VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

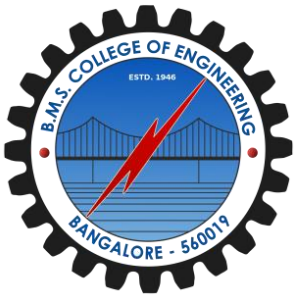
Analysis and Design of Algorithms

(22CS4PCADA)

Submitted by:

Sanchay Agrawal
(1BM21CS186)

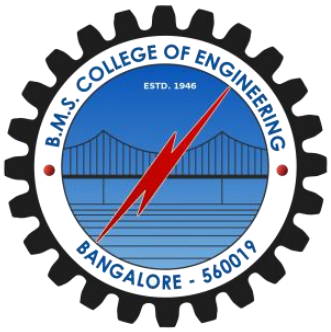
in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING



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

June 2023 - October 2023

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



CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **Sanchay Agrawal (1BM21CS186),** 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 2022-23. The Lab report has been approved as it satisfies the academic requirements in respect of **Analysis and Design of Algorithms - (22CS4PCADA)** work prescribed for the said degree.

Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Table Of Contents

1. 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.

2. Experiments

2.1 Experiment - 1

2.1.1 Question:

Write program to do the following:

(a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

(b) Check whether a given graph is connected or not using DFS method.

2.1.2 Code:

(a) <u>BFS method</u>:

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

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v)
{
        for(i=1;i<=n;i++)
        if(a[v][i] && !visited[i])
        q[++r]=i;
        if(f<=r)
        {
                visited[q[f]]=1;
                bfs(q[f++]);
        }
}

void main()
{
        int v;

        printf("\n Enter the number of vertices:");
```

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

	for(i=1;i<=n;i++)
	{
		q[i]=0;
		visited[i]=0;
	}

	printf("\n Enter graph data in matrix form:\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&a[i][j]);

	printf("\n Enter the starting vertex:");
	scanf("%d",&v);
	bfs(v);

	printf("\n The node which are reachable are:\n");
	for(i=1;i<=n;i++)
		if(visited[i])
		printf("%d\t",i);

	getch();
}
```

(b) <u>DFS method</u>:

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

int a[20][20],reach[20],n;

void dfs(int v)
{
	int i;
	reach[v]=1;
	for(i=1;i<=n;i++)
	if(a[v][i]&&!reach[i])
	{
		printf("\n%d->%d",v,i);
		dfs(i);
	}
}
```

```
int main()
{
	int i,j,count=0;
	printf("\nEnter no of vertices : ");
	scanf("%d",&n);

	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
		{
			reach[i]=0;
			a[i][j]=0;
		}
	printf("\nEnter adjacency matrix : \n");

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

	for(i=1;i<=n;i++)
	if(reach[i])
	count++;

	if(count==n)
		printf("\nGraph is connected.");
	else
		printf("\nGraph is disconnected.");

	getch();
	return(0);
}
```

2.1.3 Output:

(a) BFS method:

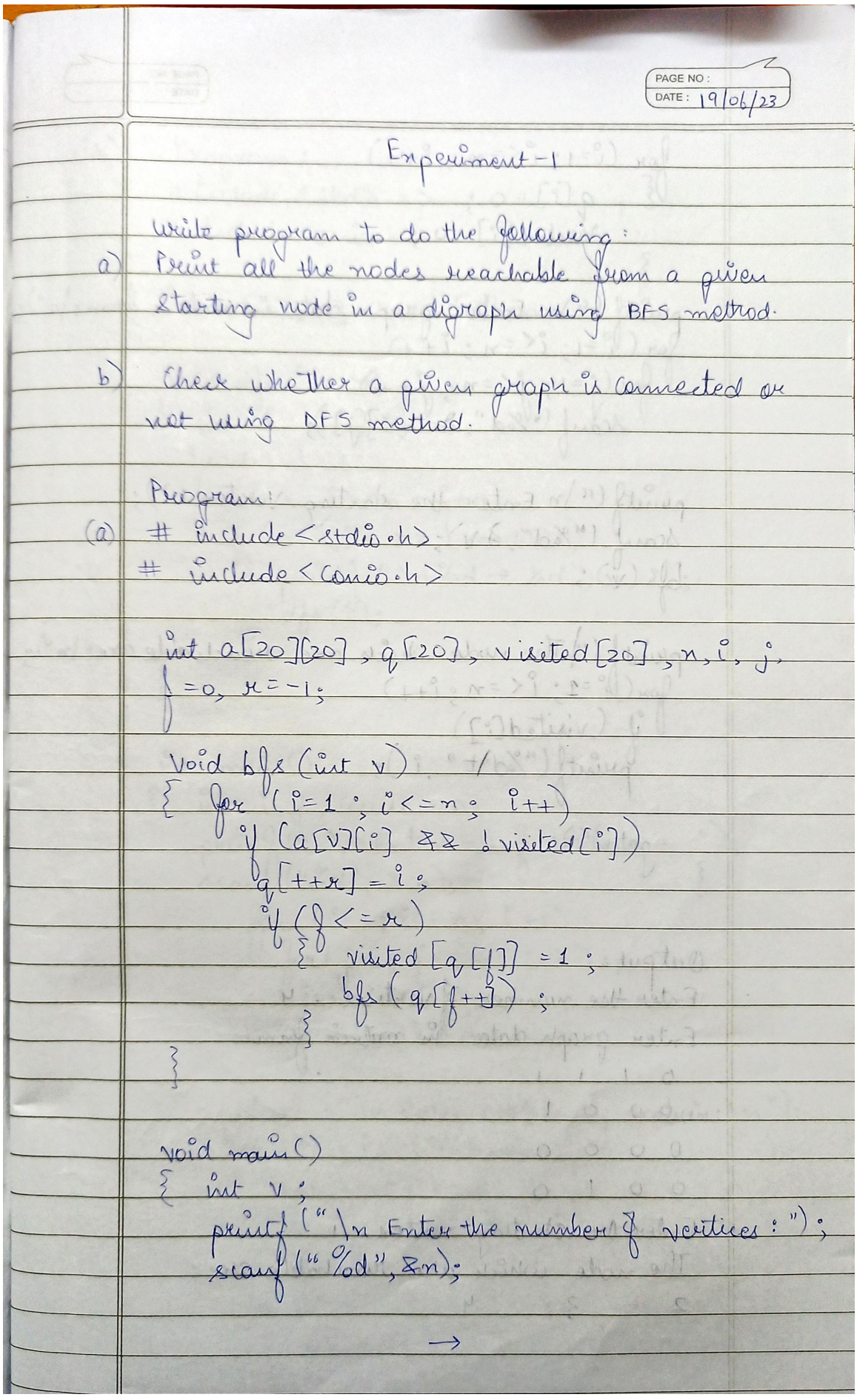
```
 Enter the number of vertices:4

 Enter graph data in matrix form:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

 Enter the starting vertex:1

 The node which are reachable are:
2       3       4
```

DFS method

```
Enter no of vertices : 4

Enter adjacency matrix :
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

1->2
2->4
4->3
Graph is connected.
```

```
Enter no of vertices : 4

Enter adjacency matrix :
0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0

1->2
2->3
Graph is disconnected.
```

2.1.4 Observation Book Pictures:



PAGE NO :
DATE : 19/06/23

Experiment - 1

Write program to do the following:

a) Print all the nodes reachable from a given starting node in a digraph using BFS method.

b) Check whether a given graph is connected or not using DFS method.

Program:

(a)

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

int a[20][20], q[20], visited[20], n, i, j,
f=0, r=-1;

Void bfs (int v)
{ for (i=1; i<=n; i++)
    if (a[v][i] && !visited[i])
      q[++r] = i;
    if (f<=r)
      { visited [q[f]] = 1;
        bfs (q[f++]);
      }
}

void main()
{ int v;
  printf ("\n Enter the number of vertices : ");
  scanf ("%d", &n);
```

→

```
for (i=1; i<=n; i++)
{   q[i] = 0;
    visited[i] = 0;
}
printf("\n Enter graph data in matrix form:\n");
for(i=1; i<=n; i++)
  for(j=1; j<=n; j++)
    scanf("%d", &a[i][j]);

printf("\n Enter the starting vertex :");
scanf("%d", &v);
bfs(v);

printf("\n The node which are reachable are:\n");
for(i=1; i<=n; i++)
  if (visited[i])
    printf("%d\t", i);

  getch();
}
```

Output:

```
Enter the number of vertices : 4
Enter graph data in matrix form:
0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0
Enter the starting vertex : 1
The node which are reachable are:
2    3    4
```

PAGE NO :
DATE : 19/06/23

(b) Program :

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

int a[20][20], reach[20], n;

void dfs (int v)
{ int i;
  reach[v] = 1;
  for (i=1; i<=n ; i++)
    if (a[v][i] && ! reach[i])
      { printf("\n %d → %d", v, i);
        dfs (i);
      }
}

int main()
{ int i, j, count = 0;
  printf("\n Enter no of vertices : ");
  scanf("%d", &n);
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        { reach[i] = 0;
          a[i][j] = 0;
        }
  printf("\n Enter adjacency matrix:\n");
    for (i=1; i<=n; i++)
      for (j=1; j<=n; j++)
        scanf("%d", &a[i][j]);
      dfs(1);
    for (i=1; i<=n; i++)
      if (reach[i])
      count++;
```

```
if (count == n)
    printf("\n Graph is connected");
else
    Printf("\n Graph is disconnected");
getch();
return(0);
}
```

Output:

(i) Enter no. of vertices : 4

Enter adjacency matrix :

0 1 1 1
0 0 0 1
0 0 0 0
0 0 1 0

1 → 2
2 → 4
4 → 3
Graph is connected.

(ii) Enter no. of vertices : 4

Enter adjacency matrix :

0 1 0 0
0 0 1 0
0 0 0 0
1 1 1 0

1 → 2
2 → 3
Graph is Disconnected

O/P saw
26/6/23.

2.2 Experiment - 2

2.2.1 Question:

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

2.2.2 Code:

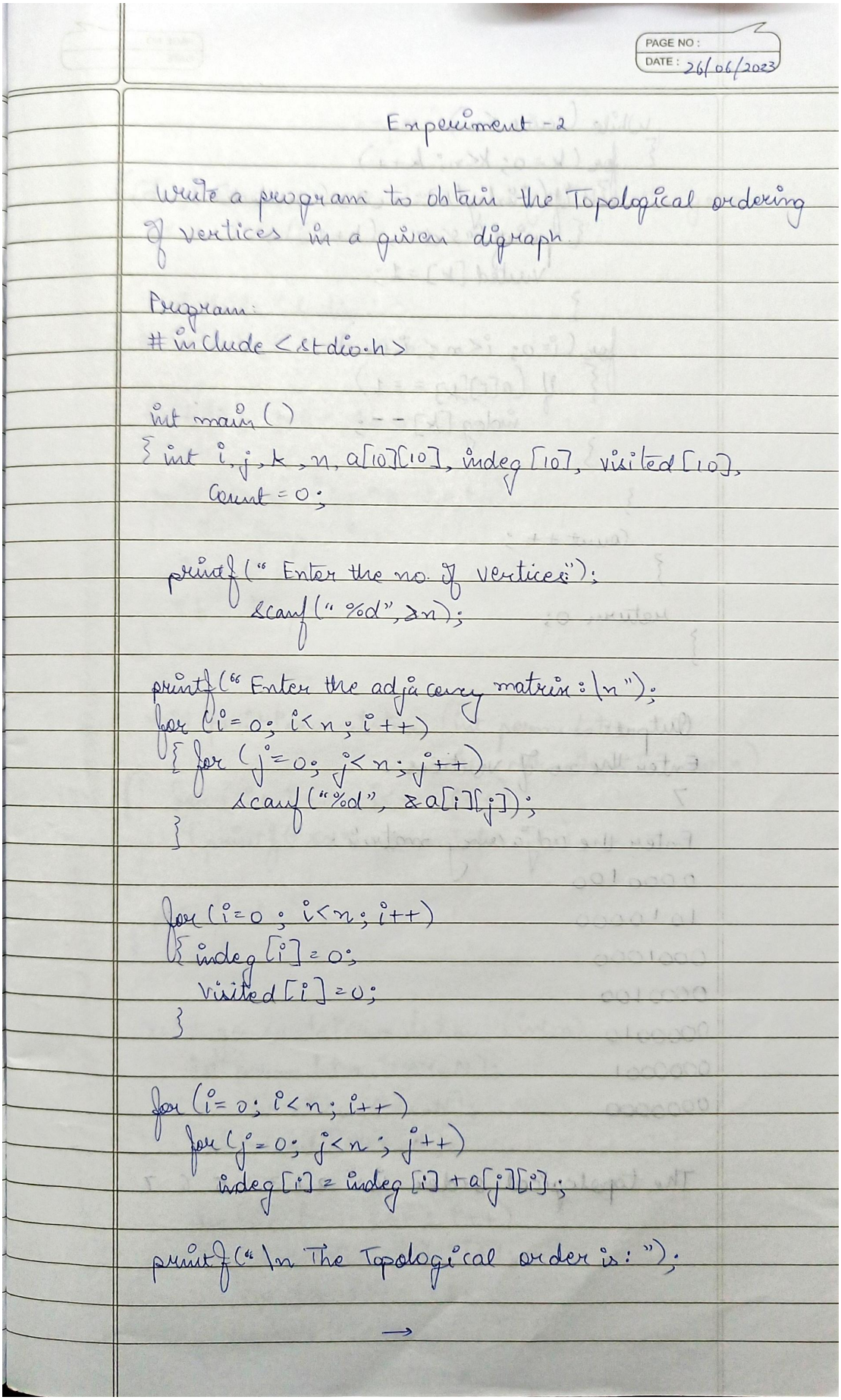
```
#include <stdio.h>

int main()
{
   int i,j,k,n,a[10][10],indeg[10],visited[10],count=0;

   printf("Enter the no of vertices:\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]);
   }

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

   for(i=0;i<n;i++)
      for(j=0;j<n;j++)
         indeg[i]=indeg[i]+a[j][i];

   printf("\nThe topological order is:");

   while(count<n)
      {
      for(k=0;k<n;k++)
      {
         if((indeg[k]==0) && (visited[k]==0))
         {
            printf("%d ",(k+1));
```

```
            visited [k]=1;
        }

        for(i=0;i<n;i++)
        {
            if(a[i][k]==1)
                indeg[k]--;
        }
    }
    count++;
  }
  return 0;
}
```

2.2.3 Output:

```
Enter the no of vertices:
7
Enter the adjacency matrix:
Enter row 1
0 0 0 0 1 0 0
Enter row 2
1 0 1 0 0 0 0
Enter row 3
0 0 0 1 0 0 0
Enter row 4
0 0 0 0 1 1 0
Enter row 5
0 0 0 0 0 1 0
Enter row 6
0 0 0 0 0 0 1
Enter row 7
0 0 0 0 0 0 0

The topological order is:2 1 3 4 5 6 7
```

2.2.4 Observation Book Pictures:



PAGE NO :
DATE : 26/06/2023

Experiment - 2

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

Program:

```
#include <stdio.h>

int main()
{ int i, j, k, n, a[10][10], indeg[10], visited[10],
    count = 0;

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

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

for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    indeg[i] = indeg[i] + a[j][i];

printf("\n The Topological order is:");

  →
```

```
while (count<n)
{ for (k=0; k<n; k++)
  { if ((indeg[k]==0 && (visited[k]==0))
    { printf("%d", (k+1));
      visited[k]=1;
    }
  for (i=0; i<n; i++)
  { if (a[i][k]==1)
      indeg[k]--;
  }
  }
  count++;
}
return 0;
}
```

Output:

Enter the no. of vertices:

7

Enter the adjacency matrix:

0 0 0 0 1 0 0

1 0 1 0 0 0 0

0 0 0 1 0 0 0

0 0 0 0 1 0 0

0 0 0 0 0 1 0

0 0 0 0 0 0 1

0 0 0 0 0 0 0

The topological order is: 2 1 3 4 5 6 7

2.3 Experiment - 3

2.3.1 Question:

Implement Johnson Trotter algorithm to generate permutations.

2.3.2 Code:

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

#define MAX_N 10

void swap(int *a, int *b)
{
   int temp = *a;
   *a = *b;
   *b = temp;
}

void printPermutation(int permutation[], int direction[], int n)
{
   for (int i = 0; i < n; i++)
   {
      printf("%d", permutation[i]);
   }
   printf("\n");
}

void generatePermutations(int n)
{
   int permutation[MAX_N];
   int direction[MAX_N];
   bool mobile[MAX_N];

   for (int i = 0; i < n; i++)
   {
      permutation[i] = i + 1;
      direction[i] = -1;
      mobile[i] = true;
   }

   printPermutation(permutation, direction, n);
```

```
    int mobileElement, mobileIndex, temp;

    while (true)
    {
        mobileElement = -1;
        mobileIndex = -1;

        for (int i = 0; i < n; i++)
        {
            if (direction[i] == -1 && i > 0 && permutation[i] > permutation[i - 1] && mobile[i])
            {
                if (mobileElement == -1 || permutation[i] > mobileElement)
                {
                    mobileElement = permutation[i];
                    mobileIndex = i;
                }
            }

            if (direction[i] == 1 && i < n - 1 && permutation[i] > permutation[i + 1] &&
mobile[i])
            {
                if (mobileElement == -1 || permutation[i] > mobileElement)
                {
                    mobileElement = permutation[i];
                    mobileIndex = i;
                }
            }
        }

        if (mobileIndex == -1)
        {
            break;
        }

        if (direction[mobileIndex] == -1)
        {
            swap(&permutation[mobileIndex], &permutation[mobileIndex - 1]);
            swap(&direction[mobileIndex], &direction[mobileIndex - 1]);
        }
```

```
        else
        {
            swap(&permutation[mobileIndex], &permutation[mobileIndex + 1]);
            swap(&direction[mobileIndex], &direction[mobileIndex + 1]);
        }

        for (int i = 0; i < n; i++)
        {
            if (permutation[i] > mobileElement)
            {
                direction[i] *= -1;
            }
        }
        printPermutation(permutation, direction, n);
    }
}

int main()
{
    int n;

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

    if (n < 1 || n > MAX_N)
    {
        printf("Invalid input!\n");
        return 0;
    }

    generatePermutations(n);
    return 0;
}
```

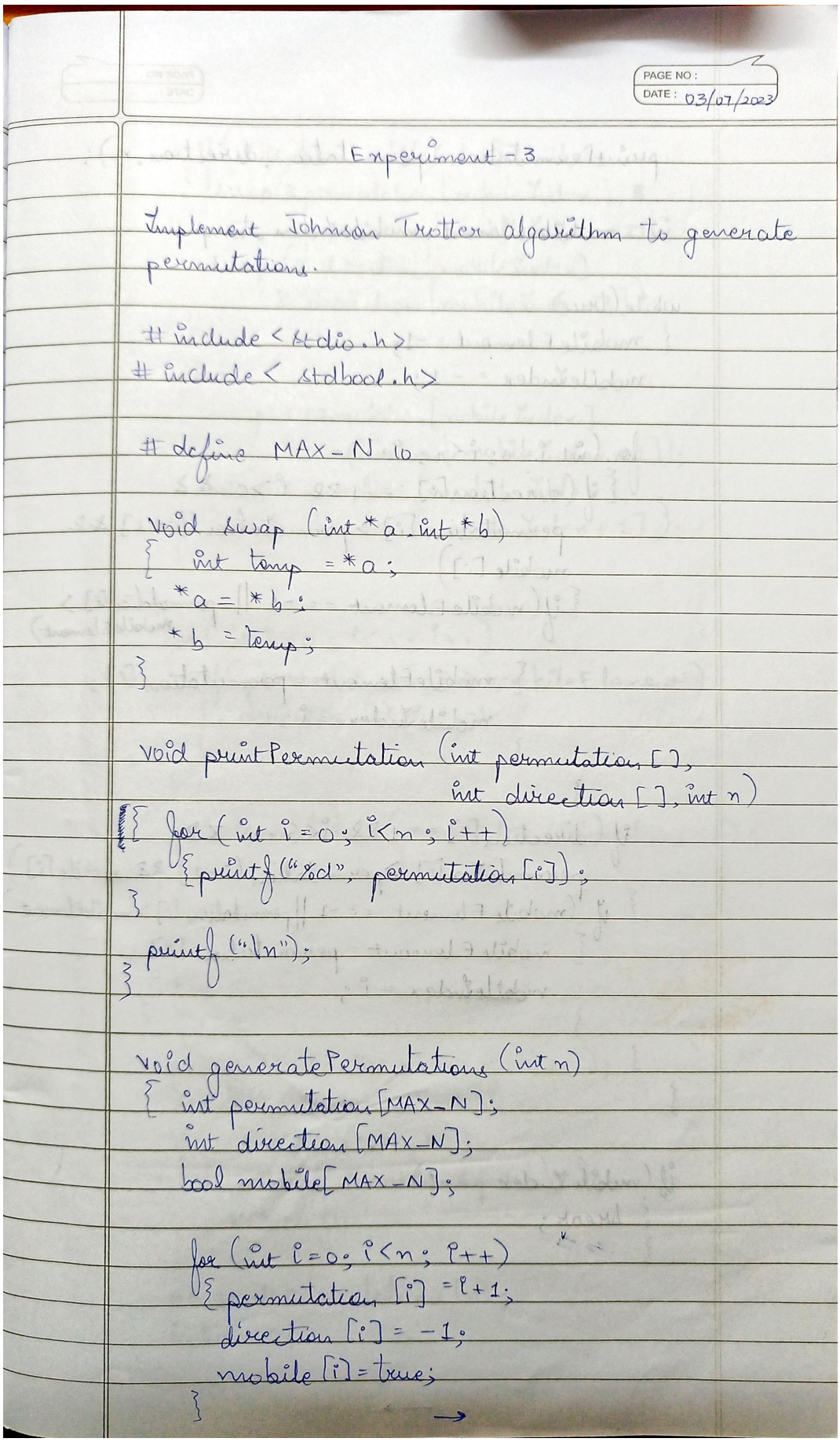
2.3.3 Output:

```
Enter the value of n: 2
12
21
```

```
Enter the value of n: 3
123
132
312
321
231
213
```

```
Enter the value of n: 4
1234
1243
1423
4123
4132
1432
1342
1324
3124
3142
3412
4312
4321
3421
3241
3214
2314
2341
2431
4231
4213
2413
2143
2134
```

2.3.4 Observation Book Pictures:



PAGE NO :
DATE : 03/07/2023

Experiment - 3

Implement Johnson Trotter algorithm to generate permutations.

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

#define MAX_N 10

void swap (int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void printPermutation (int permutation[],
                       int direction[], int n)
{
    for (int i=0; i<n; i++)
    {
        printf("%d", permutation[i]);
    }
    printf("\n");
}

void generatePermutations (int n)
{
    int permutation[MAX_N];
    int direction[MAX_N];
    bool mobile[MAX_N];

    for (int i=0; i<n; i++)
    {
        permutation[i] = i+1;
        direction[i] = -1;
        mobile[i] = true;
    }
```

→

```
printPermutation (permutation, direction, n);

int mobileElement, mobileIndex, temp;

while (true)
{ mobileElement = -1;
  mobileIndex = -1;

  for (int i = 0; i < n; i++)
  { if (direction[i] == -1 && i > 0 &&
      permutation[i] > permutation[i-1] &&
      mobile[i])
    { if (mobileElement == -1 || permutation[i] >
                                 mobileElement)
      { mobileElement = permutation[i];
        mobileIndex = i;
      }
    }
  if (direction[i] == 1 && i < (n-1) &&
      permutation[i] > permutation[i+1] && mobile[i])
  { if (mobileElement == -1 || permutation[i] > mobileElement
    { mobileElement = permutation[i];
      mobileIndex = i;
    }
  }
}

if (mobileIndex == -1)
  { break;
  }
```

```
        if (direction[mobileIndex] == -1)
        {   swap(&permutation[mobileIndex],
                &permutation[mobileIndex - 1]);
            swap(&direction[mobileIndex],
                &direction[mobileIndex - 1]);
        }
        else
        {   swap(&permutation[mobileIndex],
                &permutation[mobileIndex + 1]);
            swap(&direction[mobileIndex],
                &direction[mobileIndex + 1]);
        }

        for(int i = 0; i < n; i++)
        {   if(permutation[i] > mobileElement)
            {   direction[i] *= -1;
            }
        }
        printPermutation(permutation, direction, n);
    }
}

int main()
{   int n;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    if (n < 1 || n > MAX_N)
    {   printf("Invalid Input!\n");
        return 0;
    }
    generatePermutations(n);
    return 0;
}
```

O/P

1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

PAGE NO :
DATE :

Output:

Enter the value of n : 4

1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4

2.4 Experiment - 4

2.4.1 Question:

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.

2.4.2 Code:

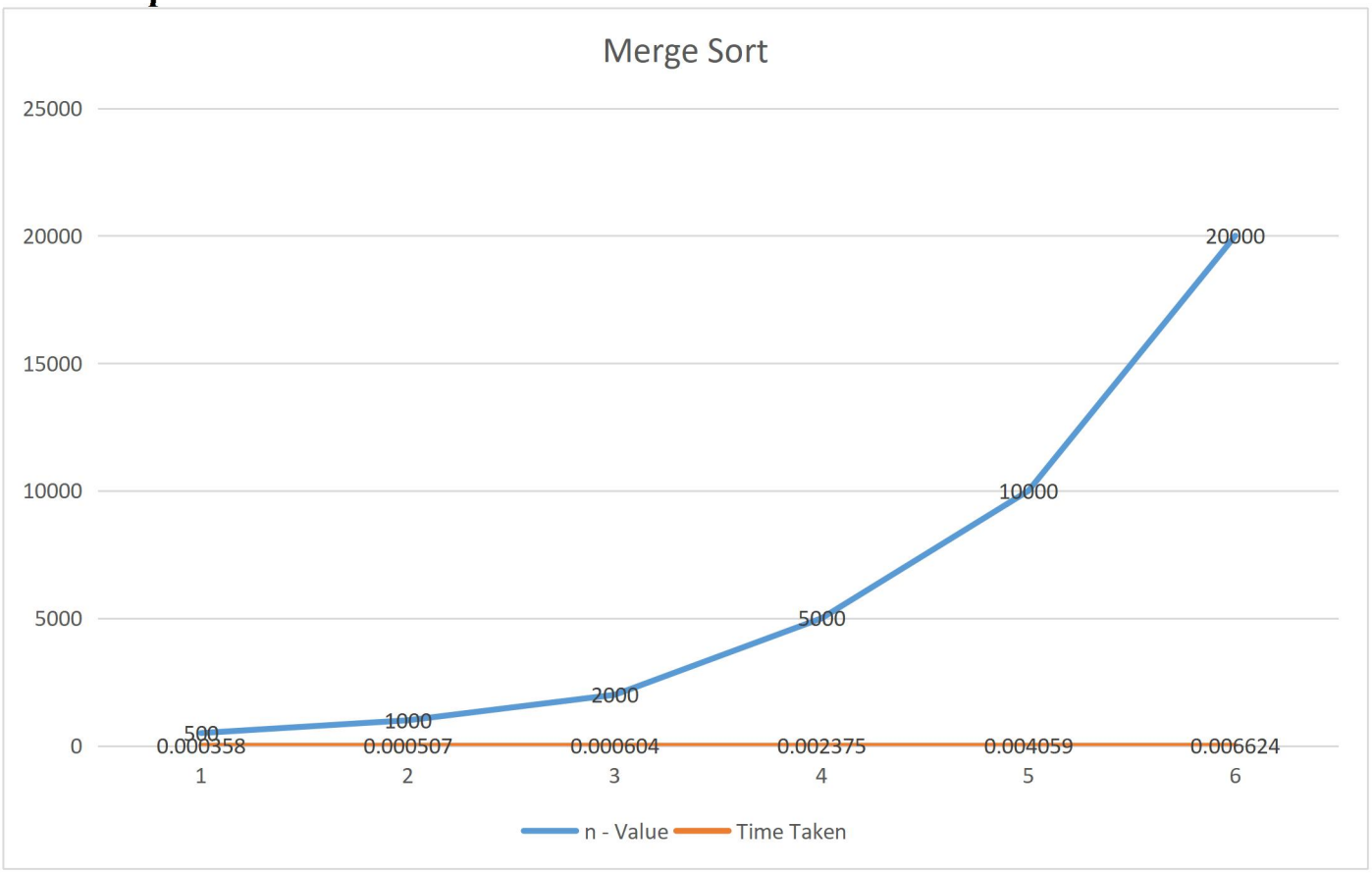
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);

int main()
{
   int a[50000],n,i;
   clock_t start_t, end_t;
   double total_t;
   srand(time(NULL));

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

   printf("Enter array elements:");
   for(i=0;i<n;i++)
   a[i]=rand()%10000;
   start_t = clock();
   printf("Starting of the program, start_t = %ld\n", start_t);
   mergesort(a,0,n-1);
   end_t = clock();

   printf("End of the program, end_t = %ld\n", end_t);
   total_t = (double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("Total time taken by CPU: %f\n", total_t  );

   printf("\nSorted array is :");
   for(i=0;i<n;i++)
   printf("%d ",a[i]);

   return 0;
}
```

```
void mergesort(int a[],int i,int j)
{
   int mid;
   if(i<j)
   {
      mid=(i+j)/2;
      mergesort(a,i,mid);
      mergesort(a,mid+1,j);
      merge(a,i,mid,mid+1,j);
   }
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
   int temp[50000];
   int i,j,k;

   i=i1;
   j=i2;
   k=0;

   while(i<=j1 && j<=j2)
   {
      if(a[i]<a[j])
      temp[k++]=a[i++];

      else
      temp[k++]=a[j++];
   }

   while(i<=j1)
   temp[k++]=a[i++];

   while(j<=j2)
   temp[k++]=a[j++];

   for(i=i1,j=0;i<=j2;i++,j++)
   a[i]=temp[j];
}
```

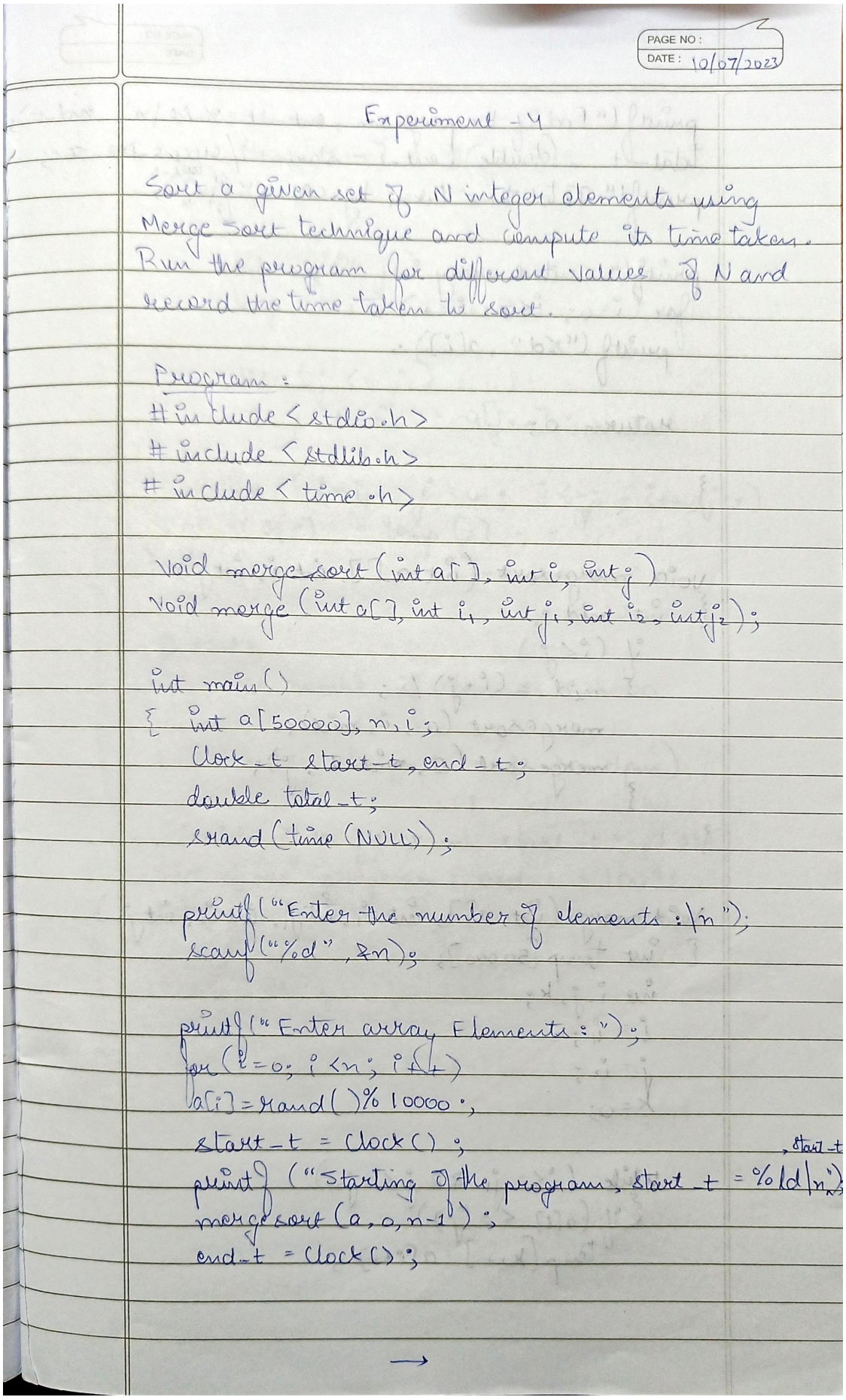
2.4.3 Output:

```
Enter the number of elements: 50

Enter array elements: (By Random)
Starting of the program, start_t = 1166
End of the program, end_t = 1387
Total time taken by CPU: 0.000221

Sorted array is:
273 304 768 1028 1368 1650 1652 2035 2230 4247 4313 4402 4470 460
3 4990 5026 5106 5133 5542 5862 5919 5953 6070 6185 6470 6688 669
6 6715 7300 7355 7515 7681 7836 7857 7887 7935 7971 8462 8750 907
9 9166 9214 9353 9384 9415 9777 9799 9820 9920 9949
```

2.4.4 Graph:



2.4.5 Observation Book Pictures:



PAGE NO :
DATE : 10/07/2023

Experiment - 4

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.

Program :

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

void merge_sort(int a[], int i, int j);
void merge(int a[], int i1, int j1, int i2, int j2);

int main()
{ int a[50000], n, i;
  clock_t start_t, end_t;
  double total_t;
  srand(time(NULL));

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

  printf("Enter array Elements :");
  for(i=0; i<n; i++)
  a[i] = rand()%10000;
  start_t = clock();
  printf("Starting of the program, start_t = %ld\n", start_t);
  merge_sort(a, 0, n-1);
  end_t = clock();
```

→

```
printf("End of the program, end_t = %ld\n", end_t);
total_t = (double)(end_t - start_t)/CLOCKS_PER_SEC;
printf("Total time taken by CPU: %f\n", total_t);

printf("Sorted array is: ");
for(i=0; i<n; i++)
  printf("%d", a[i]);

return 0;
}

void mergesort(int a[], int i, int j)
{ int mid;
  if(i<j)
  { mid = (i+j)/2;
    mergesort(a, i, mid);
    mergesort(a, mid+1, j);
  }
}

Void merge(int a[], int i1, int j1, int i2, int j2)
{ int temp[50000];
  int i, j, k;
  i = i1;
  j = i2;
  k = 0;

  while(i<=j1 && j<=j2)
  { if(a[i] < a[j])
      temp[k++] = a[i++];
```

```
        else
        temp[k++] = a[j++];
    }

    while (i<=j1)
    temp[k++] = a[i++];

    while (j<=j2)
    temp[k++] = a[j++];

    for (i=i1, j=0; i<=j2; i++, j++)
    a[i] = temp[j];
}
```

Output:

Enter the number of elements = 25000

Enter array elements : (By Random)

Starting of the program, start-t = 1331

End of the program, end-t = 10091

Total time taken by CPU = 0.008760

Sorted array :

2.5 Experiment - 5

2.5.1 Question:

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

2.5.2 Code:

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

void swap (int *a, int *b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

int partition (int arr[], int low, int high)
{
   int pivot = arr[high];
   int i = (low - 1);
   for (int j = low; j <= high - 1; j++)
   {
      if (arr[j] <= pivot)
      {
         i++;
         swap (&arr[i], &arr[j]);
      }
   }
   swap (&arr[i + 1], &arr[high]);
   return (i + 1);
}

void quickSort (int arr[], int low, int high)
{
   if (low < high)
   {
      int pi = partition (arr, low, high);
      quickSort (arr, low, pi - 1);
      quickSort (arr, pi + 1, high);
   }
}

int main ()
{
```

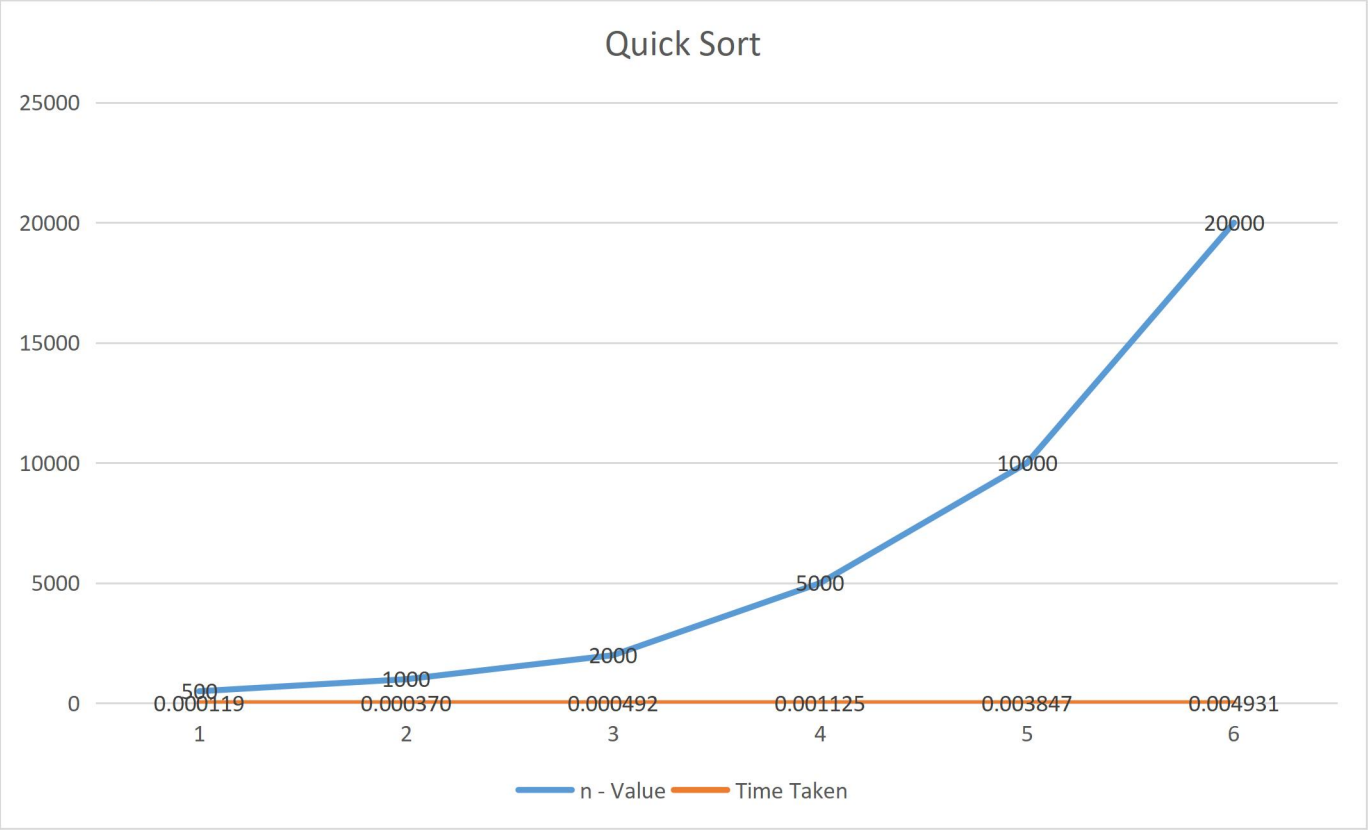
```
    int n;
    clock_t start_t, end_t;
    double total_t;

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

    printf("Enter the maximum value of the elements: ");
    int max;

    scanf("%d", &max);
    for (int i = 0; i < n; i++)
    {
       arr[i] = rand() % max;
    }

    printf("\nUnsorted array: \n");
    for (int i = 0; i < n; i++)
    {
       printf("%d ", arr[i]);
    }
    start_t = clock();
       printf("\n\nStarting of the program: %ld\n", start_t);

    quickSort (arr, 0, n - 1);

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

    end_t = clock();
       printf("\n\nEnd of the program: %ld\n", end_t);

    total_t = (double)(end_t - start_t)/CLOCKS_PER_SEC;
       printf("\n\nTotal time taken by CPU: %f\n", total_t);

    return 0;
}
```

2.5.3 Output:

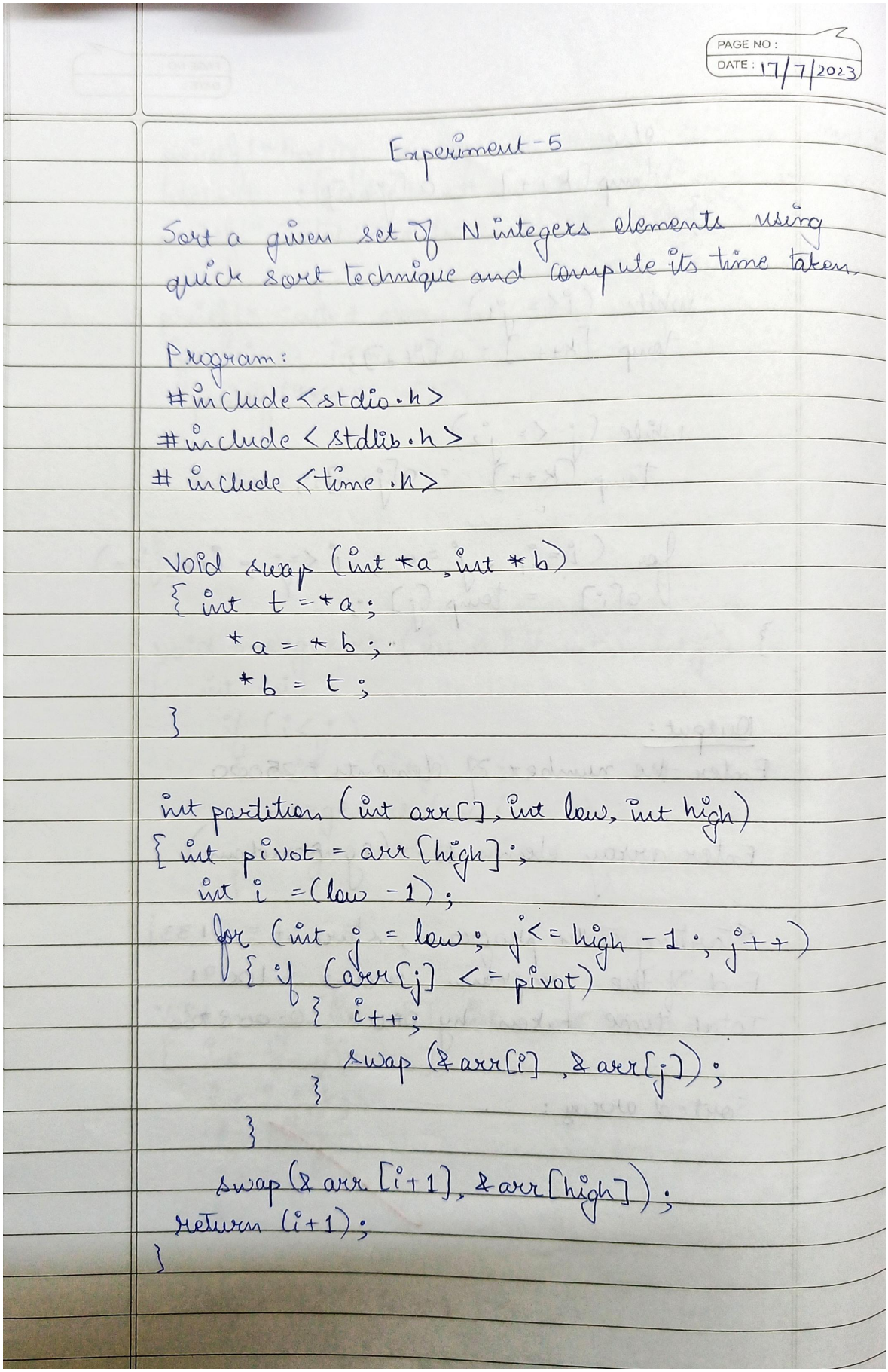
```
Enter the number of elements: 1000
Enter the maximum value of the elements: 1000

Unsorted array:
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 491 995 942 827 436 391 604 902 153 292 382 4
21 716 718 895 447 726 771 538 869 912 667 299 35 894 703 811 322 333 673 664 141 711 253 868 547 644 66
2 757 37 859 723 741 529 778 316 35 190 842 288 106 40 942 264 648 446 805 890 729 370 350 6 101 393 548
 629 623 84 954 756 840 966 376 931 308 944 439 626 323 537 538 118 82 929 541 833 115 639 658 704 930 9
77 306 673 386 21 745 924 72 270 829 777 573 97 512 986 290 161 636 355 767 655 574 31 52 350 150 941 72
4 966 430 107 191 7 337 457 287 753 383 945 909 209 758 221 588 422 946 506 30 413 168 900 591 762 655 4
10 359 624 537 548 483 595 41 602 350 291 836 374 20 596 21 348 199 668 484 281 734 53 999 418 938 900 7
88 127 467 728 893 648 483 807 421 310 617 813 514 309 616 935 451 600 249 519 556 798 303 224 8 844 609
 989 702 195 485 93 343 523 587 314 503 448 200 458 618 580 796 798 281 589 798 9 157 472 622 538 292 38
 179 190 657 958 191 815 888 156 511 202 634 272 55 328 646 362 886 875 433 869 142 844 416 881 998 322
651 21 699 557 476 892 389 75 712 600 510 3 869 861 688 401 789 255 423 2 585 182 285 88 426 617 757 832
 932 169 154 721 189 976 329 368 692 425 555 434 549 441 512 145 60 718 753 139 423 279 996 687 529 549
437 866 949 193 195 297 416 286 105 488 282 455 734 114 701 316 671 786 263 313 355 185 53 912 808 832 9
45 313 756 321 558 646 982 481 144 196 222 129 161 535 450 173 466 44 659 292 439 253 24 154 510 745 649
 186 313 474 22 168 18 787 905 958 391 202 625 477 414 314 824 334 874 372 159 833 70 487 297 518 177 77
3 270 763 668 192 985 102 480 213 627 802 99 527 625 543 924 23 972 61 181 3 432 505 593 725 31 492 142
222 286 64 900 187 360 413 974 270 170 235 833 711 760 896 667 285 550 140 694 695 624 19 125 576 694 65
8 302 371 466 678 593 851 484 18 464 119 152 800 87 60 926 10 757 170 315 576 227 43 758 164 109 882 86
565 487 577 474 625 627 629 928 423 520 902 962 123 596 737 261 195 525 264 260 202 116 30 326 11 771 41
1 547 153 520 790 924 188 763 940 851 662 829 900 713 958 578 365 7 477 200 58 439 303 760 357 324 477 1
08 113 887 801 850 460 428 993 384 405 540 111 704 835 356 72 350 823 485 556 216 626 357 526 357 337 27
1 869 361 896 22 617 112 717 696 585 41 423 129 229 565 559 932 296 855 53 962 584 734 654 972 457 369 5
32 963 607 483 911 635 67 848 675 938 223 142 754 511 741 175 459 825 221 870 626 934 205 783 850 398 27
9 701 193 734 637 534 556 993 176 705 962 548 881 300 413 641 855 855 142 462 611 877 424 678 752 443 29
6 673 40 313 875 72 818 610 17 932 112 695 169 831 40 488 685 90 497 589 990 145 353 314 651 740 44 258
335 759 192 605 264 181 503 829 775 608 292 997 549 556 561 627 467 541 129 240 813 174 601 77 215 683 2
13 992 824 601 392 759 670 428 27 84 75 786 498 970 287 847 604 503 221 663 706 363 10 171 489 240 164 5
42 619 913 591 704 818 232 750 205 975 539 303 422 98 247 584 648 971 864 913 75 545 712 546 678 769 262
 519 985 289 944 865 540 245 508 318 870 601 323 132 472 152 87 570 763 901 103 423 527 600 969 15 565 2
8 543 347 88 943 637 409 463 49 681 588 342 608 60 221 758 954 888 146 690 949 843 430 620 748 67 536 78
3 35 226 185 38 853 629 224 748 923 359 257 766 944 955 318 726 411 25 355 1 549 496 584 515 964 342 75
913 142 196 948 72 426 606 173 429 404 705 626 812 375 93 565 36 736 141 814 994 256 652 936 838 482 355
 15 131 230 841 625 11 637 186 690 650 662 634 893 353 416 452 8 262 233 454 303 634 303 256 148 124 317
 213 109 28 200 80 318 858 50 155 361 264 903 676 643 909 902 561 489 948 282 653 674 220 402 923 831 36
9 878 259 8 619 971 3 945 781 504 392 685 313 698 589 722 938 37 410 461 234 508 961 959 493 515 269 937
 869 58 700 971 264 117 215 555 815 330 39 212 288 82 954 85 710 484 774 380 815 951 541 115 679 110 898
 73 788 977 132 956 689 113 8 941 790 723 363 28 184 778 200 71 885 974 71 333 867 153 295 168 825 676 6
29 650 598 309 693 686 80 116 249
```

```
29 650 598 309 693 686 80 116 249

Starting of the program: 6640



Sorted array:
1 2 3 3 3 6 7 7 8 8 8 8 9 10 10 11 11 15 15 17 18 18 19 20 21 21 21 22 22 23 24 25 27 28 28 28 30 30 31
31 35 35 35 36 37 37 38 38 39 40 40 40 41 41 41 43 44 44 49 50 52 53 53 53 55 58 58 60 60 60 61 64 67 67
 70 71 71 72 72 72 72 73 75 75 75 75 77 80 80 82 82 84 84 85 86 87 87 88 88 90 93 93 97 98 99 101 102 10
3 105 106 107 108 109 109 110 111 112 112 113 113 114 115 115 116 116 117 118 119 123 124 125 127 129 12
9 129 131 132 132 139 140 141 141 142 142 142 142 142 144 145 145 145 146 148 150 152 152 153 153 153 15
4 154 155 156 157 159 161 161 164 164 168 168 168 169 169 169 170 170 171 173 173 174 175 176 177 179 18
1 181 182 184 185 185 186 186 187 188 189 190 190 191 191 192 192 193 193 195 195 195 196 196 199 200 20
0 200 200 202 202 202 205 205 209 212 213 213 213 215 215 216 220 221 221 221 221 222 222 223 224 224 22
6 227 229 230 232 233 234 235 240 240 245 247 249 249 253 253 255 256 256 257 258 259 260 261 262 262 26
3 264 264 264 264 264 269 270 270 270 271 272 279 279 281 281 281 282 282 285 285 286 286 287 287 288 28
8 289 290 291 292 292 292 292 295 296 296 297 297 299 300 302 303 303 303 303 303 306 308 309 309 310 31
3 313 313 313 313 314 314 314 315 316 316 317 318 318 318 321 322 322 323 323 324 326 328 329 330 333 33
3 334 334 335 337 337 342 342 343 347 348 350 350 350 350 353 353 355 355 355 355 356 357 357 357 358 35
9 359 360 361 361 362 363 363 365 368 369 369 370 371 372 374 375 376 380 382 383 384 386 389 391 391 39
2 392 393 398 401 402 404 405 409 410 410 411 411 413 413 413 414 416 416 416 418 421 421 422 422 423 42
3 423 423 423 424 425 426 426 428 428 429 430 430 432 433 434 436 437 439 439 439 441 443 446 447 448 45
0 451 452 454 455 457 457 458 459 460 461 462 463 464 464 466 466 467 467 467 472 472 474 474 476 477 47
7 477 478 480 481 482 483 483 483 484 484 484 485 485 487 487 488 488 489 489 491 492 493 496 497 498 50
0 503 503 503 504 505 506 508 508 510 510 511 511 512 512 514 515 515 518 519 519 520 520 523 525 526 52
7 527 529 529 532 534 535 536 537 537 538 538 538 539 540 540 541 541 541 542 543 543 545 546 547 547 54
8 548 548 549 549 549 549 550 555 555 556 556 556 556 557 558 559 561 561 565 565 565 565 570 573 574 57
6 576 577 578 580 584 584 584 585 585 587 588 588 589 589 589 591 591 593 593 595 596 596 598 600 600 60
0 601 601 601 602 604 604 605 606 607 608 608 609 610 611 616 617 617 617 618 619 619 620 622 623 624 62
4 625 625 625 625 626 626 626 626 627 627 627 629 629 629 629 634 634 634 635 636 637 637 637 639 641 64
3 644 646 646 648 648 648 649 650 650 651 651 652 653 654 655 655 657 658 658 659 662 662 662 663 664 66
7 667 668 668 670 671 673 673 673 674 675 676 676 678 678 678 679 681 683 685 685 686 687 688 689 690 69
0 692 693 694 694 695 695 696 698 699 700 701 701 702 703 704 704 704 705 705 705 706 710 711 711 712 71
2 713 716 717 718 718 721 722 723 723 724 724 725 726 726 728 729 734 734 734 734 736 737 740 741 741 74
5 745 748 748 750 752 753 753 754 756 756 757 757 757 758 758 758 759 759 760 760 762 763 763 763 766 76
7 769 771 771 773 774 775 777 778 778 781 783 783 786 786 787 788 788 789 790 790 796 798 798 798 800 80
1 802 805 807 808 811 812 813 813 814 815 815 815 818 818 823 824 824 825 825 827 827 829 829 829 831 83
1 832 832 833 833 833 835 836 838 840 841 842 843 844 844 847 848 850 850 851 851 853 855 855 855 858 85
9 861 864 865 866 867 868 869 869 869 869 869 870 870 874 875 875 877 878 881 881 882 885 886 887 888 88
8 890 892 893 893 894 895 896 896 898 900 900 900 900 901 902 902 902 903 905 909 909 911 912 912 913 91
3 913 923 923 924 924 924 926 928 929 930 931 932 932 932 934 935 936 937 938 938 938 940 941 941 942 94
2 943 944 944 944 945 945 945 946 948 948 949 949 951 954 954 954 955 956 958 958 958 959 961 961 962 96
2 962 962 963 964 966 966 969 970 971 971 971 972 972 974 974 975 976 977 977 982 985 985 986 989 990 99
2 993 993 994 995 996 997 998 999

End of the program: 6671


Total time taken by CPU: 0.031000
```

2.5.4 Graph:


Quick Sort
25000
20000
15000
10000
5000
0
500
1000
2000
5000
10000
20000
0.000119
0.000370
0.000492
0.001125
0.003847
0.004931
1
2
3
4
5
6
n - Value
Time Taken

2.5.5 Observation Book Pictures:



PAGE NO :
DATE : 17/7/2023

Experiment-5

Sort a given set of N integers elements using quick sort technique and compute its time taken.

Program:

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

void swap(int *a, int *b)
{ int t = *a;
  *a = *b;
  *b = t;
}

int partition(int arr[], int low, int high)
{ int pivot = arr[high];
  int i = (low - 1);
  for (int j = low; j <= high - 1; j++)
  { if (arr[j] <= pivot)
    { i++;
      swap(&arr[i], &arr[j]);
    }
  }
  swap(&arr[i+1], &arr[high]);
  return (i+1);
}
```

```
void quickSort (int arr [], int low, int high)
{ if (low < high)
  { int pi = partition (arr, low, high);
    quickSort (arr, low, pi-1);
    quickSort (arr, pi+1, high);
  }
}

int main ()
{ int n;
  double total_t;
  clock_t start_t, end_t;

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

  printf ("Enter the max value of the elements : ");
  int max;

  scanf ("%d", &max);
  for (int i=0; i<n; i++)
  { arr [i] = rand % max;
  }
  printf ("\n Unsorted array : \n");
  for (int i=0; i<n; i++)
  { printf ("%d", arr[i]);

  start_t = clock();
    printf ("\n\n Starting of the program : %ld\n",
            start_t);

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

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

end_t = clock();
printf("\n\nEnd of the program: %ld\n", end_t);

total_t = (double)(end_t - start_t)/CLOCKS_PER_SEC;
printf("\n\n Total time taken by CPU: %f\n", total_t);

return 0;
}
```

Output:

Enter the number of elements = 10

Enter the max value of the elements = 10

Unsorted array:

1 7 4 0 9 4 8 8 2 4

Starting of the program: 3093

Sorted array:

0 1 2 4 4 4 7 8 8 9

End of the program: 3093.

Total time taken by CPU: 0.000000.

2.6 Experiment - 6

2.6.1 Question:

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

2.6.2 Code:

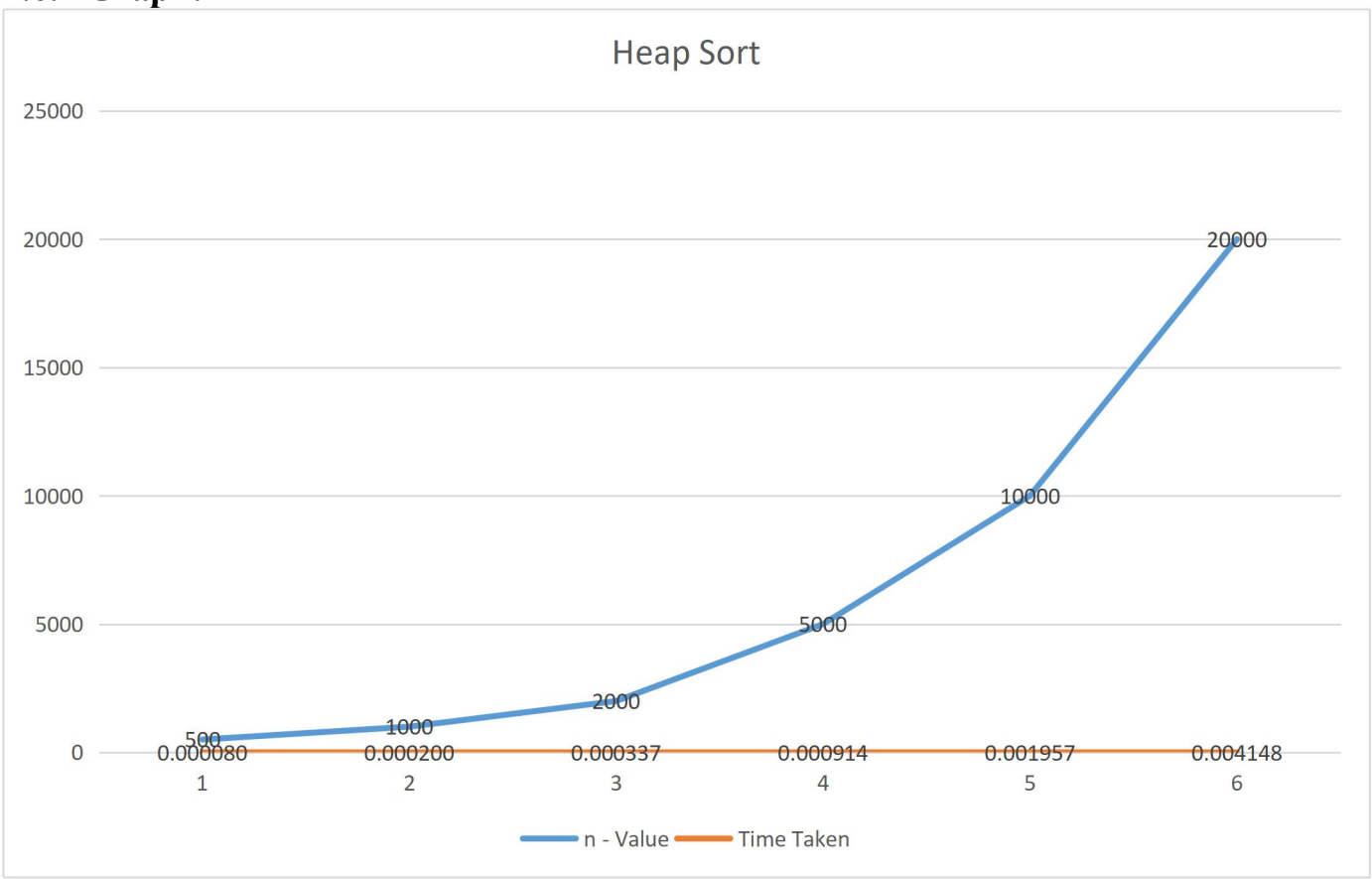
```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

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 n;
```

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

    int arr[n];

    // Generate and fill array with random numbers
    srand(time(NULL)); // Seed the random number generator
    printf("\n\nRandomly generated elements:\n");
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 1000; // Generate random numbers between 0 and 999

    printf("Original array: \n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    clock_t start_time = clock();  // Record the start time

    heapSort(arr, n);

    clock_t end_time = clock();    // Record the end time

    printf("\n\nSorted array: \n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);

    double time_taken = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("\n\nTime taken: %f seconds\n", time_taken);

    return 0;
}
```

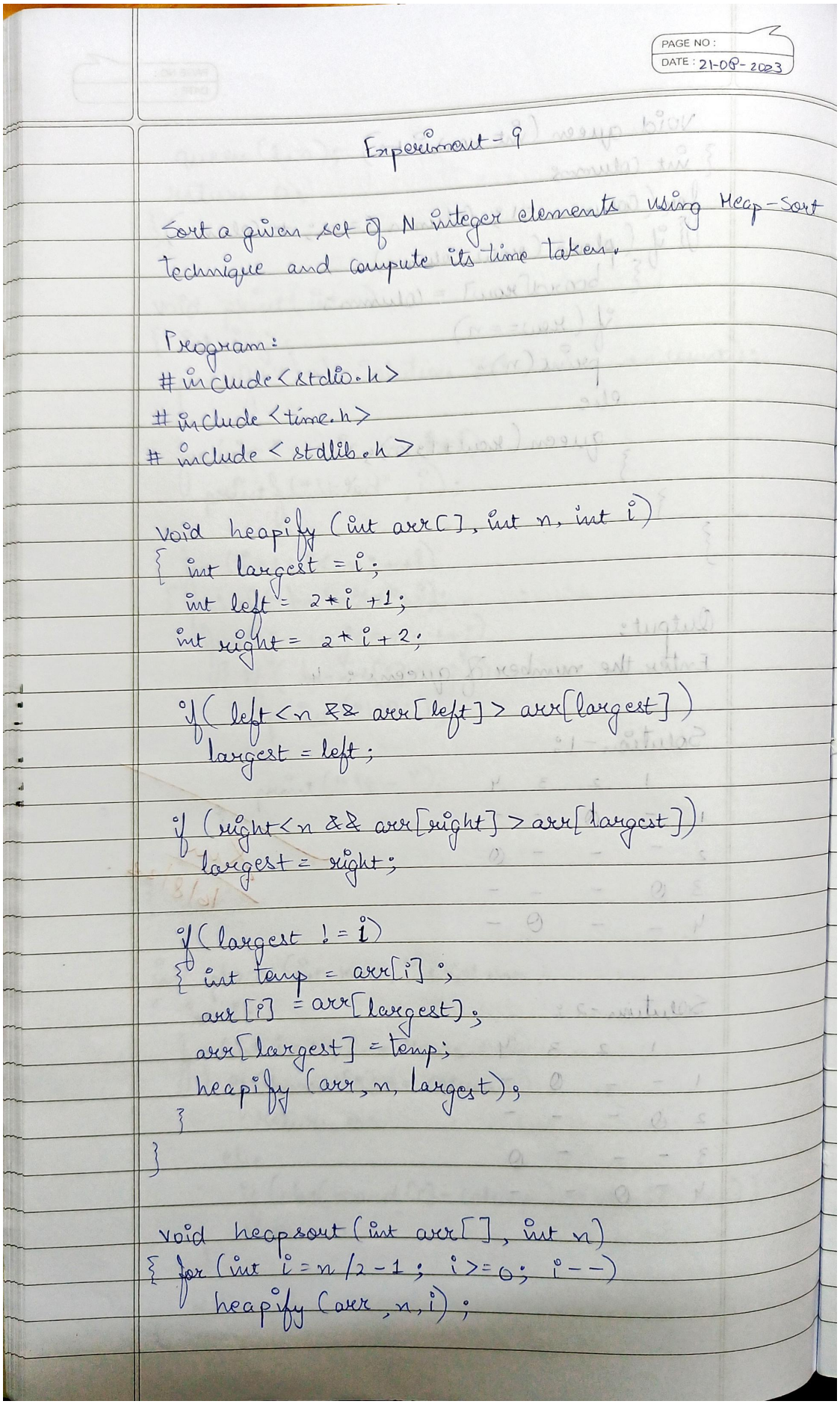
2.6.3 Output:

```
Enter the number of elements: 50

Randomly generated elements:
Original array:
887 569 555 294 826 235 206 460 222 303 286 157 89 883 791 362 451 224 936 1
61 690 18 930 495 497 8 535 281 866 436 505 753 357 60 399 184 648 958 644 2
22 261 283 731 350 518 522 712 970 98 0

Sorted array:
0 8 18 60 89 98 157 161 184 206 222 222 224 235 261 281 283 286 294 303 350
357 362 399 436 451 460 495 497 505 518 522 535 555 569 644 648 690 712 731
753 791 826 866 883 887 930 936 958 970

Time taken: 0.000007 seconds
```

2.6.4 Graph:



2.6.5 Observation Book Pictures:



PAGE NO :
DATE : 21-09-2023

Experiment - 9

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

Program:

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

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

  int arr[n];

  srand (time[NULL]);
  printf("\n\n Randomly generated elements : \n");
  for (int i = 0; i<n; i++)
    arr[i] = rand() % 1000;

  printf("original array : \n");
  for (int i=0; i<n; i++)
    printf("%d", arr[i]);

  clock_t start_time = clock();

  heapSort(arr, n);

  clock_t end_time = clock();
```

→

```
printf("\n\n Sorted array: \n");
for (int i = 0; i < n; i++)
    printf("%d", arr[i]);

double time_taken = (double)(end_time - start_time) /
                    CLOCKS_PER_SEC;

printf("\n\n Time Taken: %f seconds\n", time_taken);

return 0;

}
```

Output:

Enter the number of elements: 20

Randomly generated elements:

Original array:

307 592 675 816 440 301 524 733 505 899
772 822 943 400 913 152 138 620 350 6

Sorted array:

6 138 152 301 307 350 400 440 505 524 591
620 675 733 772 816 822 899 913 943

Time taken: 0.000003

2.7 Experiment - 7

2.7.1 Question:

Implement 0/1 Knapsack problem using dynamic programming.

2.7.2 Code:

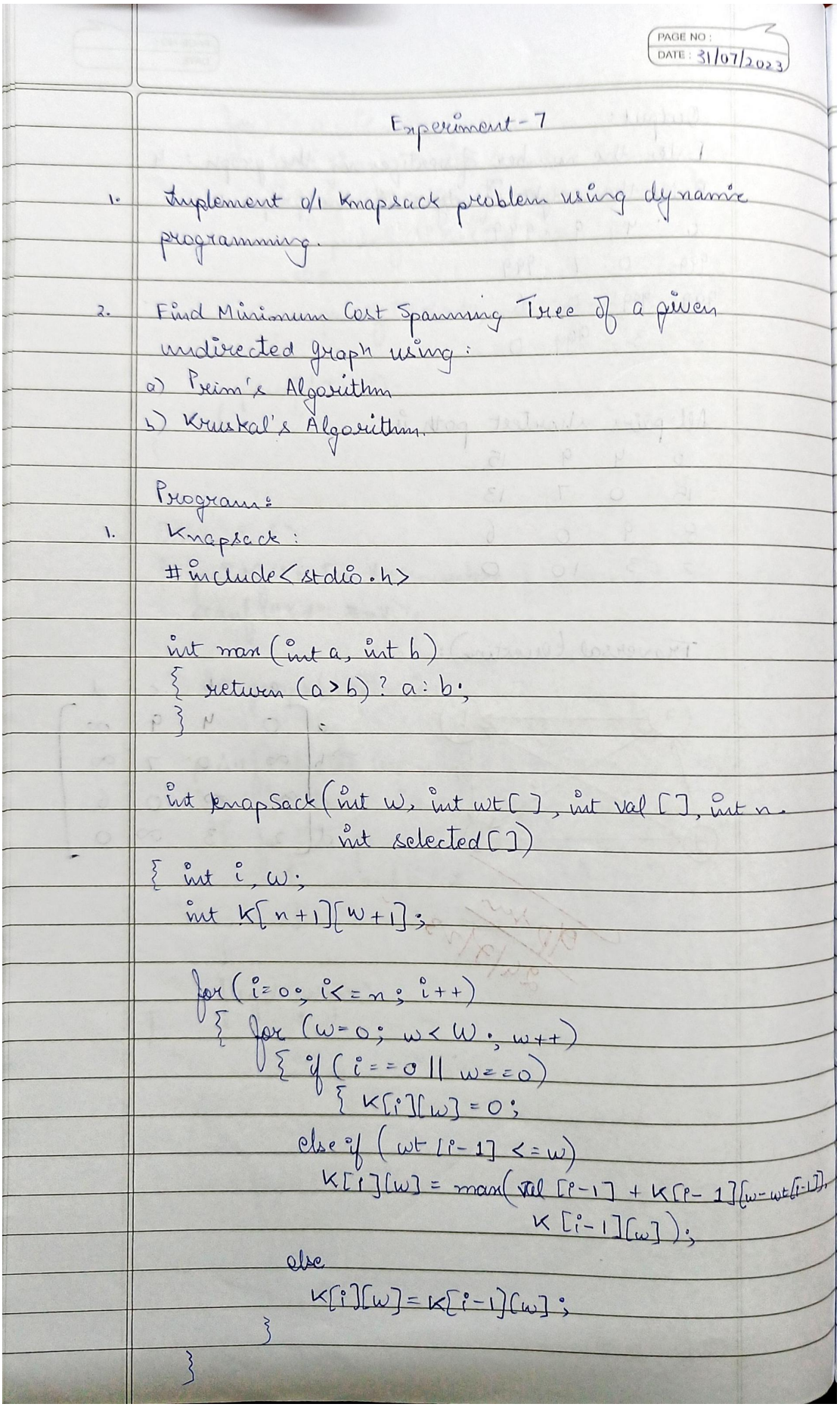
```
#include <stdio.h>

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

int knapSack(int W, int wt[], int val[], int n, int selected[])
{
   int i, w;
   int K[n + 1][W + 1];

   for (i = 0; i <= n; i++)
   {
      for (w = 0; w <= W; w++)
      {
         if (i == 0 || w == 0)
            K[i][w] = 0;
         else if (wt[i - 1] <= w)
            K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
         else
            K[i][w] = K[i - 1][w];
      }
   }

   int res = K[n][W];
   w = W;
   for (i = n; i > 0 && res > 0; i--)
   {
      if (res == K[i - 1][w])
         continue;
      else {
         selected[i - 1] = 1;
         res = res - val[i - 1];
         w = w - wt[i - 1];
      }
   }

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

```
int main()
{
   int n, W, i;
   printf("Enter the number of items: ");
   scanf("%d", &n);

   int val[n], wt[n], selected[n];
   printf("Enter the values of the items: ");
   for (i = 0; i < n; i++)
      scanf("%d", &val[i]);

   printf("Enter the weights of the items: ");
   for (i = 0; i < n; i++)
      scanf("%d", &wt[i]);

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

   int max_profit = knapSack(W, wt, val, n, selected);
   printf("The maximum profit is %d\n", max_profit);

   printf("The objects selected for the optimal solution are: ");
   for (i = 0; i < n; i++) {
      if (selected[i]==1)
         printf("%d ", i + 1);
   }
   return 0;
}
```

2.7.3 Output:

```
Enter the number of items: 4
Enter the values of the items: 3 4 5 6
Enter the weights of the items: 2 3 4 5
Enter the capacity of the knapsack: 5
The maximum profit is 7
The objects selected for the optimal solution are: 1 2
```

2.7.4 Observation Book Pictures:



PAGE NO :
DATE : 31/07/2023

Experiment - 7

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

2. Find Minimum Cost Spanning Tree of a given undirected graph using:
   a) Prim's Algorithm
   b) Kruskal's Algorithm.

Program:

1. Knapsack:

```
#include<stdio.h>

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

int knapSack(int W, int wt[], int val[], int n,
             int selected[])
{ int i, w;
  int K[n+1][W+1];

  for(i=0; i<=n; i++)
  { for(w=0; w<W; w++)
    { if(i==0 || w==0)
      { K[i][w]=0;
      else if(wt[i-1]<=w)
        K[i][w]=max(val[i-1]+K[i-1][w-wt[i-1]],
                    K[i-1][w]);
      else
        K[i][w]=K[i-1][w];
    }
  }
```

```
    int res = K[n][W];
    w = W;
    for (i = n; i > 0 && res > 0; i--)
    { if (res == K[i-1][w])
        continue;
      else
      { selected[i-1] = 1;
        res = res - val[i-1];
        w = w - wt[i-1];
      }
    }
    return K[n][W];
}
int main()
{ int n, W, i;
  printf("Enter the number of items : ");
  scanf("%d", &n);

  int val[n], wt[n], selected[n];
  printf("Enter the values of items : ");
  for(i = 0; i < n; i++)
      scanf("%d", &val[i]);

  printf("Enter the weights of the items: ");
  for(i = 0; i < n; i++)
      scanf("%d", &wt[i]);

  printf("Enter the Knapsack capacity : ");
  scanf("%d", &W);

  int max_profit = knapSack(W, wt, val, n, selected);
  printf("The maximum profit is %d\n", max_profit);
```

→

```
printf("The objects selected for the optimal
        solution are: ");
for (i=0; i<n; i++)
  { if (selected[i] == 1)
      printf("%d", i+1);
  }
return 0;
}
```

Output:

Enter the number of items: 4
Enter the values of items: 3 4 5 6
Enter the weights of items: 2 3 4 5
Enter the knapsack capacity: 5

The maximum profit is: 7

The objects selected for the optimal solution are:
1 2

Item	weight	Value
1	2	3
2	3	4
3	4	5
4	5	6

2.8 Experiment - 8

2.8.1 Question:

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

2.8.2 Code:

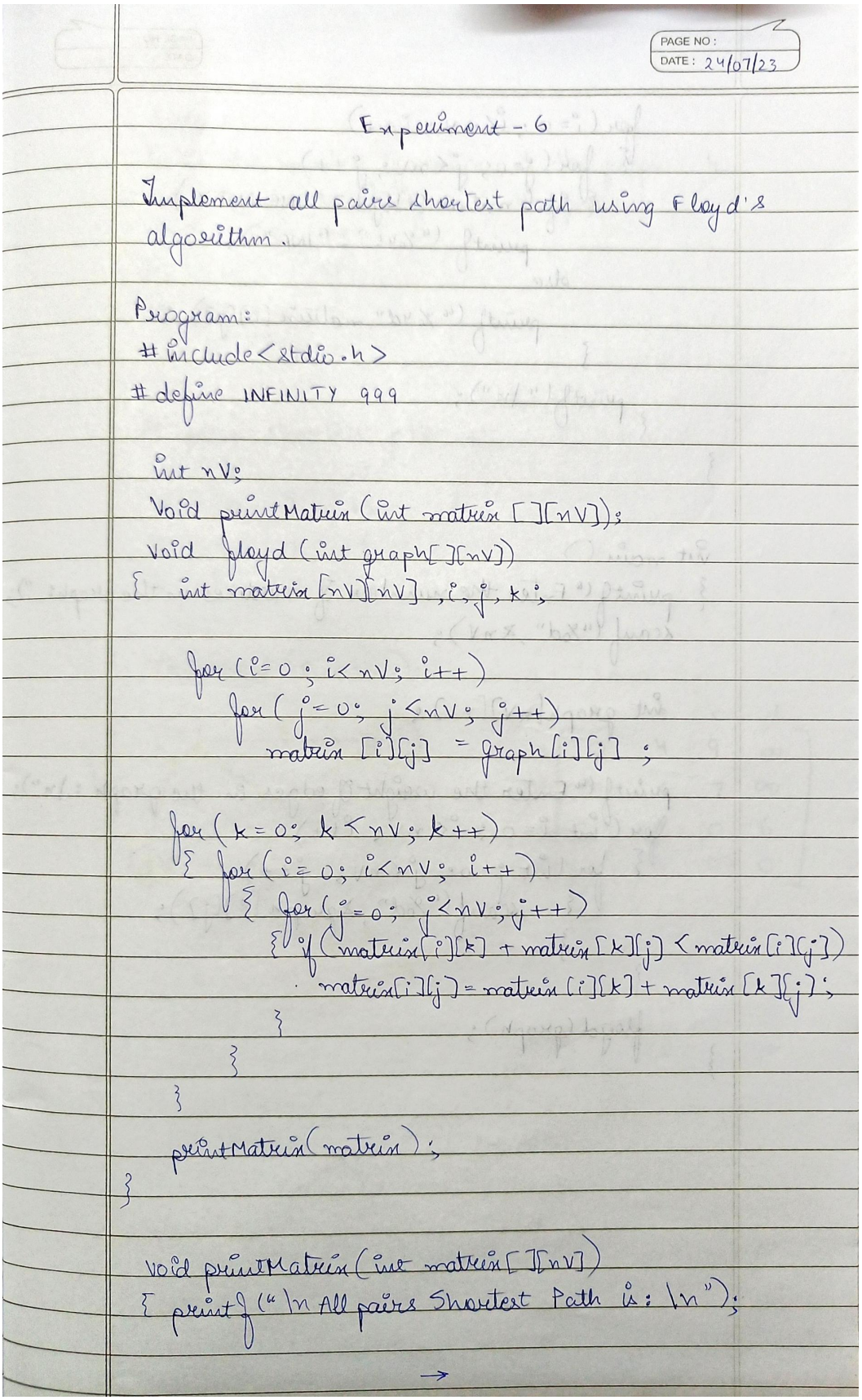
```
#include <stdio.h>

#define INFINITY 999

int nV;
void printMatrix(int matrix[][nV]);

void floyd(int graph[][nV])
{
   int matrix[nV][nV], i, j, k;

   for (i = 0; i < nV; i++)
      for (j = 0; j < nV; j++)
         matrix[i][j] = graph[i][j];

   for (k = 0; k < nV; k++)
   {
      for (i = 0; i < nV; i++)
      {
         for (j = 0; j < nV; j++)
         {
            if (matrix[i][k] + matrix[k][j] < matrix[i][j])
            matrix[i][j] = matrix[i][k] + matrix[k][j];
         }
      }
   }
   printMatrix(matrix);
}

void printMatrix(int matrix[][nV])
{
   printf("\nAll Pairs Shortest Path is :\n");
   for (int i = 0; i < nV; i++)
   {
      for (int j = 0; j < nV; j++)
      {
         if (matrix[i][j] == INFINITY)
            printf("%4s", "INF");
         else
```

```
            printf("%4d", matrix[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    printf("Enter the number of vertices in the graph: ");
    scanf("%d", &nV);

    int graph[nV][nV];
    printf("Enter the weight of edges in the graph:\n");
    for (int i = 0; i < nV; i++)
    {
        for (int j = 0; j < nV; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    floyd(graph);
}
```

2.8.3 Output:

```
Enter the number of vertices in the graph: 4
Enter the weight of edges in the graph:
0 4 9 999
999 0 7 999
999 999 0 6
2 3 999 0

All Pairs Shortest Path is :
   0   4   9  15
  15   0   7  13
   8   9   0   6
   2   3  10   0
```

2.8.4 Observation Book Pictures:



PAGE NO :
DATE : 24/07/23

Experiment - 6

Implement all pairs shortest path using Floyd's algorithm.

Program:

```
#include<stdio.h>
#define INFINITY 999

int nV;
Void printMatrix(int matrix[][nV]);
Void floyd(int graph[][nV])
{ int matrix[nV][nV], i, j, k;

    for(i=0; i<nV; i++)
        for(j=0; j<nV; j++)
            matrix[i][j] = graph[i][j];

    for(k=0; k<nV; k++)
    { for(i=0; i<nV; i++)
        { for(j=0; j<nV; j++)
            { if(matrix[i][k] + matrix[k][j] < matrix[i][j])
                matrix[i][j] = matrix[i][k] + matrix[k][j];
            }
        }
    }

    printMatrix(matrix);
}

void printMatrix(int matrix[][nV])
{ printf("\n All pairs Shortest Path is: \n");
```

→

```
    for (i=0; i<nV; i++)
    { for (j=0; j<nV; j++)
      { if (matrix[i][j] == INFINITY)
          printf ("%4s", "INF");
        else
          printf ("%4d", matrix[i][j]);
      }
      printf("\n");
    }
}

int main()
{ printf("Enter the number of vertices in the graph: ");
  scanf("%d", &nV);

  int graph[nV][nV];

  printf("Enter the weight of edges in the graph :\n");
  for(int i=0; i<nV; i++)
  { for (int j=0; j<nV; j++)
    { scanf("%d", &graph[i][j]);
    }
  }
  floyd(graph);
}
```

DATE :

Output:

Enter the number of vertices in the graph : 4

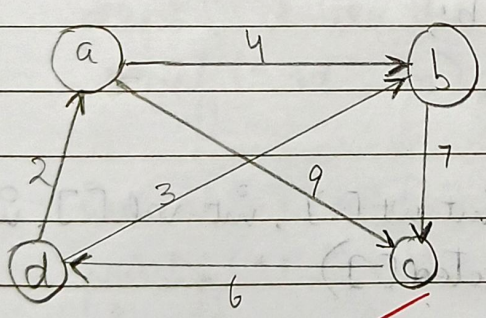
Enter the weight of edges in the graph :

```
  0   4   9  999
999   0   7  999
999 999   0    6
  2   3 999    0
```

All pairs shortest path is :

```
  0   4   9   15
 15   0   7   13
  8   9   0    6
  2   3  10    0
```

Traversal (Question):



	a	b	c	d
a	0	4	9	∞
⇒ b	∞	0	7	∞
c	∞	∞	0	6
d	2	3	∞	0

dp sen
24/7/23

2.9 Experiment - 9

2.9.1 Question:

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

2.9.2 Code:

(a) Prim's Algorithm:

```
#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()
{
   printf("Prim's algorithm:\n");

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

        printf("\nEnter the adjacency matrix:\n");
        for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
      {
         scanf("%d",&cost[i][j]);
         if(cost[i][j]==0)
            cost[i][j]=999;
      }
        visited[1]=1;
        printf("\n");

   printf("\nThe edges of Minimum Cost Spanning Tree are:");
        while(ne < n)
        {
                for(i=1,min=999;i<=n;i++)
          for(j=1;j<=n;j++)
             if(cost[i][j]< min)
             if(visited[i]!=0)
             {
                min=cost[i][j];
                a=u=i;
                b=v=j;
             }
```

```
            if(visited[u]==0 || visited[v]==0)
            {
                printf("\nEdge %d:(%d,%d) Weight:%d",ne++,a,b,min);
                mincost+=min;
                visited[b]=1;
            }
            cost[a][b]=cost[b][a]=999;
        }
        printf("\n\nMinimun Cost=%d",mincost);
}
```

(b) Kruskal's Algorithm:

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[10][10], parent[9];

int find(int);
int uni(int, int);

void main()
{
 printf("Kruskal's algorithm:\n");

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

 printf("\nEnter the cost adjacency matrix:\n");
 for (i = 1; i <= n; i++)
 {
  for (j = 1; j <= n; j++)
  {
   scanf("%d", &cost[i][j]);
   if (cost[i][j] == 0)
    cost[i][j] = 999;
  }
 }

 printf("\nThe edges of Minimum Cost Spanning Tree are\n");
 while (ne < n)
 {
  for (i = 1, min = 999; i <= n; i++)
  {
   for (j = 1; j <= n; j++)
```

```
      {
       if (cost[i][j] < min)
       {
         min = cost[i][j];
         a = u = i;
         b = v = j;
       }
      }
    }

    u = find(u);
    v = find(v);

    if (uni(u, v))
    {
      printf("Edge %d:(%d,%d) Weight:%d\n", ne++, a, b, min);
      mincost += min;
    }

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

  printf("\nMinimum cost = %d\n", mincost);
}

int find(int i)
{
  while (parent[i])
    i = parent[i];
  return i;
}

int uni(int i, int j)
{
  if (i != j)
  {
    parent[j] = i;
    return 1;
  }

  return 0;
}
```

2.9.3 Output:

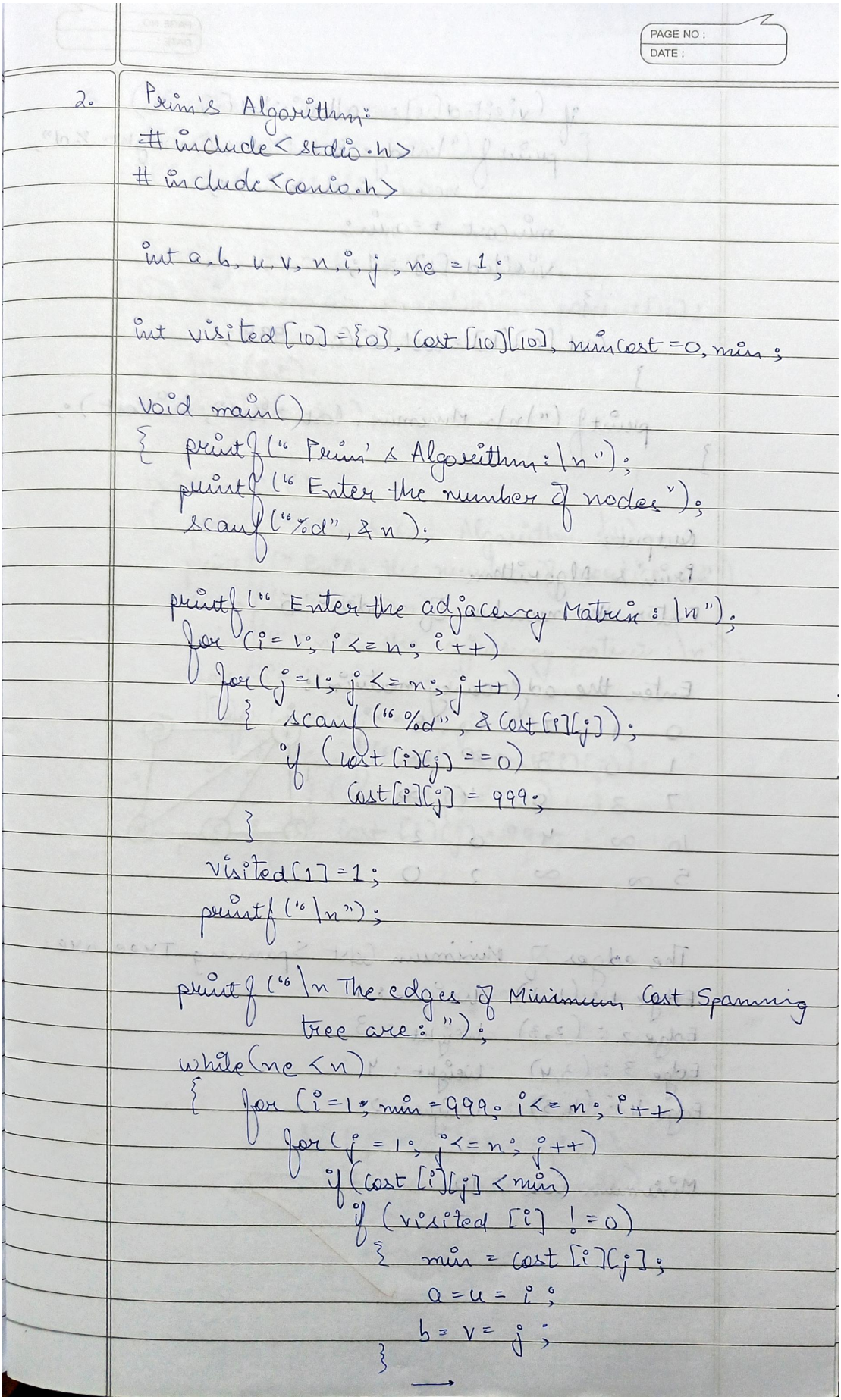
(a) Prim's Algorithm:

```
Prim's algorithm:
Enter the number of nodes:5

Enter the adjacency matrix:
0 1 7 10 5
1 0 3 999 999
7 3 0 4 999
10 999 4 0 2
5 999 999 2 0


The edges of Minimum Cost Spanning Tree are:
Edge 1:(1,2) Weight:1
Edge 2:(2,3) Weight:3
Edge 3:(3,4) Weight:4
Edge 4:(4,5) Weight:2

Minimun Cost=10
```

(b) Kruskal's Algorithm:

```
Kruskal's algorithm:
Enter the no. of vertices:
5

Enter the cost adjacency matrix:
0 1 7 10 5
1 0 3 999 999
7 3 0 4 999
10 999 4 0 2
5 999 999 2 0

The edges of Minimum Cost Spanning Tree are
Edge 1:(1,2) Weight:1
Edge 2:(4,5) Weight:2
Edge 3:(2,3) Weight:3
Edge 4:(3,4) Weight:4

Minimum cost = 10
```

2.9.4 Observation Book Pictures:



2. Prim's Algorithm:

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

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0}, cost[10][10], mincost=0,min;

void main()
{ printf("Prim's Algorithm:\n");
  printf("Enter the number of nodes");
  scanf("%d",&n);

 printf("Enter the adjacency Matrix:\n");
  for(i=1;i<=n;i++)
   for(j=1;j<=n;j++)
    { scanf("%d",&cost[i][j]);
      if(cost[i][j]==0)
        cost[i][j]=999;
    }
  visited[1]=1;
  printf("\n");

printf("\n The edges of Minimum Cost Spanning
          tree are:");
while(ne<n)
 { for(i=1;min=999;i<=n;i++)
    for(j=1;j<=n;j++)
      if(cost[i][j]<min)
        if(visited[i]!=0)
          { min=cost[i][j];
            a=u=i;
            b=v=j;
          }
```

```
        if (visited[u] == 0 || visited[v] == 0)
        {
            printf("\n Edge %d: (%d,%d) weight: %d",
                    ne++, a, b, min);
            mincost += min;
            visited[b] = 1;
        }
        cost[a][b] = cost[b][a] = 999;
    }
    printf("\n\n Minimum cost = %d", mincost);
}
```

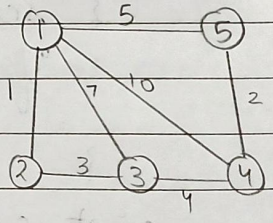
Output:

Prim's Algorithm:

Enter the number of nodes: 5

Enter the adjacency matrix:

0	1	7	10	5
1	0	3	∞	∞
7	3	0	4	∞
10	∞	4	0	2
5	∞	∞	2	0



The edges of Minimum Cost Spanning Tree are:

Edge 1: (1,2) Weight: 1

Edge 2: (2,3) Weight: 3

Edge 3: (3,4) Weight: 4

Edge 4: (4,5) Weight: 2

Minimum Cost = 10

PAGE NO :
DATE :

3. Kruskal's Algorithm:

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

int i, j, k, a, b, u, v, n, ne = 1;
int min, mincost = 0, cost[10][10], parent[10];

int find(int);
int uni(int, int);

void main()
{ printf("Kruskal's Algorithm :\n");
  printf("Enter the number of vertices :\n");
  scanf("%d", &n);
  printf("Enter the adjacency matrix :\n");
  for (i = 1; i <= n; i++)
  { for (j = 1; j <= n; j++)
    { scanf("%d", &cost[i][j]);
      if (cost[i][j] == 0)
        cost[i][j] = 999;
    }
  }

printf("\n The edges of the Minimum Cost
        Spanning Tree are :\n");
while (ne < n)
{ for (i = 1; min = 999; i <= n; i++)
  { for (j = 1; j <= n; j++)
    { if (cost[i][j] < min)
        a = u = i;
        b = v = j;
    }
  }
}
```

→

```
        u = find(u);
        v = find(v);

        if(uni(u,v))
        { printf("Edge %d:(%d,%d) weight:%d\n",
                 ne++,a,b,min);
          mincost += min;
          cost[a][b] = cost[b][a] = 999;
        }

        printf("\n Minimum cost: %d\n", mincost);
}

int find(int i)
{ while(parent[i])
      i = parent[i];
      return i;
}

int uni(int i, int j)
{ if(i != j)
    { parent[j] = i;
      return 1;
    }
  return 0;
}
```

Output:

Kruskal's Algorithm:

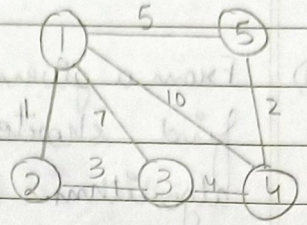
Enter the number of vertices: 5

→

PAGE NO :
DATE :

Enter the adjacency matrix:

0	1	7	10	5
1	0	3	∞	∞
7	3	0	4	∞
10	∞	4	0	2
5	∞	∞	2	0



The edges of Minimum Cost Spanning Tree are:

Edge 1: (1,2) Weight: 1
Edge 2: (4,5) Weight: 2
Edge 3: (2,3) Weight: 3
Edge 4: (3,4) Weight: 4

Minimum Cost = 10

1/8/23

2.10 Experiment - 10

2.10.1 Question:

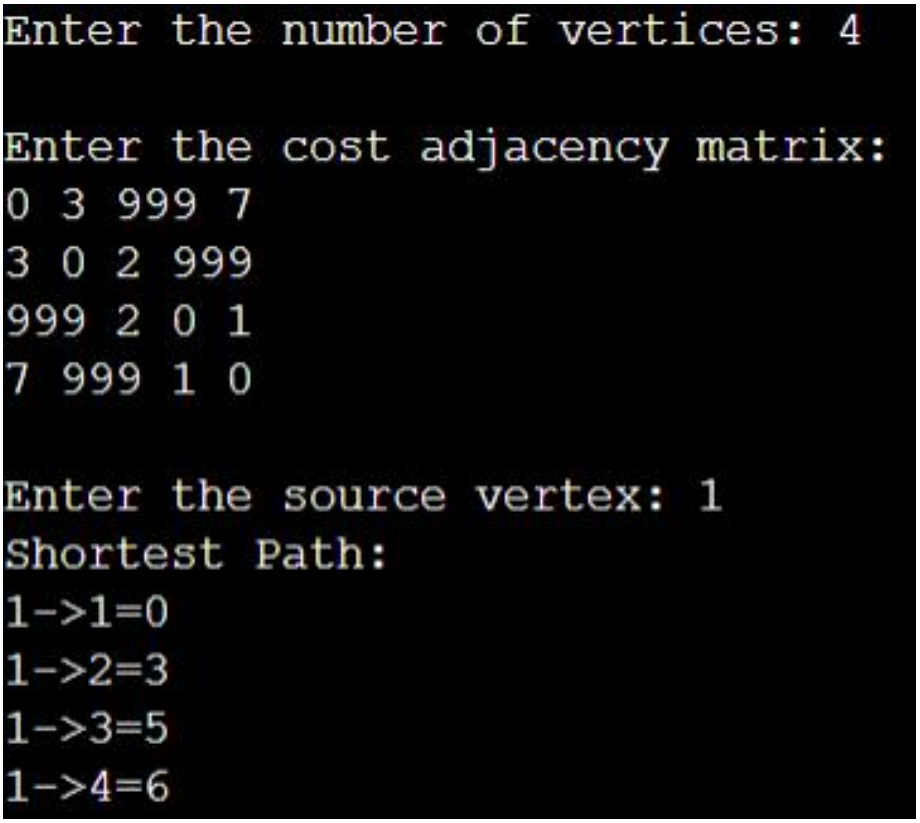
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

2.10.2 Code:

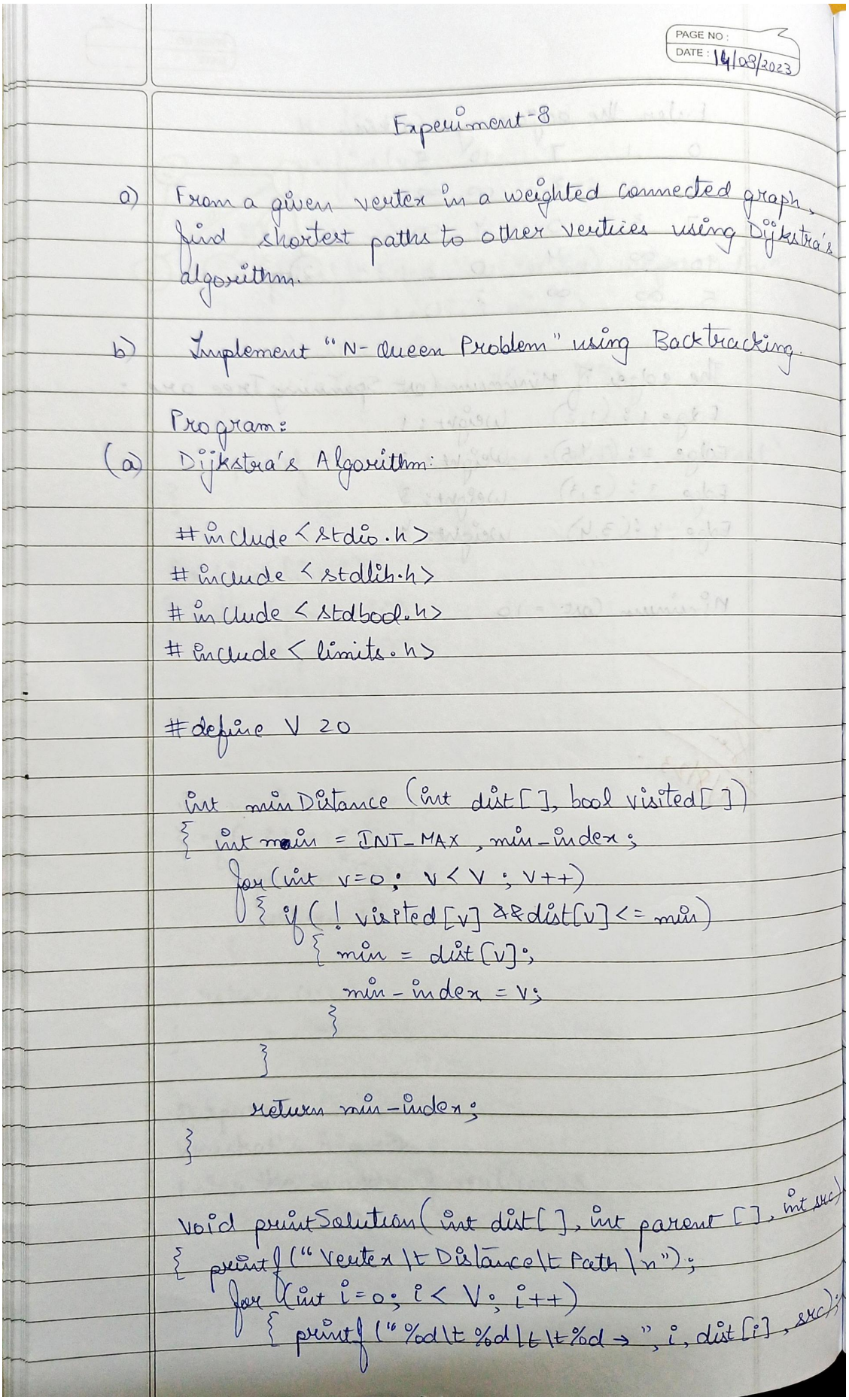
```
#include<stdio.h>
#include<conio.h>

void dijkstra(int n,int cost[10][10],int src)
{
   int i,j,u,dis[10],vis[10],min;
   for(i=1;i<=n;i++)
   {
      dis[i]=cost[src][i];
      vis[i]=0;
   }

   vis[src]=1;
   for(i=1;i<=n;i++)
   {
      min=999;
      for(j=1;j<=n;j++)
      {
         if(vis[j]==0 && dis[j]<min)
         {
            min=dis[j];
            u=j;
         }
      }

      vis[u]=1;
      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 Path:\n");
   for(i=1;i<=n;i++)
   printf("%d->%d=%d\n",src,i,dis[i]);
```

```
}

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

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

   printf("\nEnter the cost adjacency matrix: \n");
   for(i=1;i<=n;i++)
     for(j=1;j<=n;j++)
         scanf("%d",&cost[i][j]);

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

   dijkstra(n,cost,src);
}
```

2.10.3 Output:



2.10.4 Observation Book Pictures:



PAGE NO :
DATE : 14/09/2023

Experiment-8

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

b) Implement "N-Queen Problem" using Backtracking.

Program:

(a) Dijkstra's Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

#define V 20

int minDistance (int dist[], bool visited[])
{ int min = INT_MAX , min_index;
    for(int v=0; v<V; v++)
    { if( !visited[v] && dist[v]<=min)
        { min = dist[v];
          min_index = v;
        }
    }
    return min_index;
}

void printSolution( int dist[], int parent[], int src)
{ printf("Vertex \t Distance \t Path \n");
    for (int i=0; i<V; i++)
    { printf("%d \t %d \t %d -> ", i, dist[i], src);
```

```
        printf("\n");
    }
}

void dijkstra(int graph[V][V], int src)
{ int dist[V];
  bool visited[V];
  int parent[V];

  for(int i=0; i<V; i++)
  { dist[i] = INT_MAX;
    visited[i] = false;
    parent[i] = -1;
  }

  dist[src] = 0;

  for(int count=0; count<V-1; count++)
  { int u = minDistance(dist, visited);
    visited[u] = true;

    for(int v=0; v<V; v++)
    { if(!visited[v] && graph[u][v] &&
         dist[u] != INT_MAX &&
         dist[u] + graph[u][v] < dist[v])
      { dist[v] = dist[u] + graph[u][v];
        parent[v] = u;
      }
    }
  }
  printSolution(dist, parent, src);
}
```

→

```
int main()
{ int graph[V][V];
  int vertices, edges;

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

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

for (int i=0; i<V; i++)
{ for(int j=0; j<V; j++)
  { graph[i][j] = 0;
  }
}

printf("Enter the edges and weights:\n");
for (int i=0; i< edges; i++)
  { int u,v,w;
    scanf("%d %d %d", &u, &v, &w);
    graph[u][v] = w;
  }

int source;
printf("Enter the source vertex (0 to %d): ",
        vertices-1);
scanf("%d", &source);

dijkstra(graph, source);

return 0;

}
```

PAGE NO :
DATE :

Output:

Enter the number of vertices (<=20) : 4

Enter the number of edges : 7

Enter the edges and weights :

0 1 3

0 2 4

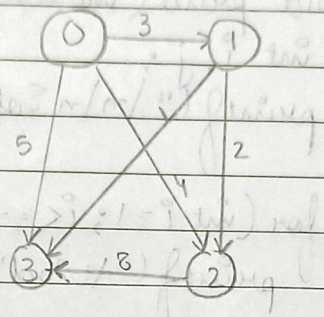
0 3 5

1 2 2

1 3 1

2 3 8

3 2 3



Enter the source vertex (0 to 3) : 0

Vertex	Distance	Path
0	0	0
1	3	0 → 1
2	4	0 → 2
3	4	0 → 1 → 3

2.11 Experiment - 11

2.11.1 Question:

Implement “N-Queens Problem” using Backtracking.

2.11.2 Code:

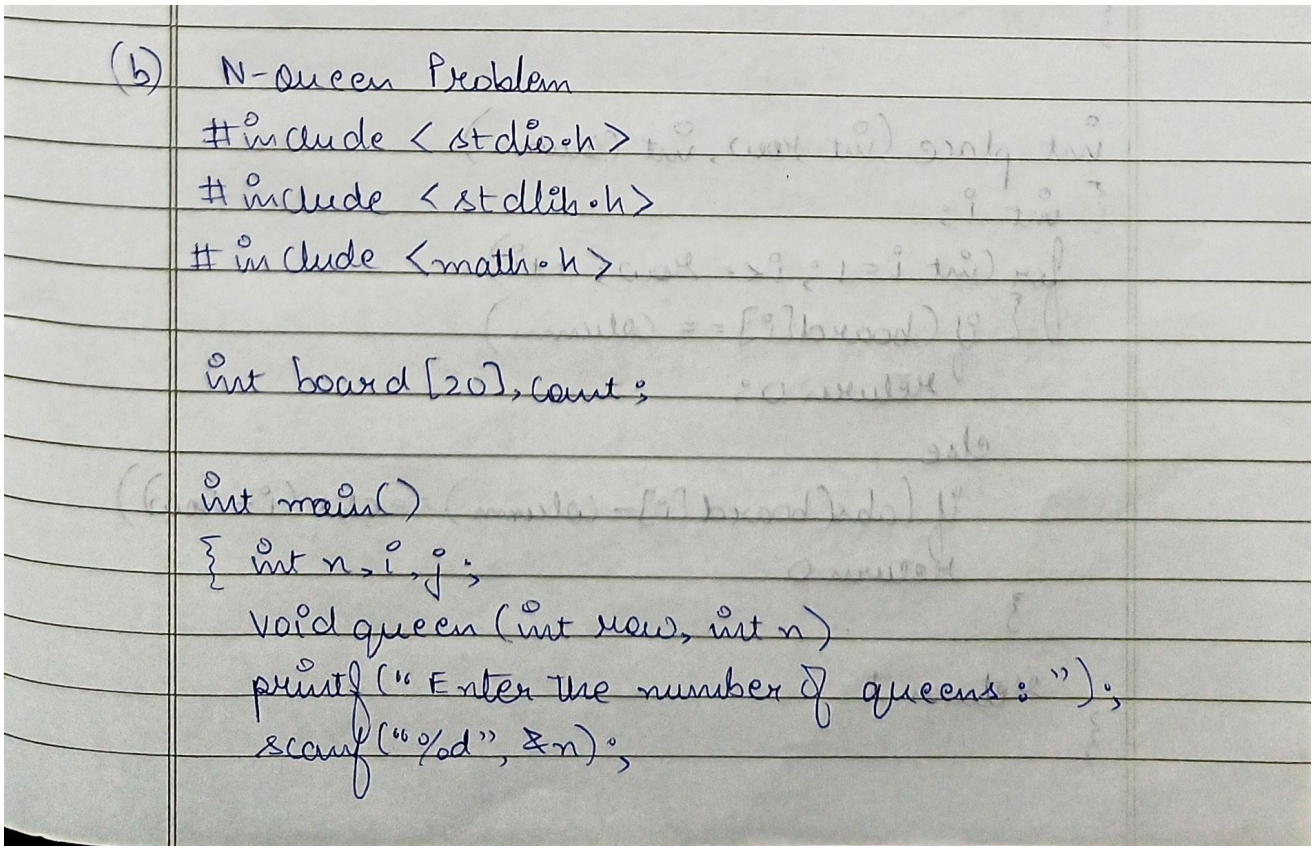
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

int board[20],count;

int main()
{
   int n,i,j;

   void queen(int row,int n);

   printf("N Queens Problem Using Backtracking:");
   printf("\n\nEnter number of Queens:");
   scanf("%d",&n);

   queen(1,n);
   return 0;
}

void print(int n)
{
   int i,j;
   printf("\n\nSolution %d:\n\n",++count);

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

   for(i=1;i<=n;++i)
   {
      printf("\n\n%d",i);
      for(j=1;j<=n;++j)
      {
         if(board[i]==j)
            printf("\tQ");

         else
            printf("\t-");
      }
```

```
    }
}

int place(int row,int column)
{
    int i;
    for(i=1;i<=row-1;++i)
    {
        if(board[i]==column)
            return 0;

        else
            if(abs(board[i]-column)==abs(i-row))
            return 0;
    }
    return 1;
}

void queen(int row,int n)
{
    int column;
    for(column=1;column<=n;++column)
    {
        if(place(row,column))
        {
            board[row]=column;
            if(row==n)
                print(n);
            else
                queen(row+1,n);
        }
    }
}
```

2.11.3 Output:

```
N Queens Problem Using Backtracking:

Enter number of Queens:4


Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```

2.11.4 Observation Book Pictures:


(b) N-Queen Problem

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

int board[20], count;

int main()
{ int n, i, j;
  void queen(int row, int n)
  printf("Enter the number of queens: ");
  scanf("%d", &n);
```

```
    queen(1,n);
    return 0;
}

void print(int n)
{ int i, j;
  printf("\n\n Solution %d:\n\n", ++count);

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

  for(i=1; i<=n; ++i)
  { printf("\n\n%d", i);
    for(j=1; j<=n; ++j)
    { if( board[i]==j )
        print("\tQ");
      else
        print("\t-");
    }
  }
}

int place(int row, int column)
{ int i;
  for(int i=1; i<=row; ++i)
  { if(board[i]==column)
      return 0;
    else
      if(abs(board[i]-column)==abs(i-row))
      return 0
  }

  return 1
}
```

PAGE NO :
DATE :

```
void queen (int row, int n)
{ int column;
  for (column = 1; column <= n; ++column)
  { if (place (row, column))
    { board[row] = column;
      if (row == n)
        print (n);
      else
        queen (row+1, n);
    }
  }
}
```

Output:

Enter the number of queens : 4

Solution-1:

	1	2	3	4
1	-	0	-	-
2	-	-	-	0
3	0	-	-	-
4	-	-	0	-

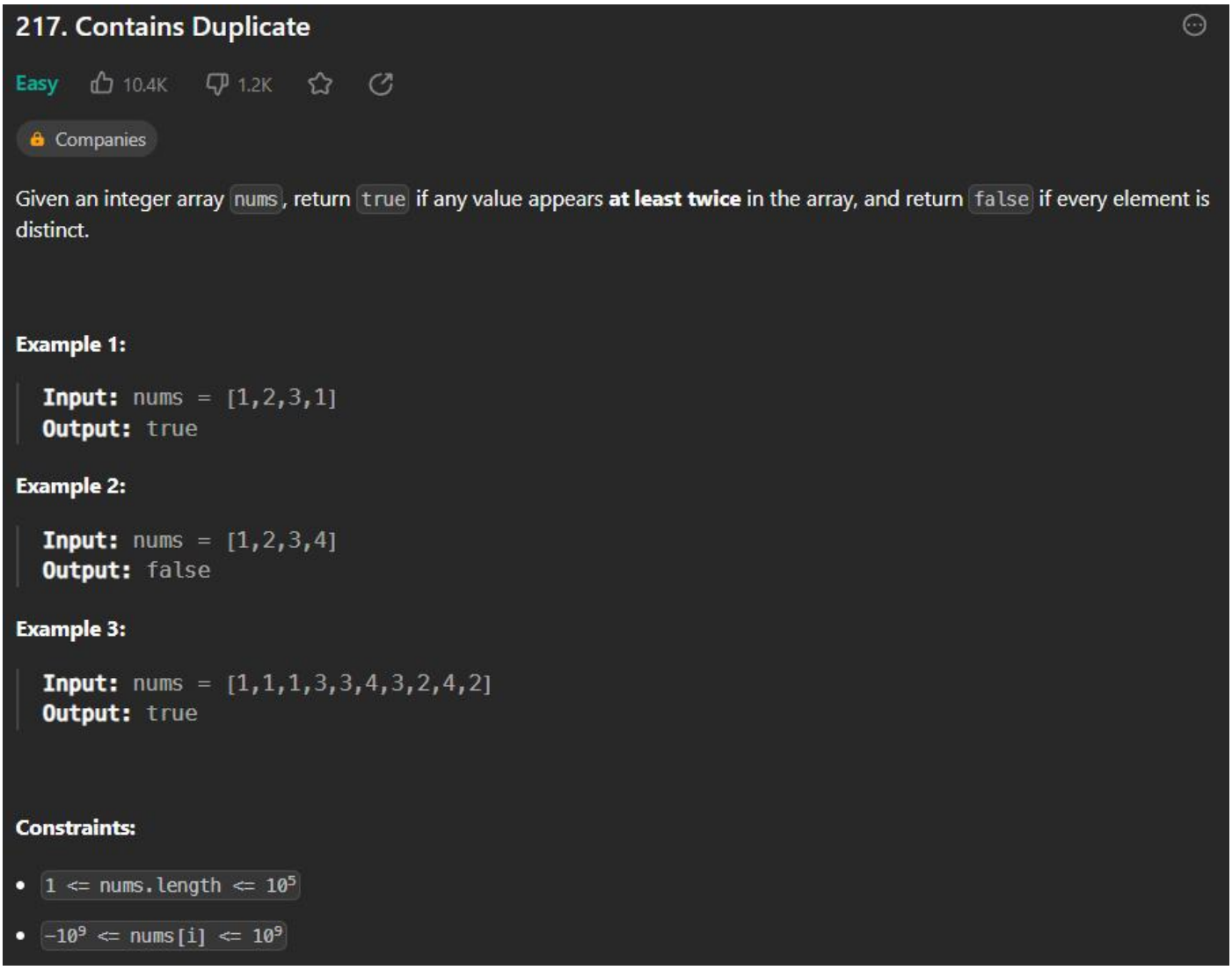
Sun 16/8/23.

Solution-2:

	1	2	3	4
1	-	-	0	-
2	0	-	-	-
3	-	-	-	0
4	-	0	-	-

3. LeetCode Problems:

3.1 Problem - 1:



217. Contains Duplicate

Easy 10.4K 1.2K

Companies

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

```
Input: nums = [1,2,3,1]
Output: true
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: false
```

Example 3:

```
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true
```

Constraints:

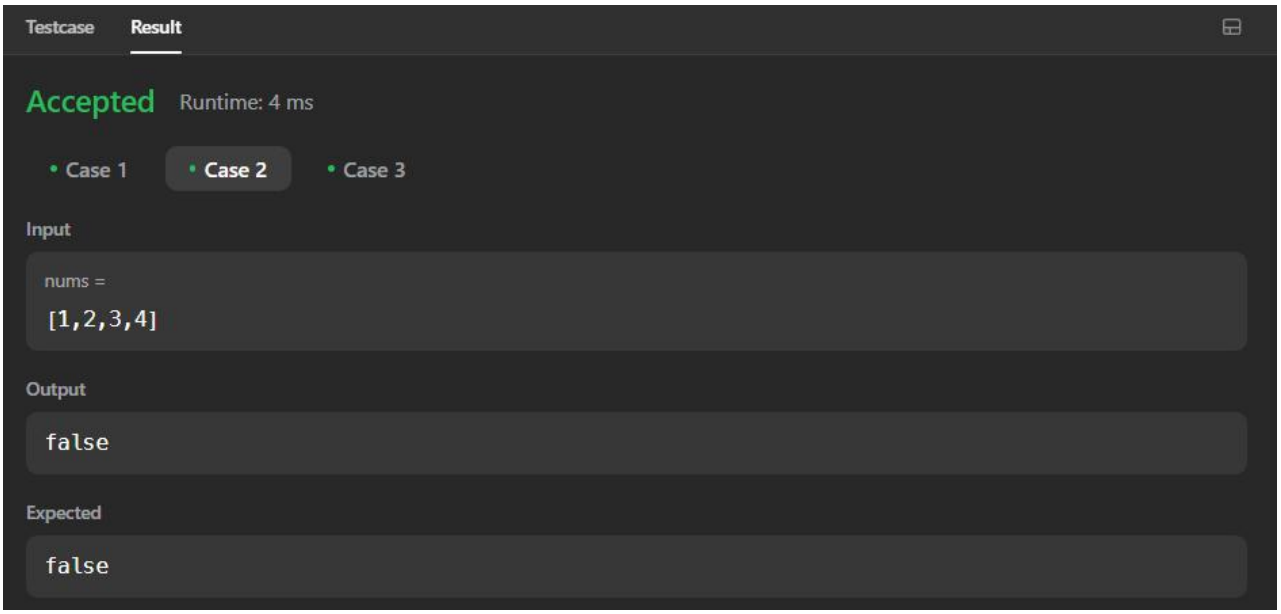
- $1 <= nums.length <= 10^5$
- $-10^9 <= nums[i] <= 10^9$

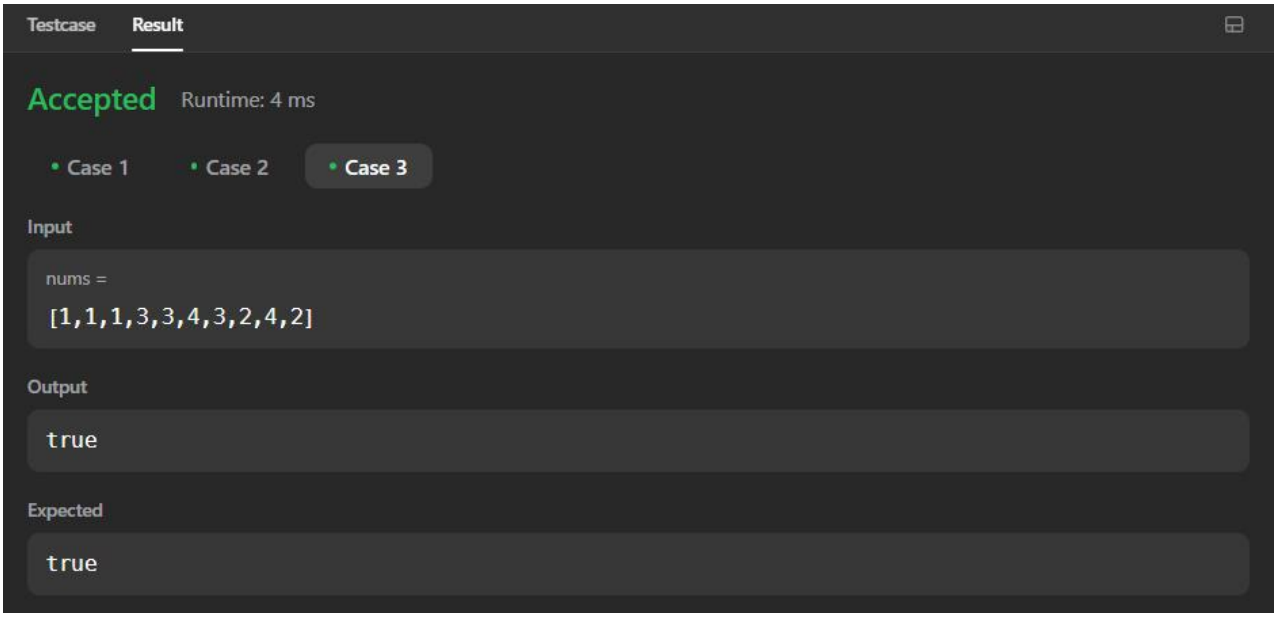
3.1.1 Code:

```
bool containsDuplicate(int* nums, int numsSize) {
    bool flag = false;
    for(int i = 0; i < numsSize; i++){
        for(int j = i + 1; j < numsSize; j++){
            if(nums[i] == nums[j]) return true;
        }
    }
    return flag;
}
```

3.1.2 Output:


Testcase
Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
nums =
[1,2,3,1]
Output
true
Expected
true


Testcase
Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
nums =
[1,2,3,4]
Output
false
Expected
false


Testcase
Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Case 3
Input
nums =
[1,1,1,3,3,4,3,2,4,2]
Output
true
Expected
true

3.2 Problem - 2:

2685. Count the Number of Complete Components

Hint

Medium 457 11

Companies

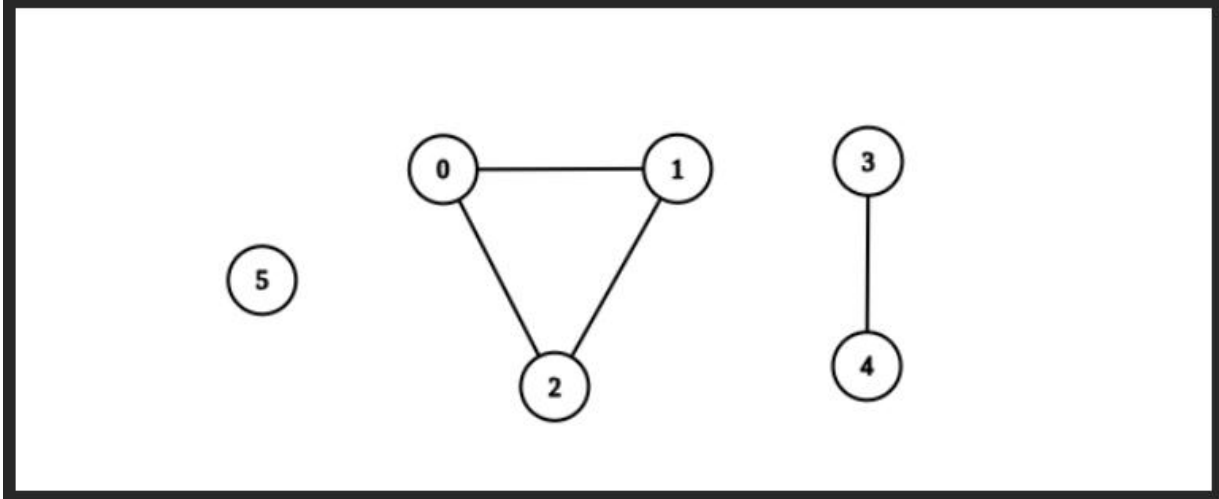
You are given an integer `n`. There is an **undirected** graph with `n` vertices, numbered from `0` to `n - 1`. You are given a 2D integer array `edges` where `edges[i] = [a_i, b_i]` denotes that there exists an **undirected** edge connecting vertices `a_i` and `b_i`.

Return *the number of **complete connected components** of the graph.*

A **connected component** is a subgraph of a graph in which there exists a path between any two vertices, and no vertex of the subgraph shares an edge with a vertex outside of the subgraph.

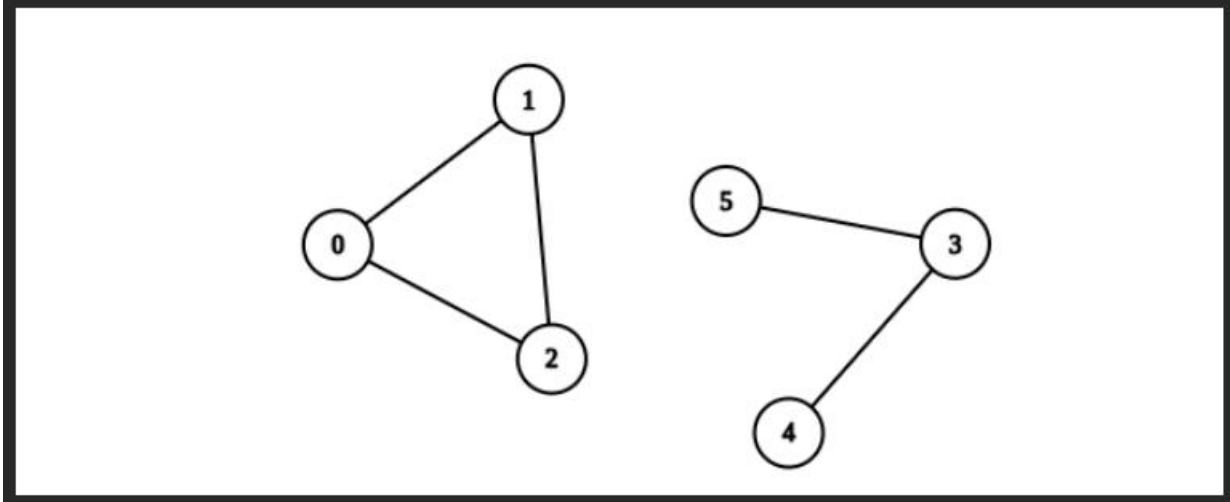
A connected component is said to be **complete** if there exists an edge between every pair of its vertices.

Example 1:




```
Input: n = 6, edges = [[0,1],[0,2],[1,2],[3,4]]
Output: 3
Explanation: From the picture above, one can see that all of the components of this graph are complete.
```

Example 2:



Input: n = 6, edges = [[0,1],[0,2],[1,2],[3,4],[3,5]]
Output: 1
Explanation: The component containing vertices 0, 1, and 2 is complete since there is an edge between every pair of two vertices. On the other hand, the component containing vertices 3, 4, and 5 is not complete since there is no edge between vertices 4 and 5. Thus, the number of complete components in this graph is 1.

Constraints:

- `1 <= n <= 50`
- `0 <= edges.length <= n * (n - 1) / 2`
- `edges[i].length == 2`
- `0 <= a_i, b_i <= n - 1`
- `a_i != b_i`
- There are no repeated edges.

3.2.1 Code:

```
void dfs(int src, int** g, int n, int* nodes, int* vis, int* nodeCount)
{
   if (vis[src])
      return;
   vis[src] = 1;

   nodes[*nodeCount] = src;

   (*nodeCount)++;

   for (int i = 0; i < n; i++)
   {
      if (g[src][i] && !vis[i])
```

```
        {
            dfs(i, g, n, nodes, vis, nodeCount);
        }
    }
}

int countCompleteComponents(int n, int** edges, int edgesSize, int* edgesColSize)
{
    int** g = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++)
    {
        g[i] = (int*)calloc(n, sizeof(int));
    }

    for (int i = 0; i < edgesSize; i++)
    {
        int u = edges[i][0];
        int v = edges[i][1];

        g[u][v] = 1;
        g[v][u] = 1;
    }

    int* vis = (int*)calloc(n, sizeof(int));
    int res = 0;

    for (int i = 0; i < n; i++)
    {
        if (!vis[i])
        {
            int* nodes = (int*)malloc(n * sizeof(int));
            int nodeCount = 0;
            dfs(i, g, n, nodes, vis, &nodeCount);

            int count = 0;

            for (int j = 0; j < nodeCount; j++)
            {
                int pathNode = nodes[j];
                int neighborCount = 0;

                for (int k = 0; k < n; k++)
                {
                    if (g[pathNode][k])
                    {
```

```
                neighborCount++;
            }
        }

        if (neighborCount >= nodeCount - 1)
        {
            count++;
        }
    }

    if (count == nodeCount)
    {
        res++;
    }
    free(nodes);
  }
 }

 for (int i = 0; i < n; i++)
 {
    free(g[i]);
 }

 free(g);

 free(vis);

 return res;
}
```

3.2.2 Output:


Testcase
Result
Accepted
Runtime: 5 ms
Case 1
Case 2
Input
n =
6
edges =
[[0,1],[0,2],[1,2],[3,4]]
Output
3
Expected
3


Testcase
Result
Accepted
Runtime: 5 ms
Case 1
Case 2
Input
n =
6
edges =
[[0,1],[0,2],[1,2],[3,4],[3,5]]
Output
1
Expected
1

3.3 Problem - 3:

64. Minimum Path Sum

Medium 11.5K 149

Companies

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

Example 1:


```
Input: grid = [[1,3,1],[1,5,1],[4,2,1]]
Output: 7
Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.
```

Example 2:

```
Input: grid = [[1,2,3],[4,5,6]]
Output: 12
```

Constraints:

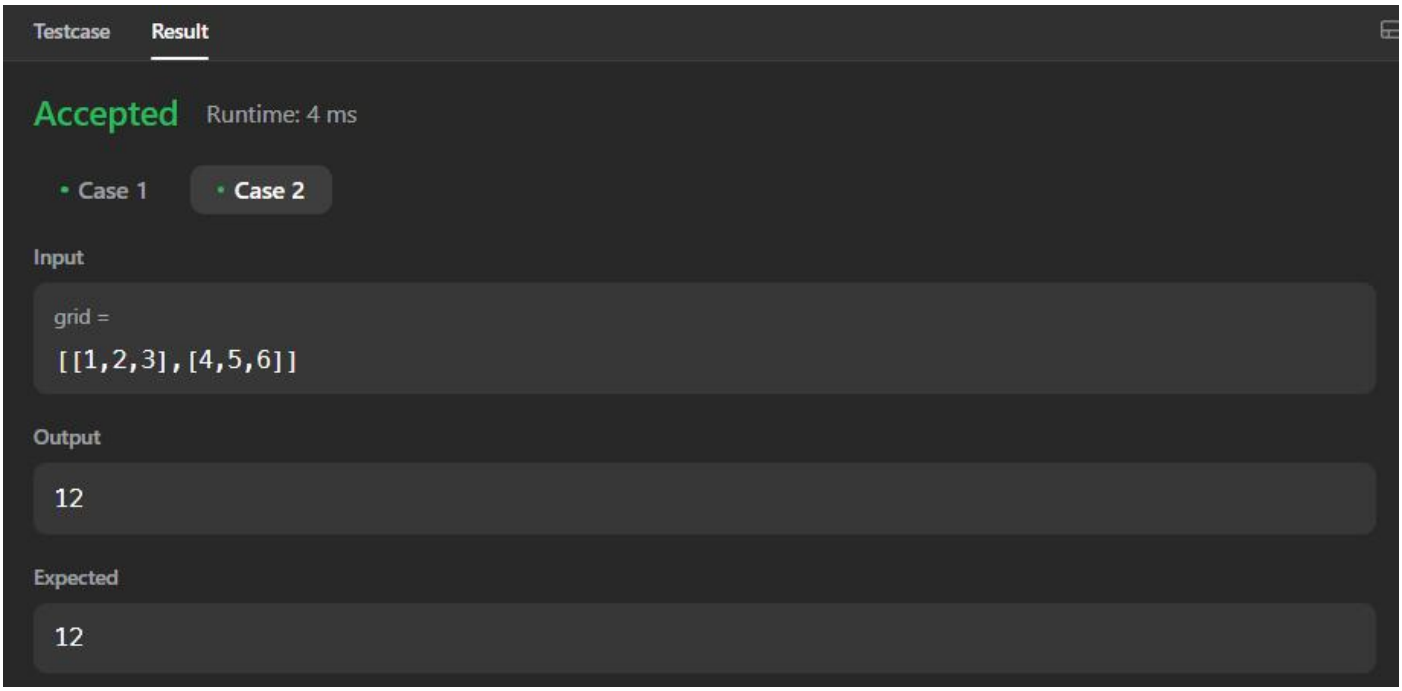
- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 200`
- `0 <= grid[i][j] <= 200`

3.3.1 Code:

```
int recursive(int** grid, int i, int j, int m, int n) {
    if (i < 0 || i >= m || j < 0 || j >= n) {
        return 0;
    }
    if (i == m - 1 && j == n - 1) {
        return grid[i][j];
    }
    int take1 = INT_MAX, take2 = INT_MAX;
    if (i + 1 < m) {
        take1 = recursive(grid, i + 1, j, m, n);
    }
    if (j + 1 < n) {
        take2 = recursive(grid, i, j + 1, m, n);
    }
    return grid[i][j] + ((take1 < take2) ? take1 : take2);
}

int minPathSum(int** grid, int gridSize, int* gridColSize) {
    int m = gridSize, n = *gridColSize;
    return recursive(grid, 0, 0, m, n);
}
```

3.3.2 Output:


Testcase
Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Input
grid =
[[1,3,1],[1,5,1],[4,2,1]]
Output
7
Expected
7


Testcase
Result
Accepted
Runtime: 4 ms
Case 1
Case 2
Input
grid =
[[1,2,3],[4,5,6]]
Output
12
Expected
12

3.4 Problem - 4:

122. Best Time to Buy and Sell Stock II

Medium 12.2K 2.6K

Companies

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the `i^th` day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the **maximum** profit you can achieve.*

Example 1:

> **Input:** prices = [7,1,5,3,6,4]
> **Output:** 7
> **Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
> Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
> Total profit is 4 + 3 = 7.

Example 2:

> **Input:** prices = [1,2,3,4,5]
> **Output:** 4
> **Explanation:** Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
> Total profit is 4.

Example 3:

> **Input:** prices = [7,6,4,3,1]
> **Output:** 0
> **Explanation:** There is no way to make a positive profit, so we never buy the stock to achieve the maximum profit of 0.

Constraints:

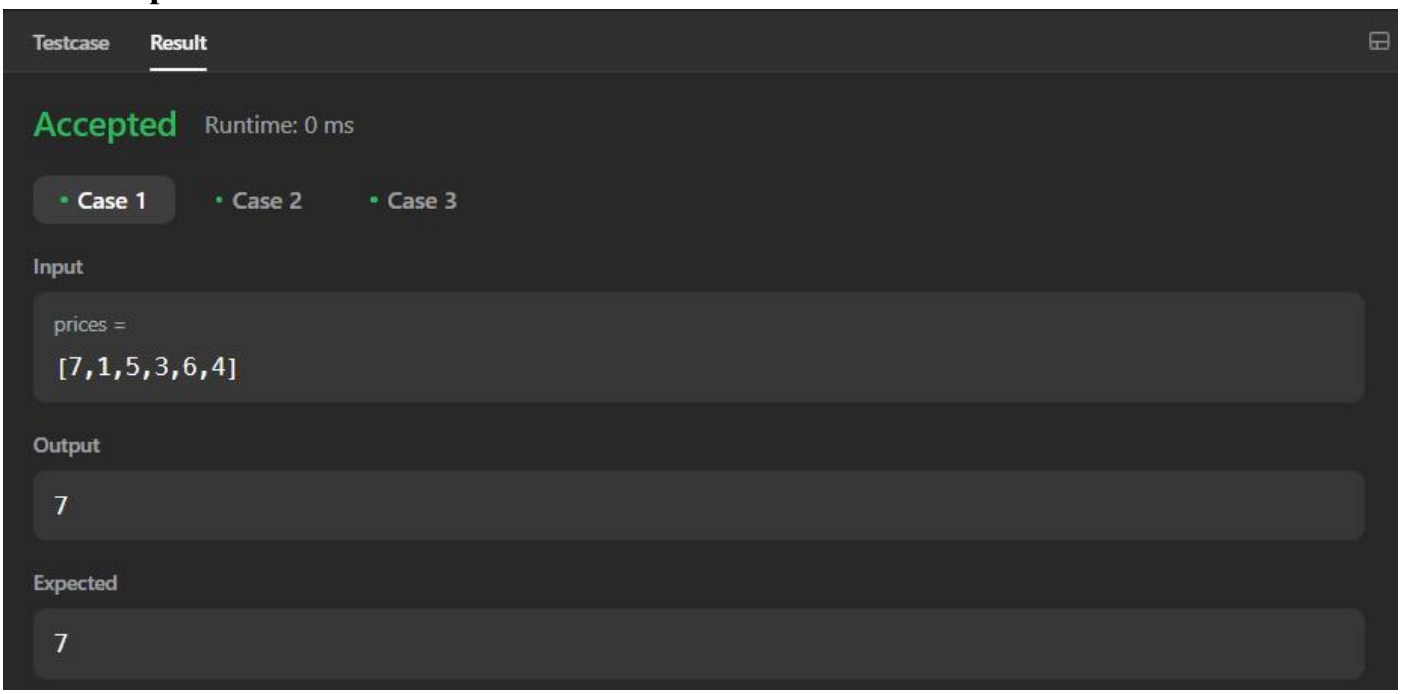
- `1 <= prices.length <= 3 * 10^4`
- `0 <= prices[i] <= 10^4`

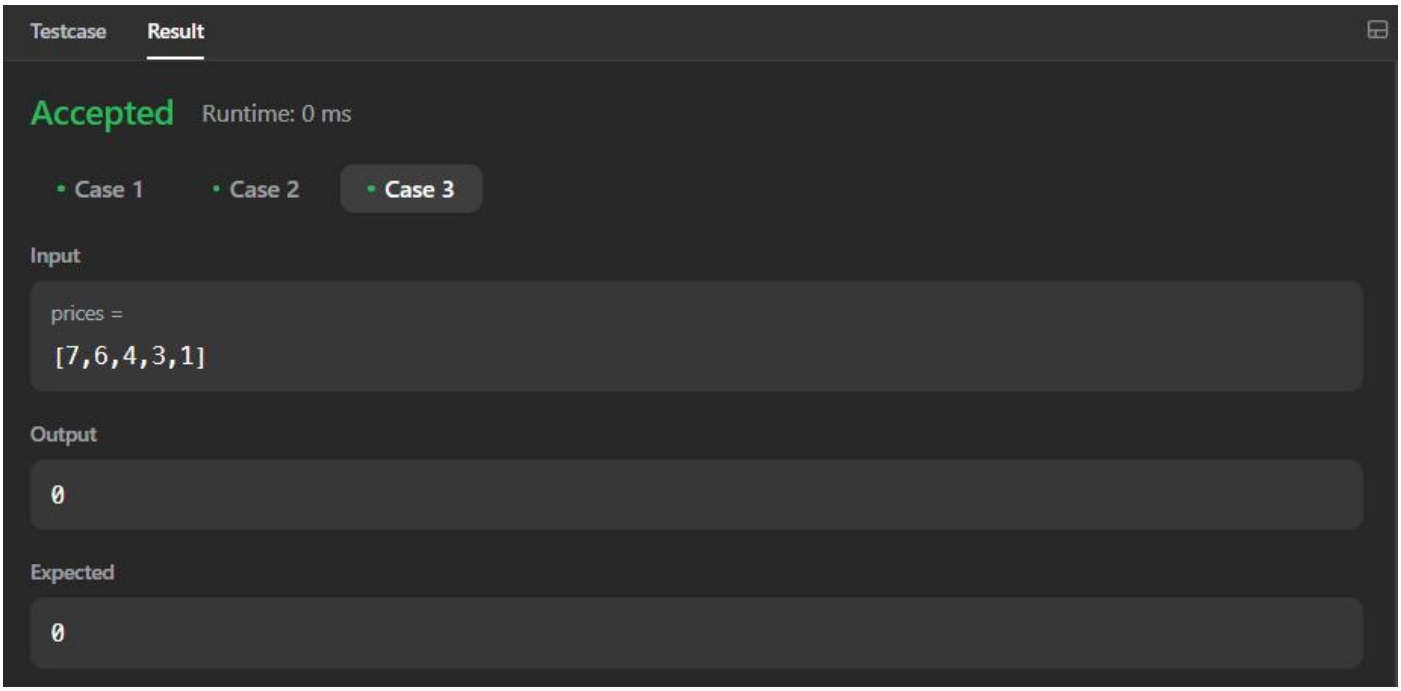
3.4.1 Code:

```
int maxProfit(int prices[], int length) {
    int profit = 0;
    for (int i = 1; i < length; i++) {
        if (prices[i] > prices[i - 1]) {
            profit += prices[i] - prices[i - 1];
        }
    }
```

```
    return profit;
}
```

3.4.2 Output:




Testcase
Result
Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
prices =
[7,6,4,3,1]
Output
0
Expected
0