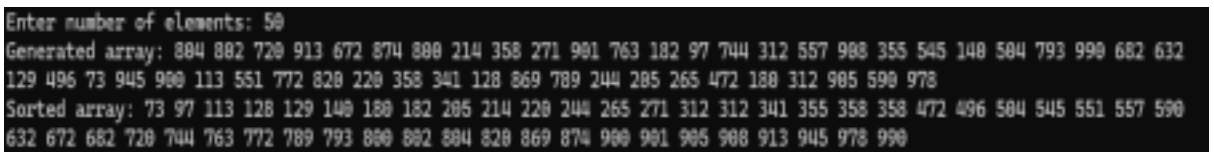
Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


// Function to swap two elements

void swap(int *a, int *b) {

    int temp = *a;

    *a = *b;

    *b = temp;

 }
// Partition function for Quick Sort

int partition(int arr[], int low, int high) {

    int pivot = arr[high]; // pivot element

    int i = (low - 1); // index of smaller element
```

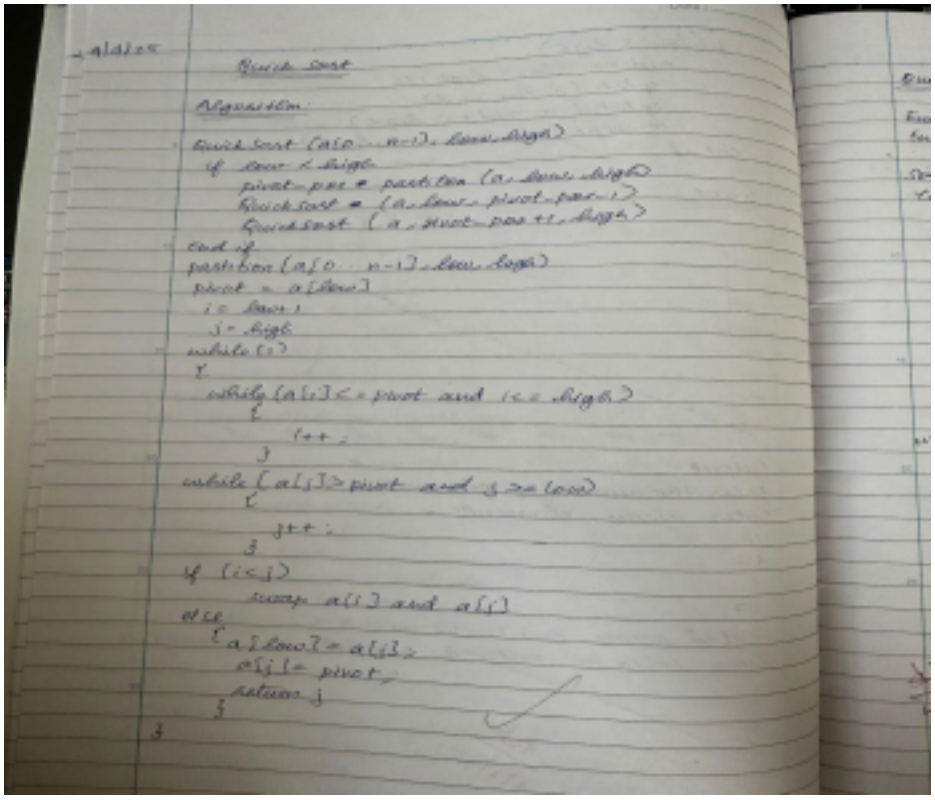
```
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // partitioning index
    quickSort(arr, low, pi - 1); // Recursively sort left subarray
    quickSort(arr, pi + 1, high); // Recursively sort right subarray
    }
} // Main function
int main() {
    int n; // Input the size of the array
    printf("Enter number of elements: ");
    scanf("%d", &n);
    int arr[n];

    // Seed the random number generator
    srand(time(0));
    // Generate random numbers for the array
    printf("Generated array: ");
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000; // Generate random numbers between 0 and 999
        printf("%d ", arr[i]);
    }
    printf("\n"); // Measure time taken for sorting
    clock_t start = clock();
    quickSort(arr, 0, n - 1);
```

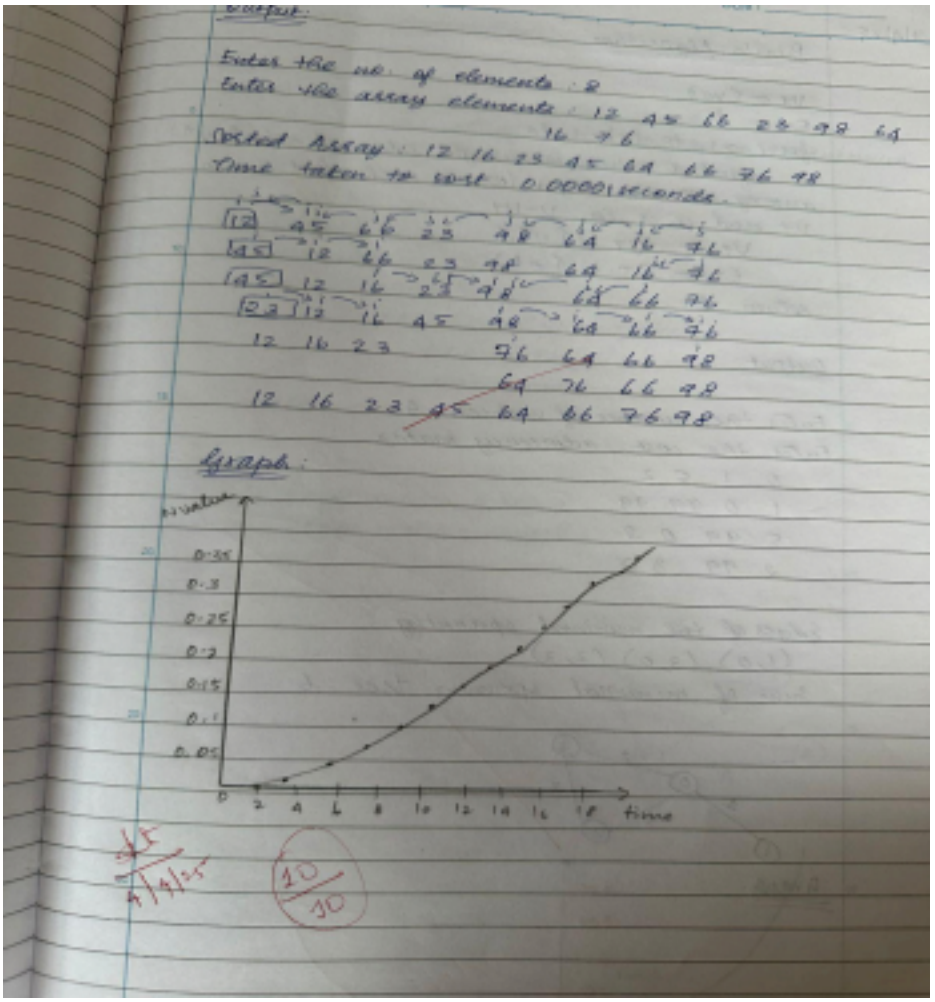
```
    clock_t end = clock();
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Output the sorted array
    printf("Sorted array: ");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Output the time taken
    printf("Time taken to sort: %f seconds\n", time_taken);

    return 0;
}
```

Result:



Algorithm and Tracing :





LEETCODE: 3Sum

Code:

```
var threeSum = function (nums) {
```

```
  nums.sort((a, b) => a - b);
  const op = [];
  for (let i = 0; i < nums.length; i++) {
    if (i > 0 && nums[i] === nums[i - 1]) continue;
    const target = -nums[i];
    let left = i + 1, right = nums.length - 1;
    while (left < right) {
      const current_sum = nums[left] + nums[right];
      if (current_sum === target) {
        op.push([nums[i], nums[left], nums[right]]);
        while (left < right && nums[left] === nums[left + 1]) left++;
        while (left < right && nums[right] === nums[right - 1]) right--;
        left++;
        right--;
      } else if (current_sum < target) {
        left++;
      } else {
        right--;
      }
    }
  }
  return op;
};
```

Result:

Lab program 5 :

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

Code:

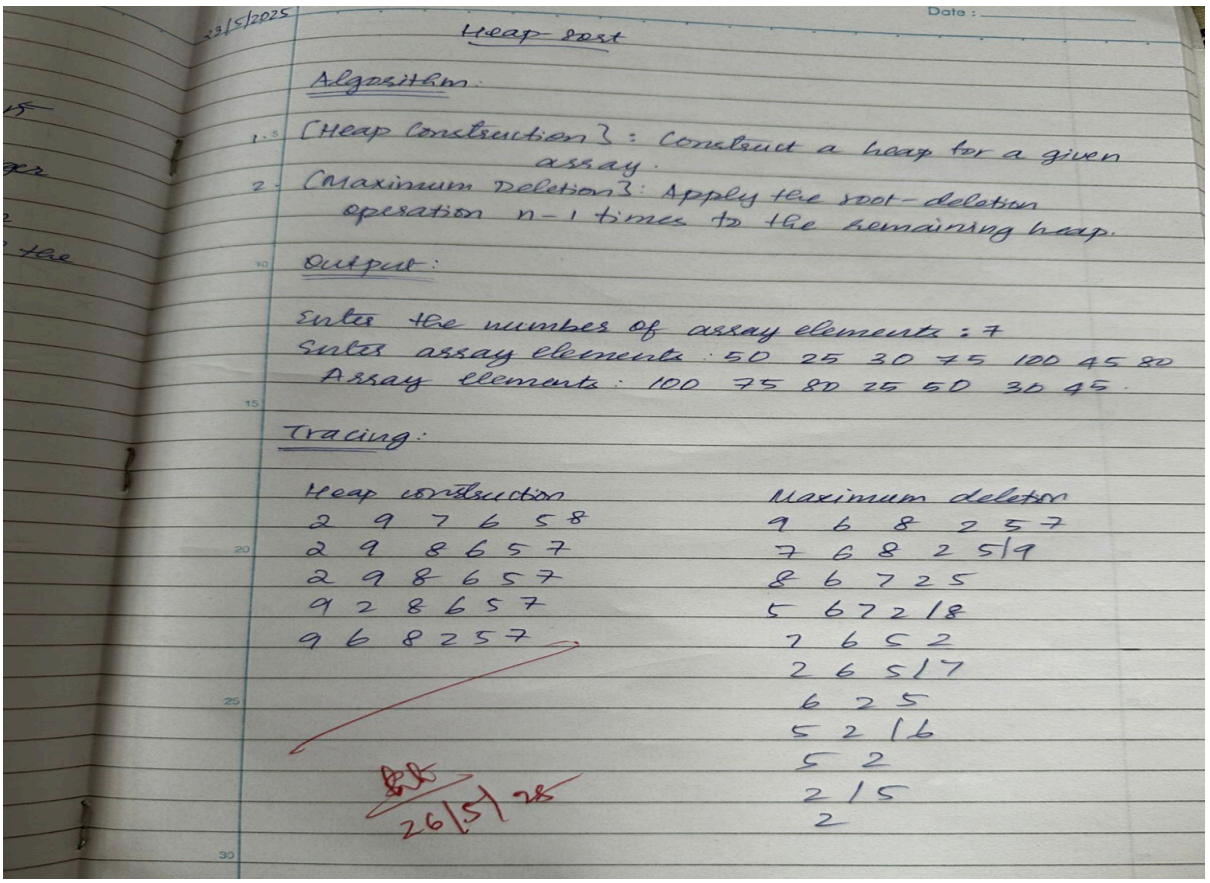
```
#include <stdio.h>

#include <time.h>

#define MAX 1000

int a[MAX];

int n;
void heapify(int a[], int n, int i) {

 int largest = i;

 int left = 2 * i + 1;

 int right = 2 * i + 2;

 if (left < n && a[left] > a[largest])

 largest = left;

 if (right < n && a[right] > a[largest])

 largest = right;

 if (largest != i) {

 int temp = a[i];

 a[i] = a[largest];

 a[largest] = temp;

 heapify(a, n, largest);

 }

 }


void heapSort(int a[], int n) {
 // Build max heap

 for (int i = n / 2 - 1; i >= 0; i--)

 heapify(a, n, i);

 for (int i = n - 1; i >= 1; i--) {

 int temp = a[0];

 a[0] = a[i];
```

```
 a[i] = temp;
 heapify(a, i, 0);
 }
 }

int main() {
 clock_t start, end;
 double time_taken;
 printf("Enter number of elements: ");
 scanf("%d", &n);

 if (n > MAX) {
 printf("Error: Maximum number of elements is %d.\n",
MAX); return 1;
 }
 printf("Enter %d elements:\n", n);
 for (int i = 0; i < n; i++)
 scanf("%d", &a[i]);
 start = clock();
 heapSort(a, n);
 end = clock();
 time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
 printf("Sorted array:\n");
 for (int i = 0; i < n; i++)
 printf("%d ", a[i]);
 printf("\n");
 printf("Time taken to sort using Heap Sort: %f seconds\n", time_taken); return 0;
}
```

Result:

```
Enter number of elements: 6
Enter 6 elements:
5 12 45 23 31 10
Sorted array:
5 10 12 23 31 45
Time taken to sort using Heap Sort: 0.000002 seconds
```

Algorithm and Tracing :



Lab program 6 :

Implement 0/1 Knapsack problem using dynamic programming. Code:

```
#include <stdio.h>

int n, m, w[10], p[10], v[11][11];

void knapsack(int, int, int[], int[]);
```

```
int max(int, int);

int main() {

 int i, j;

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

  printf("Enter the capacity of the knapsack: ");

  scanf("%d", &m);

 printf("Enter weights:\n");

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

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

 }

 printf("Enter profits:\n");

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

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

 }

 knapsack(n, m, w, p);

 printf("\nDP Table:\n");

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

 for (j = 0; j <= m; j++) {

 printf("%3d ", v[i][j]);

 }
 printf("\n");

 }

printf("\nOptimal Profit: %d\n", v[n][m]);

return 0;

 }

void knapsack(int n, int m, int w[], int p[])

 { int 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 (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]); }}}}

int max(int a, int b) {

return (a > b) ? a : b;}
```

Result:

```
Enter the number of items: 4
Enter the capacity of the knapsack: 5
Enter weights:
2 1 3 2
Enter profits:
12 10 20 15

DP Table:
  0   0   0   0   0   0
  0   0  12  12  12  12
  0  10  12  22  22  22
  0  10  12  22  30  32
  0  10  15  25  30  37

Optimal Profit: 37
```

Algorithm and Tracing:

16/5/2025

Knapsack Problem using dynamic programming

Algorithm:

Input: n - No. of objects to be selected
m - capacity of knapsack
w - weight of all objects
p - profits of all objects
v - optimal solution for the numbers of objects selected with specified remaining capacity.

```
for i=0 -> n do
    for j=0 -> m do
        if (i=0 or j=0)
            v[i,j]=0
        elseif (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])
        end if
    end for
end for.
```

output:

Enter the no. of items : 4
Enter the capacity of knapsack : 5
Enter weights : 2 1 3 2
Enter profits : 12 10 20 15
Optimal solution:

0	0	0	0
0	10	10	10
0	10	10	20
0	10	15	25

Tracing:

Item	w
1	
2	
3	
4	

0	0
1	0
2	
3	
4	

N=4

of
the o
selecte
profit

Tracing:

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

$$v[i,j] = \begin{cases} 0 & (i=j=0 \\ v[i-1,j] & (wi>j) \\ \max(v[i-1,j], v[i-1,j-wi] + pi & (wi \le j) \end{cases}$$

M=5

	0	1	2	3	4	5
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
4	0	10	15	25	30	~~37~~

N=4

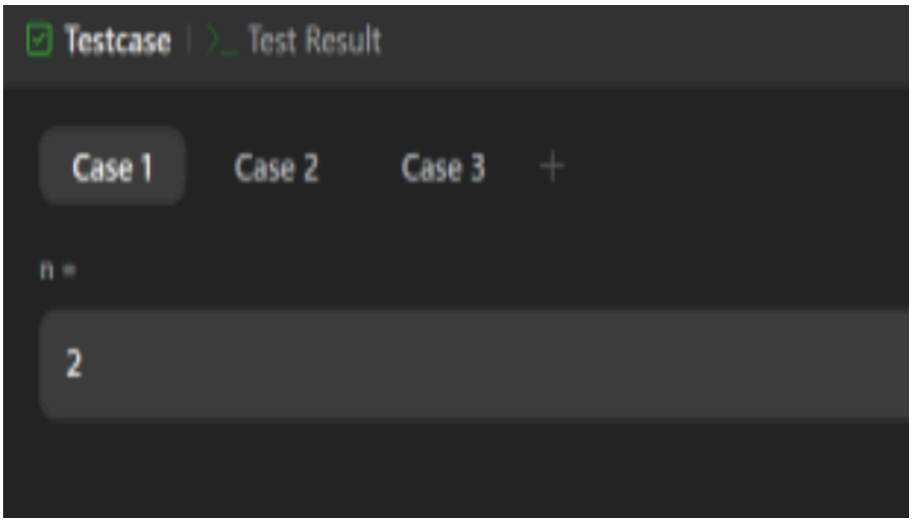
of
The optimal solution is ~~37~~ which is acheived by selected item 2 & 3 with weights (1 & 3) and profit 10 & 20).

LEETCODE: Fibonacci Number

Code:

```
int fib(int n) {
if (n == 0) return 0;
if (n == 1) return 1;

int a = 0, b = 1, c;

// Calculate Fibonacci iteratively
for (int i = 2; i <= n; i++) {
c = a + b; // Fibonacci relation
a = b; // Update a to previous b
b = c; // Update b to current Fibonacci number
}
return b; // Return the nth Fibonacci number
}
```

Result:



Lab program 7 :

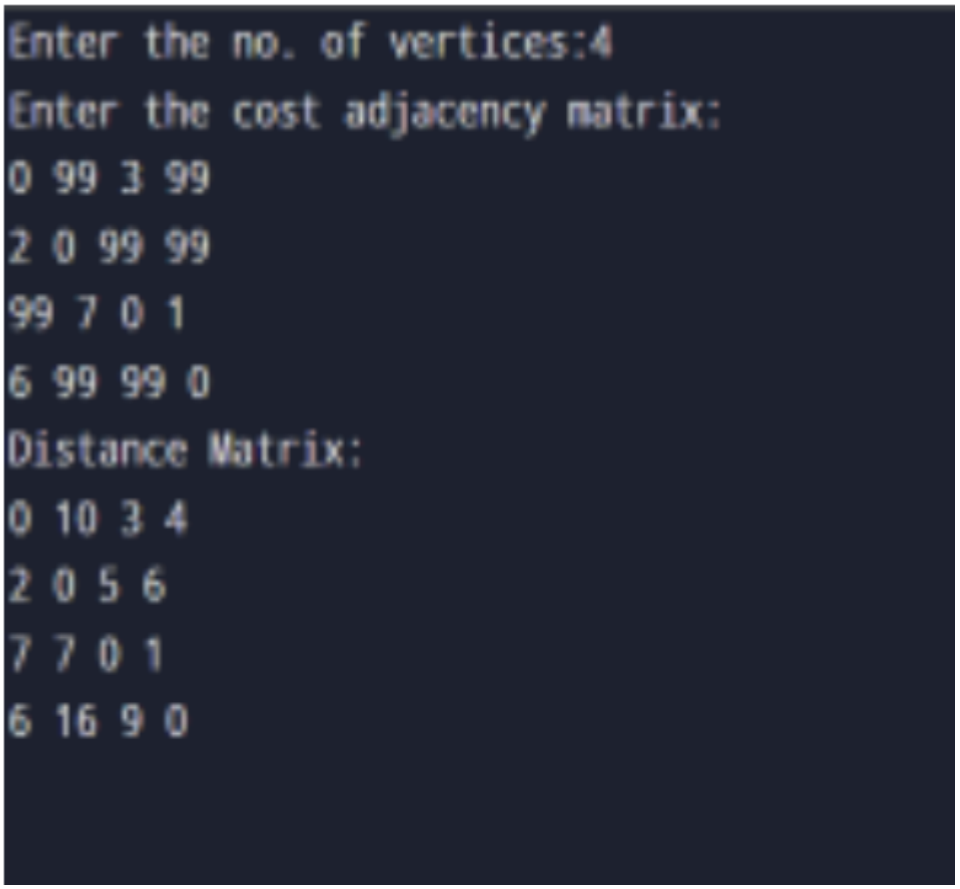
Implement All Pair Shortest paths problem using Floyd's algorithm. Code:

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

```
void floyd(int [][10],int);
int min(int,int);
int main()
 {
printf("Enter the no. of vertices:");
 scanf("%d",&n);
printf("Enter the cost adjacency matrix:\n");
int i,j;
for(i=0;i<n;i++){
for(j=0;j<n;j++){
 scanf("%d",&a[i][j]);
 }
 }
floyd(a,n);
printf("Distance Matrix:\n");
for(i=0;i<n;i++){
for(j=0;j<n;j++){
printf("%d ",D[i][j]);
 }
printf("\n");
 }
return 0;
 }
void floyd(int a[][10],int n){
int i,j,k;
for(i=0;i<n;i++){
for(j=0;j<n;j++){
D[i][j]=a[i][j];
 }
```

```
}
for(k=0;k<n;k++){
for(i=0;i<n;i++){
for(j=0;j<n;j++){
D[i][j]=min(D[i][j],(D[i][k]+D[k][j]));
} } } }
int min(int a,int b){
 if(a<b){
 return a;
 }else{
 return b;
 } }
```

Result:



Algorithm and Tracing :

Date :

16/5/2025

Floyd's Algorithm

Input: The weight matrix W of a graph with no negative-length cycle.
Output: The distance matrix of the shortest path's length.

```
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,j], D[i,k] + D[k,j]}
return D.
```

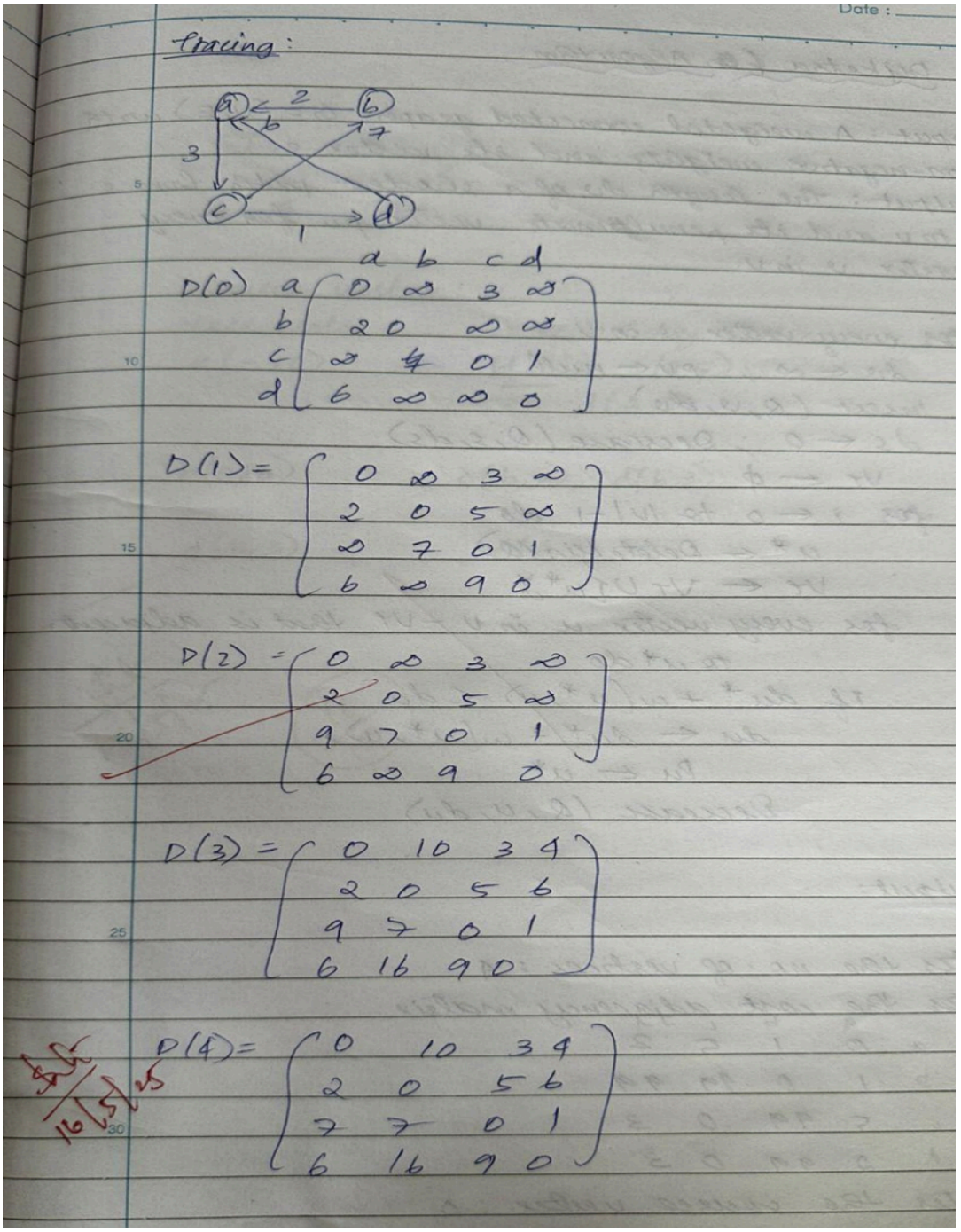
Output:

Enter the no of vertices: 4
Enter the cost of adjacency matrix:

0	99	3	99
2	0	99	99
99	6	0	1
7	99	99	0

Distance Matrix

0	9	3	4
2	0	5	6
8	6	0	1
7	16	10	0



LEETCODE: Shortest Path Visiting All Nodes

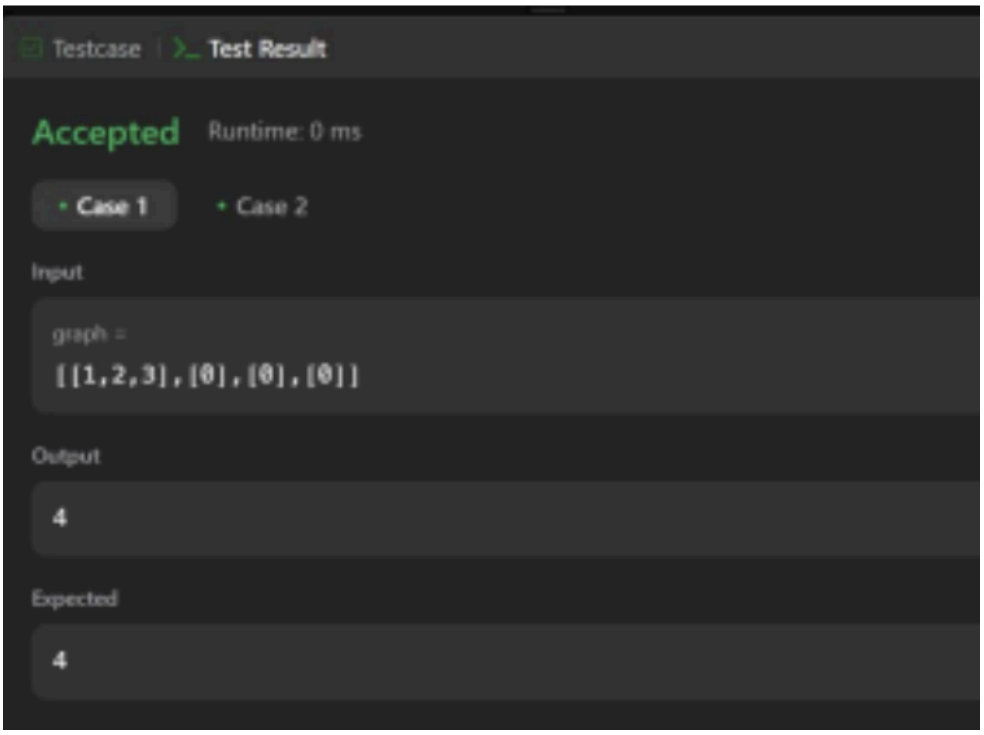
Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>


#define MAXN 12

#define MAXQ (1 << MAXN) * MAXN


typedefstruct {
```

```
    int node;
    int mask;
    int dist;
} State;

int shortestPathLength(int** graph, int graphSize, int* graphColSize)
    { int allVisited = (1 << graphSize) - 1;
    bool visited[MAXN][1 << MAXN] = { false };
    State queue[MAXQ];
    int front = 0, rear = 0;

    // Initialize queue with each node as starting point
    for (int i = 0; i < graphSize; i++) {
        int mask = 1 << i;
        queue[rear++] = (State){i, mask, 0};
        visited[i][mask] = true;
    }

    while (front < rear) {
        State curr = queue[front++];
        if (curr.mask == allVisited) {
            return curr.dist;
        }

        for (int i = 0; i < graphColSize[curr.node]; i++) {
            int neighbor = graph[curr.node][i];
            int nextMask = curr.mask | (1 << neighbor);

            if (!visited[neighbor][nextMask]) {
                visited[neighbor][nextMask] = true;
                queue[rear++] = (State){neighbor, nextMask, curr.dist +
            1}; }
        }
    }
```

```
    return -1; // Should never reach here
}
```

Result



Lab program 8 :

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

```
#include <stdio.h>

#define INF 999

int main() {

 int n, i, j, min, u, v;

 int cost[10][10], visited[10], parent[10], dist[10];

 int totalCost = 0;

 printf("Enter number of vertices: ");

 scanf("%d", &n);

 printf("Enter cost adjacency matrix (use 999 for no edge):\n");
 for (i = 0; i < n; i++)

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

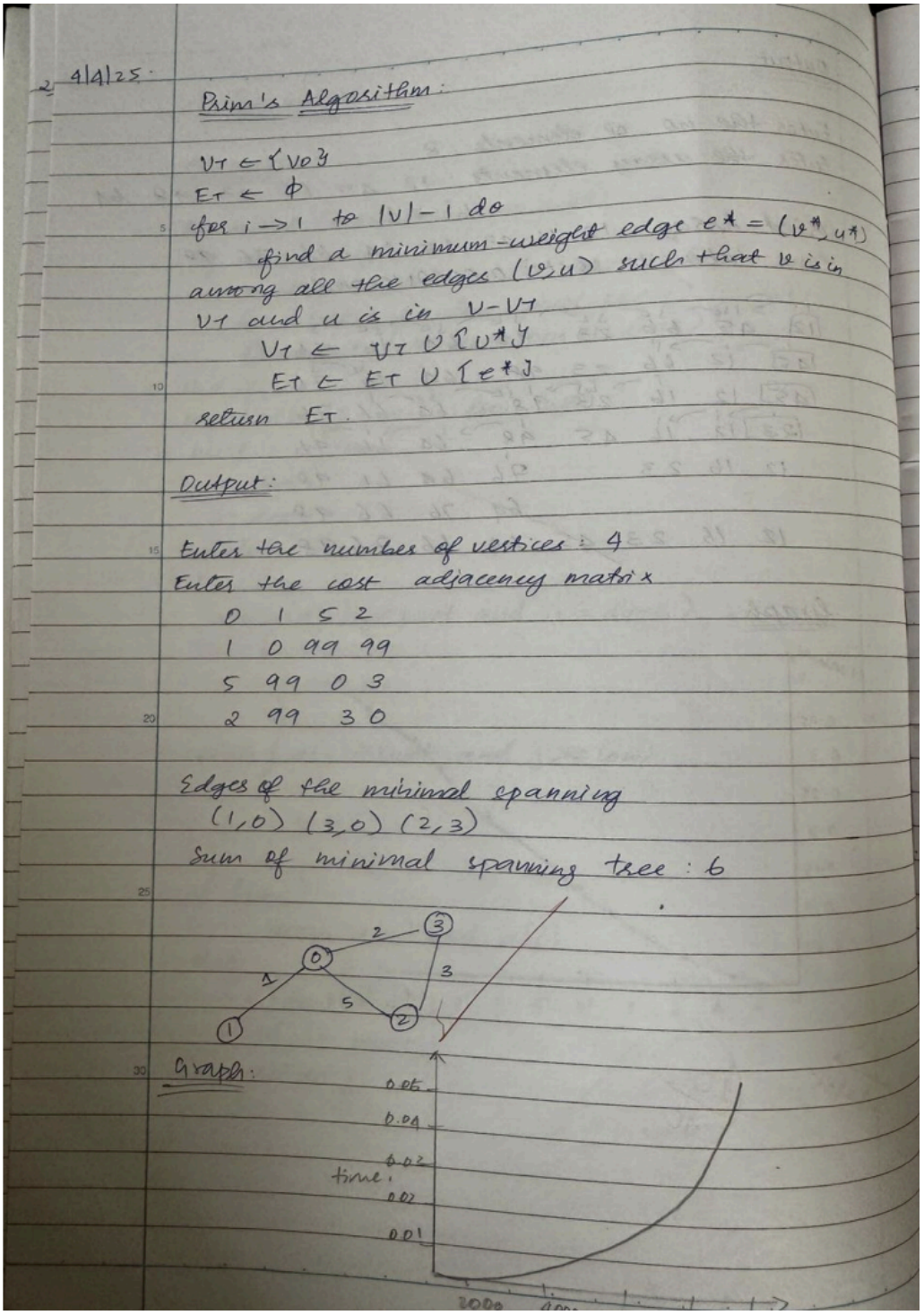
```
scanf("%d", &cost[i][j]);

// Initialize

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

dist[i] = cost[0][i];

parent[i] = 0;

visited[i] = 0;

}

visited[0] = 1; // Start from vertex 0

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

min = INF;

u = -1;

// Find the unvisited vertex with the smallest edge weight

for (j = 0; j < n; j++) {
if (!visited[j] && dist[j] < min) {

min = dist[j];

u = j;

}

}

if (u == -1) break; // Graph is disconnected

visited[u] = 1;

totalCost += dist[u];

printf("Edge selected: (%d, %d) cost = %d\n", parent[u], u, dist[u]); //

Update distance of adjacent vertices

for (v = 0; v < n; v++) {

if (!visited[v] && cost[u][v] < dist[v]) {
dist[v] = cost[u][v];

parent[v] = u;

}

}
```

```
}
printf("Total cost of Minimum Spanning Tree: %d\n", totalCost);
return 0;
}
```

Result:

```
Enter number of vertices: 4
Enter cost adjacency matrix (use 999 for no edge):
0 2 999 6
2 0 3 8
999 3 0 1
6 8 1 0
Edge selected: (0, 1) cost = 2
Edge selected: (1, 2) cost = 3
Edge selected: (2, 3) cost = 1
Total cost of Minimum Spanning Tree: 6
```

Algorithm and Tracing :



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

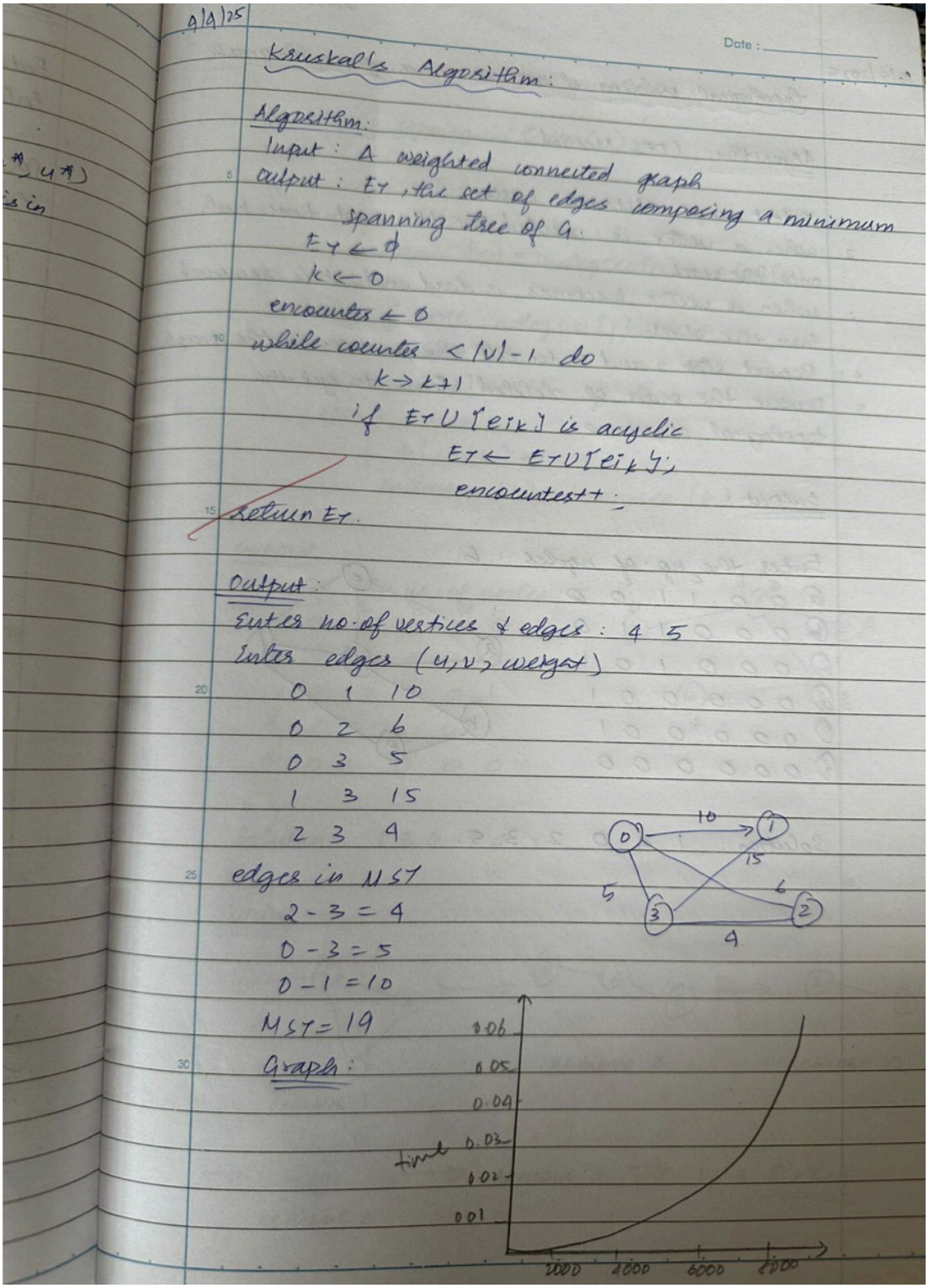
Code:

```
#include <stdio.h>

#define INF 999

int cost[10][10], parent[10];

int n; // Number of vertices

// Function to find the root of a vertex (used in union-find)

int find(int i) {
```

```
 while (parent[i] != i)
 i = parent[i];
 return i;
 }
// Function to join two sets (union)
void unionSets(int i, int j) {
 int root_i = find(i);
 int root_j = find(j);
 parent[root_i] = root_j;
 }
void kruskal() {
 int edges = 0, totalCost = 0;
 // Initialize each vertex's parent to itself
 for (int i = 0; i < n; i++)
 parent[i] = i;
 printf("Edges in the Minimum Spanning Tree:\n");
 while (edges < n - 1) {
 int min = INF, u = -1, v = -1;
 // Find the edge with the minimum cost
 for (int i = 0; i < n; i++) {
 for (int j = 0; j < n; j++) {
 if (i != j && cost[i][j] < min && find(i) != find(j)) { min =
cost[i][j];
 u = i;
 v = j;
 }} }
 if (u != -1 && v != -1) {
 unionSets(u, v);
 printf("(%d, %d) -> cost = %d\n", u, v, min);
 totalCost += min;
 edges++;
```

```
// Mark edge as used
 cost[u][v] = cost[v][u] = INF;

 } else {

 break; // No valid edge found

 } }

 printf("Total cost of the Minimum Spanning Tree: %d\n", totalCost);

 }

int main() {

 printf("Enter the number of vertices: ");

 scanf("%d", &n);

 printf("Enter the cost adjacency matrix (999 for no edge):\n");

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

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

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

 kruskal();

 return 0;

 }
```

Result:



Lab program 9 :
Implement Fractional Knapsack using Greedy

technique. Code:

```
#include <stdio.h>

void knapsack(int n, int p[], int w[], int W) {

 int used[n];

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

```
used[i] = 0;
int cur_w = W;
float tot_v = 0.0;
int i, maxi;
while (cur_w > 0) {
maxi = -1;
for (i = 0; i < n; ++i) {
if ((used[i] == 0) &&
((maxi == -1) || ((float)p[i] / w[i] > (float)p[maxi] / w[maxi])))
maxi = i;
}
used[maxi] = 1;
if (w[maxi] <= cur_w) {
cur_w -= w[maxi];
tot_v += p[maxi];
printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi
+ 1, w[maxi], p[maxi], cur_w);
} else {
int taken = cur_w;
cur_w = 0;
tot_v += ((float)taken / w[maxi]) * p[maxi];
printf("Added %d%% (%d, %d) of object %d in the bag.\n",
(int)((float)taken / w[maxi] * 100), w[maxi], p[maxi], maxi + 1); }
}
printf("Filled the bag with objects worth %.2f.\n", tot_v);
}
int main() {
int n, W;
printf("Enter the number of objects: ");
```

```
scanf("%d", &n);
int p[n], w[n];
printf("Enter the profits of the objects: ");
for (int i = 0; i < n; i++) {
scanf("%d", &p[i]);
}
printf("Enter the weights of the objects: ");
for (int i = 0; i < n; i++) {
scanf("%d", &w[i]);
}
printf("Enter the maximum weight of the bag: ");
scanf("%d", &W);
knapsack(n, p, w, W);
return 0;
}
```

Result:

tab

Date :

23/5/2025

Fractional knapsack using Greedy Technique

Algorithm:

* Consider all the item with their weights and profits mentioned.
* Calculate Pi/wi of all the items & sort the items in descending order based on their Pi/wi values.
* Without exceeding the limit, add the items into the knapsack.
* If the knapsack can still store some weight, but the weights of other items exceed the limit, the fractional part of the next time can be added

Output:

Enter the no of objects : 7
Enter the profit of the objects:
5 10 15 7 8 9 4
Enter the weight of the objects:
1 3 5 4 1 3 2
Enter the maximum weight of the bag : 15

Added 6 4 (4,7) in bags. space left : 11
Added 7(2,4) in bag. space left : 9
Added 3(5,15) in bag. space left: 4
Added 6(3,9) in bag. space left: 1
Added 33% (3,10) of object 2.

Filled the bag with object worth 36.00

Date :

Tracing

Item	1	2	3	4	5
weight	3	3	2	5	1
Profit	10	15	10	20	8
Pi/wi	3.3	5	5	4	8

weight = [(1*1) + (1*3) + (1*2) + (4/5*5)]
= 10

Max. profit = [(1*8) + (1*15) + (1*10) + (4/5*20)] = 37.

LEETCODE: Largest Odd Number in String

Code:

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

// Traverse from the end to find the rightmost odd digit
for (int i = len - 1; i >= 0; i--) {
if ((num[i] - '0') % 2 == 1) {
// Temporarily terminate the string at the right place
num[i + 1] = '\0';
return num;
}
}
return ""; // No odd digit found
}
```

Lab program 10 :

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

Code:

```
#include <stdio.h>
#define MAX 10
#define INF 999
int cost[MAX][MAX], n, result[MAX][2], weight[MAX];
void dijkstras(int cost[][MAX], int s);
int main() {
 int i, j, s;
 printf("Enter the number of vertices: ");
 scanf("%d", &n);
 if (n > MAX) {
 printf("Maximum supported vertices is %d.\n", MAX);
 return 1;
```

```
}
printf("Enter the cost adjacency matrix (use 999 for no edge):\n");
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
scanf("%d", &cost[i][j]);
printf("Enter the source vertex: ");
scanf("%d", &s);
dijkstras(cost, s);
printf("\nShortest paths from source vertex %d:\n", s);
for (i = 0; i < n; i++) {
if (i != s)
printf("(%d -> %d) with weight %d\n", result[i][0], result[i][1], weight[i]);
}
return 0;
}
void dijkstras(int cost[][MAX], int s) {
int d[MAX], visited[MAX], p[MAX];
int i, j, u, v, min;
for (i = 0; i < n; i++) {
d[i] = INF;
visited[i] = 0;
p[i] = s;
}
d[s] = 0;
for (i = 0; i < n - 1; i++) {
min = INF;
u = -1;
// Find unvisited vertex with smallest distance for
(j = 0; j < n; j++) {
```

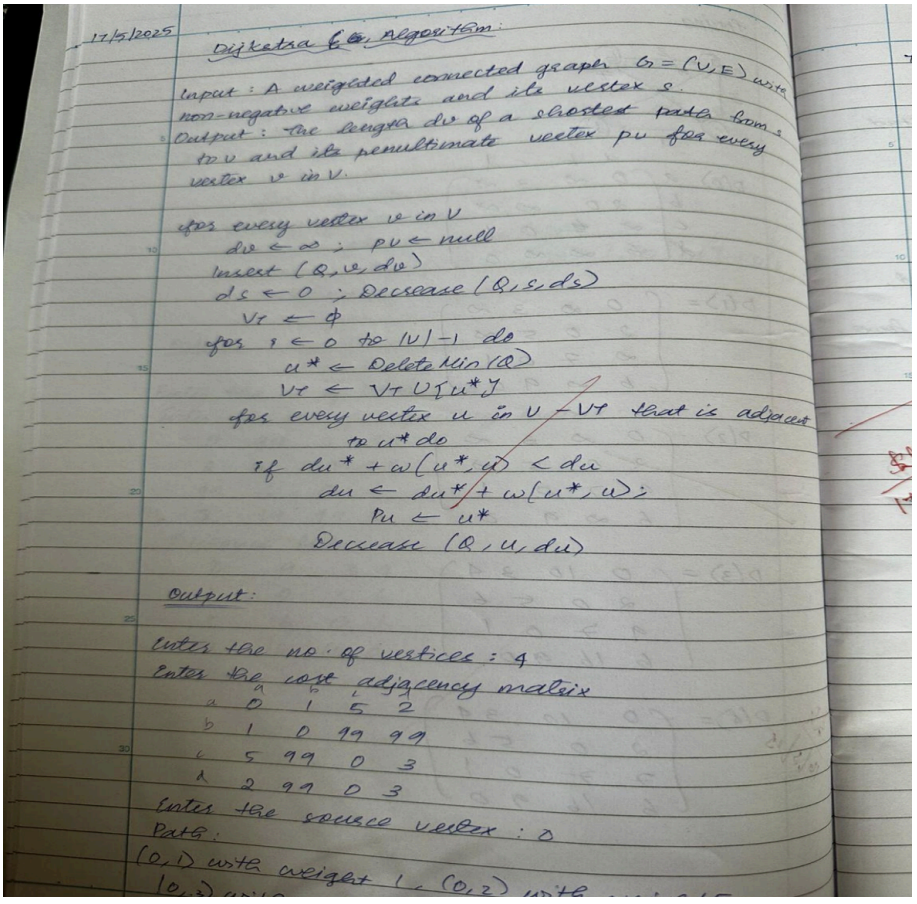
```
if (!visited[j] && d[j] < min) { min
= d[j];
u = j;
}
}
if (u == -1)
break;
visited[u] = 1;
// Update distances
for (v = 0; v < n; v++)
if (!visited[v] && cost[u][v] != INF && d[u] + cost[u][v] < d[v]) { d[v] = d[u] + cost[u][v];
p[v] = u;
}
}
}
// Store the result
for (i = 0; i < n; i++) {
result[i][0] = p[i];
result[i][1] = i;
weight[i] = d[i];
}
}
```

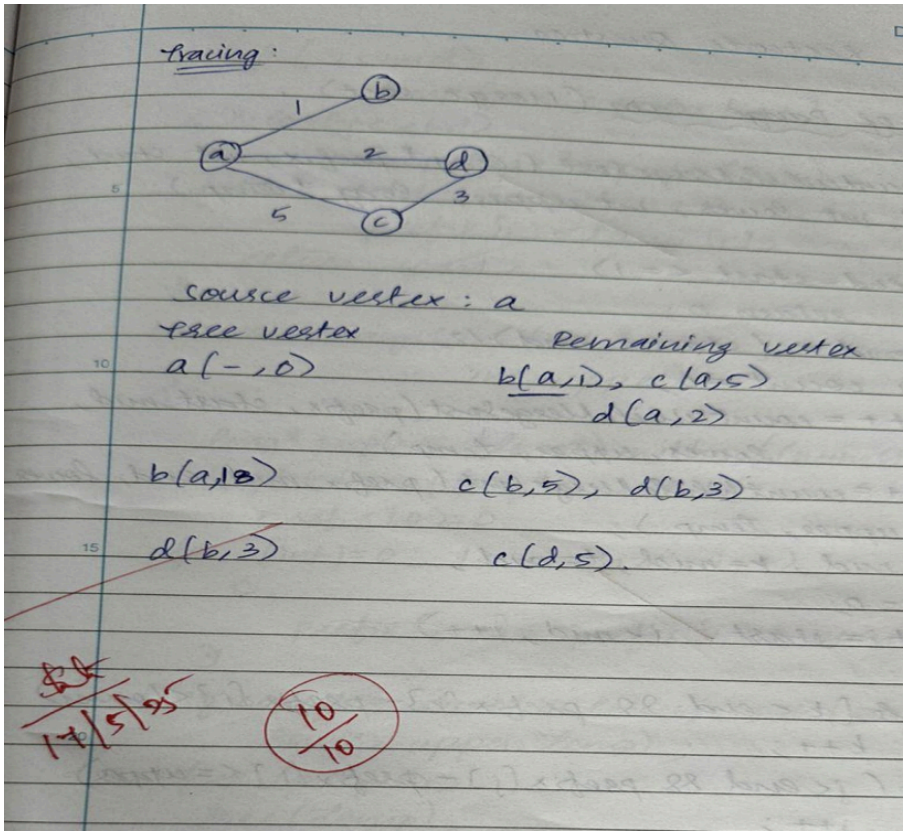
Result:

Output

```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Enter the source vertex: 0
Path:
(0, 1) with weight 1 (0, 2) with weight 5 (0, 3) with weight 2

=== Code Execution Successful ===
```

Algorithm and Tracing :





Lab program 11 :

Implement "N-Queens Problem" using

Backtracking. Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
bool isSafe(int** board, int row, int col, int N)
 { int i, j;
 // Check left side of the current row
 for (i = 0; i < col; i++)
 if (board[row][i])
 return false;
 // Check upper diagonal on left side
 for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
if (board[i][j])
 return false;
 // Check lower diagonal on left side
 for (i = row, j = col; j >= 0 && i < N; i++, j--)
if (board[i][j])
```

```
 return false;
 return true;
 }
void printBoard(int** board, int N) {
 for (int i = 0; i < N; i++) {
 for (int j = 0; j < N; j++)
 printf("%s ", board[i][j] ? "Q" : ".");
printf("\n");
 }
 printf("\n");
 }
void solveNQueensAll(int** board, int col, int N, int* solutionCount)
 { if (col == N) {
 (*solutionCount)++;
 printf("Solution %d:\n", *solutionCount);
 printBoard(board, N);
 return;
 }
 for (int i = 0; i < N; i++) {
 if (isSafe(board, i, col, N)) {
 board[i][col] = 1;
 solveNQueensAll(board, col + 1, N, solutionCount);
board[i][col] = 0; // Backtrack
 }
 }
 }
int main() {
 int N;
 printf("Enter the number of queens: ");
 scanf("%d", &N);
 // Dynamically allocate NxN board
 int** board = (int**)malloc(N *
```

```
sizeof(int*)); for (int i = 0; i < N; i++) {
board[i] = (int*)calloc(N, sizeof(int));
}
int solutionCount = 0;
solveNQueensAll(board, 0, N,
&solutionCount); if (solutionCount == 0)
printf("No solutions found for %d-Queens.\n", N);
else
printf("Total solutions found: %d\n", solutionCount);
// Free allocated memory
for (int i = 0; i < N; i++)
free(board[i]);
free(board);

return 0;
}
```

Result:

Output

```
Enter the number of queens: 4
2 4 1 3
Solution found
3 1 4 2
Solution found
Total solutions: 2


=== Code Execution Successful ===
```

Algorithm and Tracing :

23/5/25

N- Queen Problem

Algorithm:

1. Start with the first row ($k = 1$).
2. Try placing a queen in the current row.
   * Move through each column from left to right.
   * For each column, check if placing a queen there is safe.
3. If a safe position is found:
   * Place the queen.
   * If this is the last row ($k == n$), print solution.
   * Otherwise, move to the next row ($k = k + 1$) & repeat step 2.
4. If no safe column in the current row:
   * Backtrack to the previous row ($k = k - 1$) & try the next column there.
5. Repeat steps 2-4 until all solutions are found.
6. Print the total no of solutions found.

Output:

```
Enter the number of queen : 4
  2  4  1  3
Solution found
  3  1  4  2
Solution found
Total solution : 2
```

Tracing:

Date :