Student Name:-Srishti Shrivastava

Student Roll No.:- 1905647

Algorithm Lab. Class Assignment-2

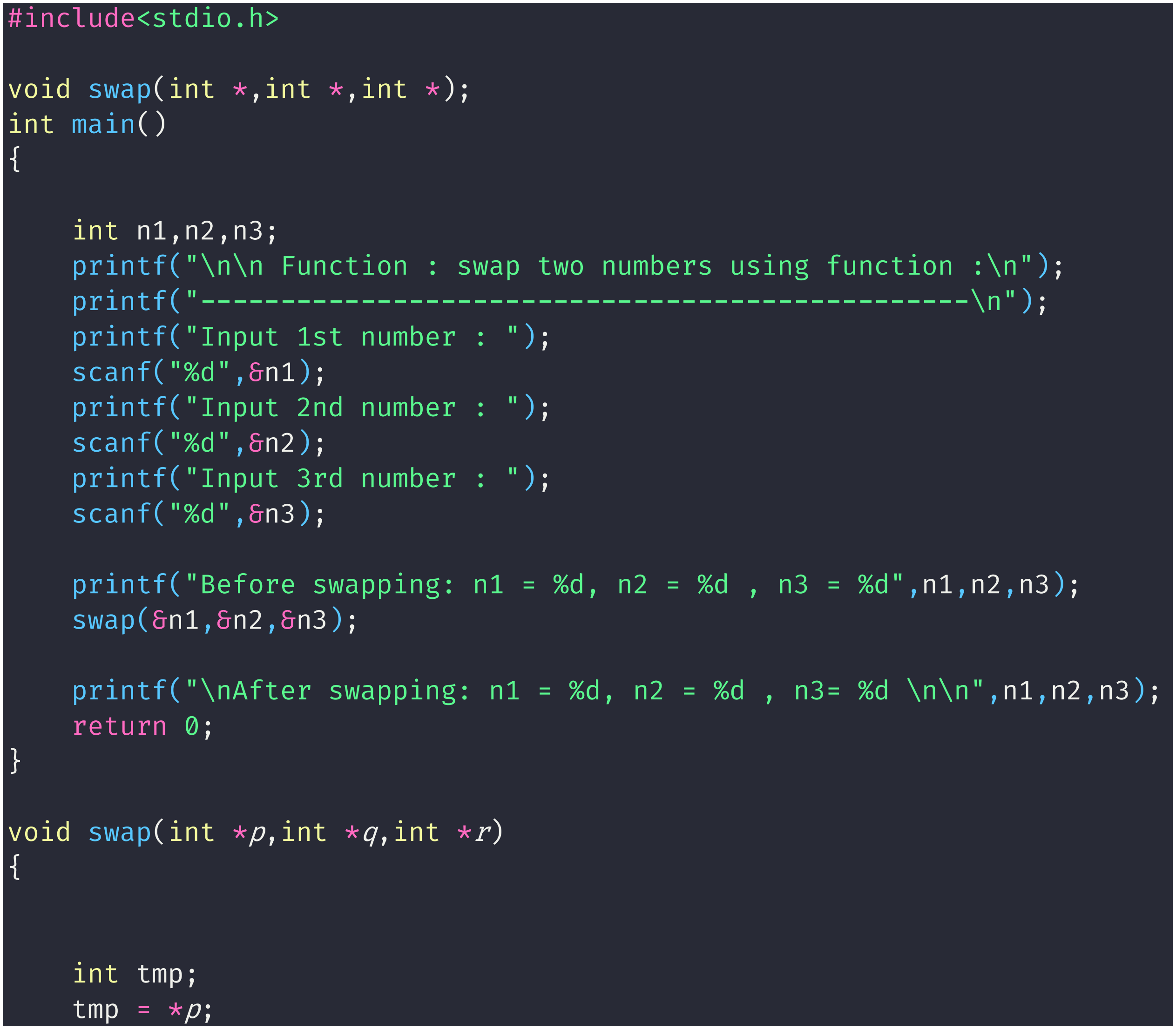
CSE Group 1

Date: - 16th July 2021

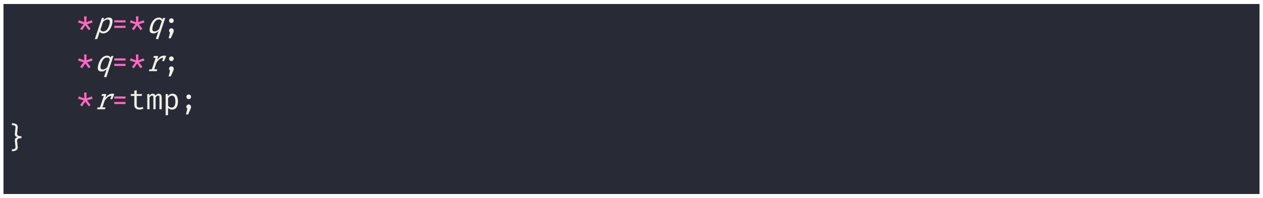
(Upload only the PDF file and the name of the file must be your roll number.)

1. Write a program that takes three variables (A, B, C) as separate parameters and rotates the values stored so that value A goes to B, B to C, and C to A by using SWAP(x,y) as a function that swaps/exchanges the numbers x & y.

Program

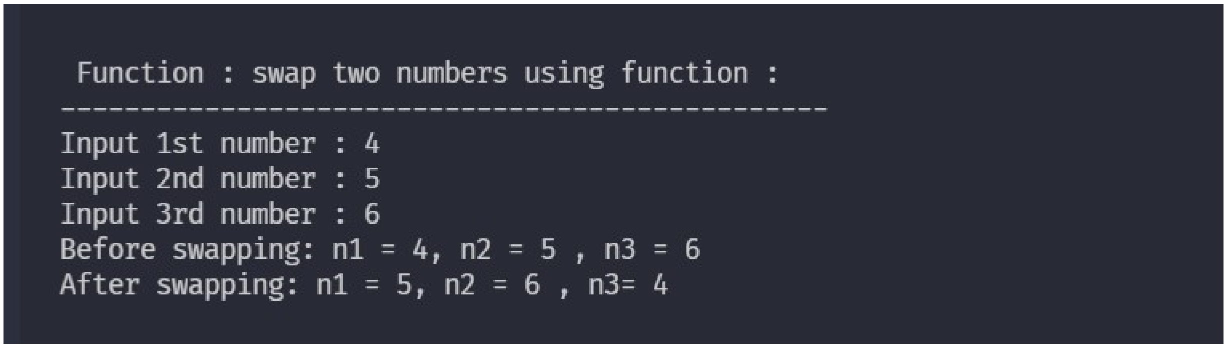


```c
#include<stdio.h>

void swap(int *,int *,int *);
int main()
{

    int n1,n2,n3;
    printf("\n\n Function : swap two numbers using function :\n");
    printf("------------------------------------------------\n");
    printf("Input 1st number : ");
    scanf("%d",&n1);
    printf("Input 2nd number : ");
    scanf("%d",&n2);
    printf("Input 3rd number : ");
    scanf("%d",&n3);

    printf("Before swapping: n1 = %d, n2 = %d , n3 = %d",n1,n2,n3);
    swap(&n1,&n2,&n3);

    printf("\nAfter swapping: n1 = %d, n2 = %d , n3= %d \n\n",n1,n2,n3);
    return 0;
}

void swap(int *p,int *q,int *r)
{


    int tmp;
    tmp = *p;
```



```
    *p=*q;
    *q=*r;
    *r=tmp;
}
```

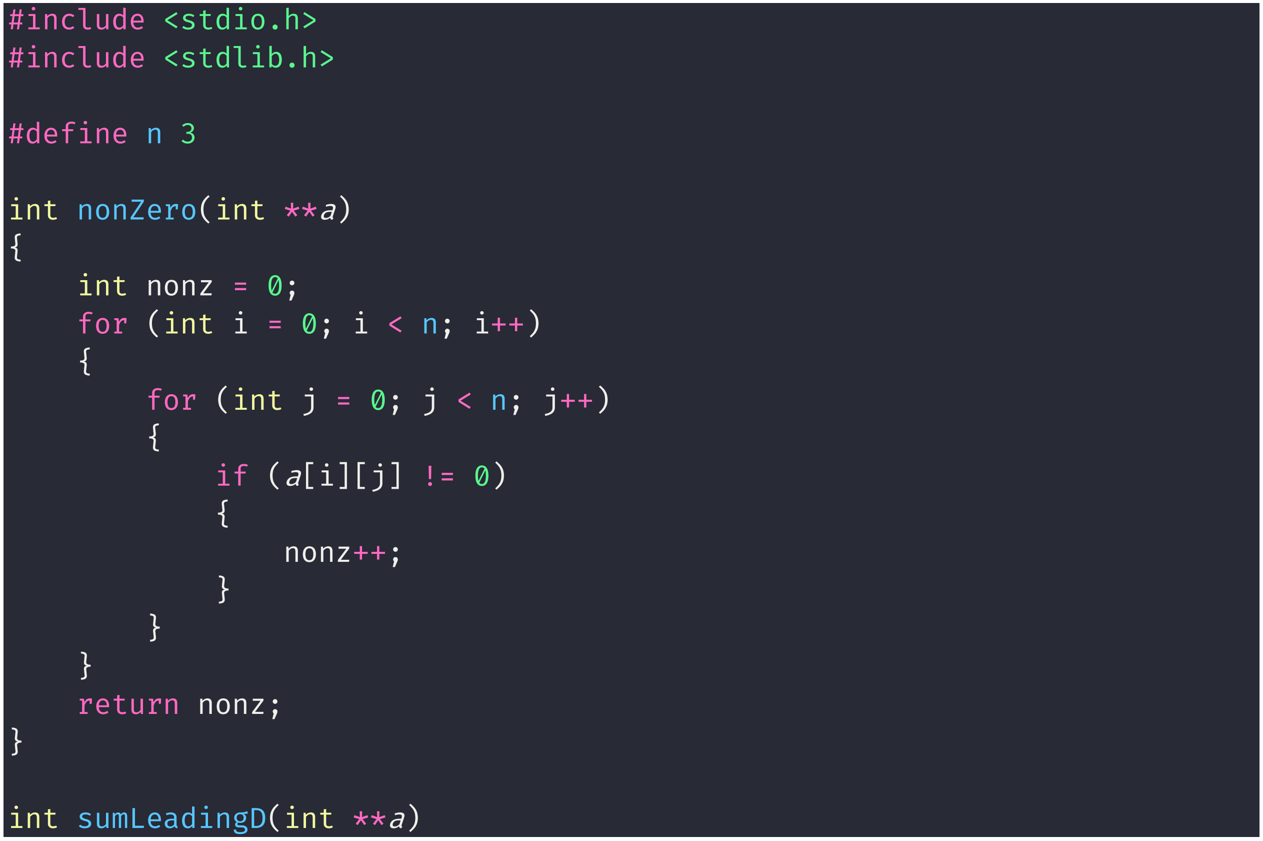
Output



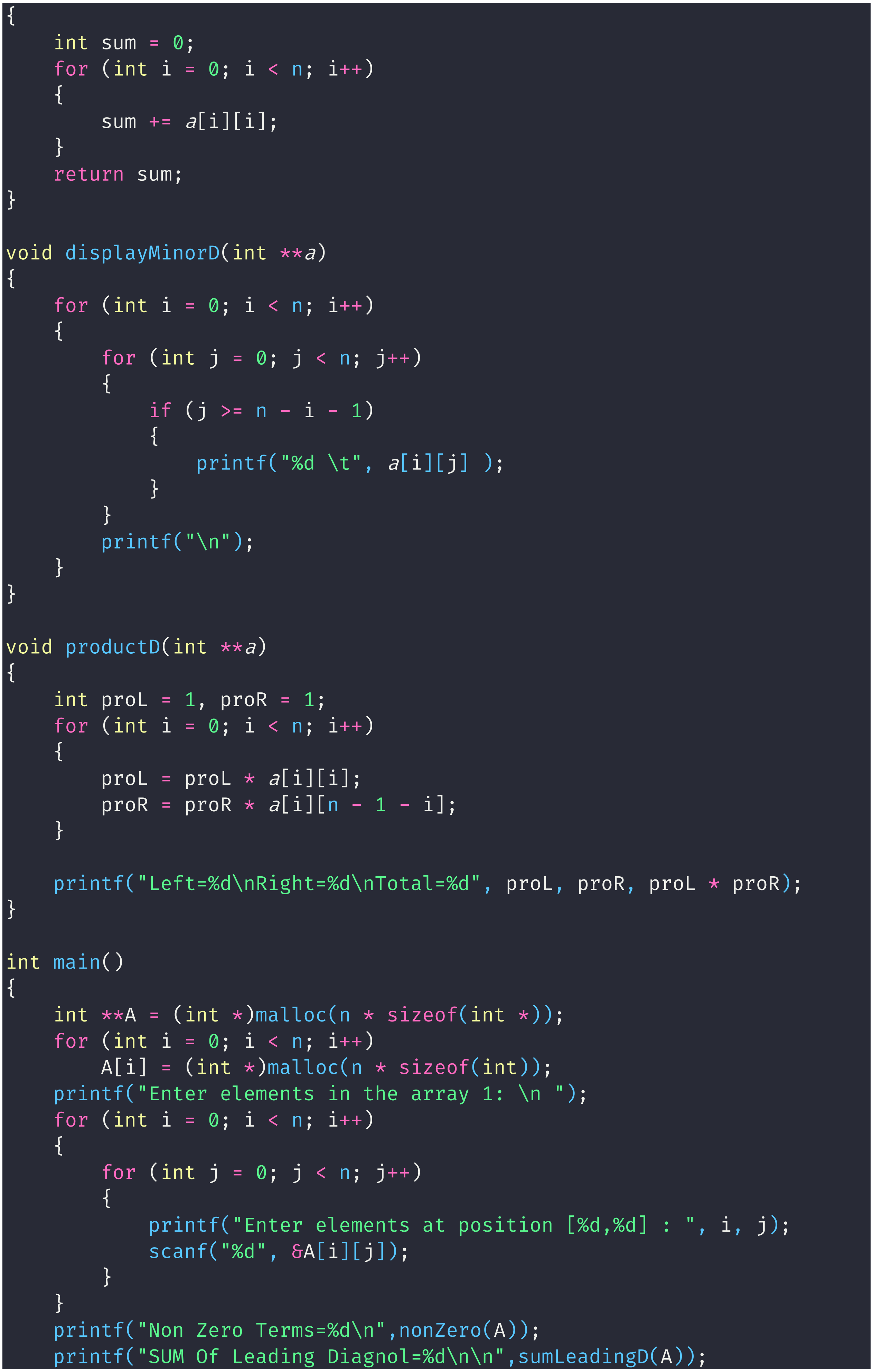
```
 Function : swap two numbers using function :
------------------------------------------------
Input 1st number : 4
Input 2nd number : 5
Input 3rd number : 6
Before swapping: n1 = 4, n2 = 5 , n3 = 6
After swapping: n1 = 5, n2 = 6 , n3= 4
```

2. Let A be n*n square matrix array. WAP by using appropriate user-defined functions for the following:
    a) Find the number of nonzero elements in A
    b) Find the sum of the elements above the leading diagonal.
    c) Display the elements below the minor diagonal.
    d) Find the product of the diagonal elements.

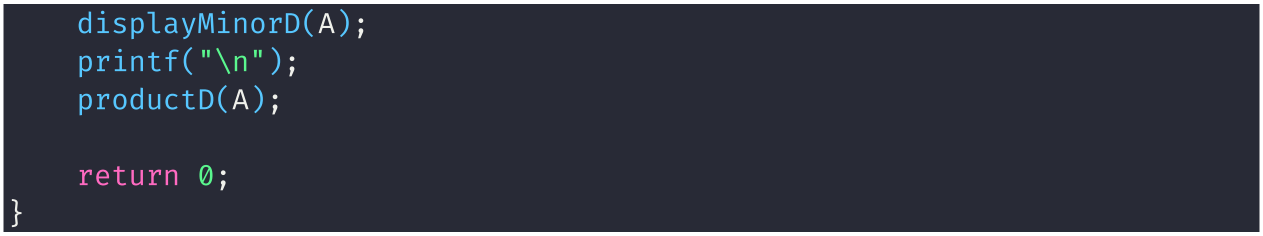
Program



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

#define n 3

int nonZero(int **a)
{
    int nonz = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (a[i][j] != 0)
            {
                nonz++;
            }
        }
    }
    return nonz;
}

int sumLeadingD(int **a)
```

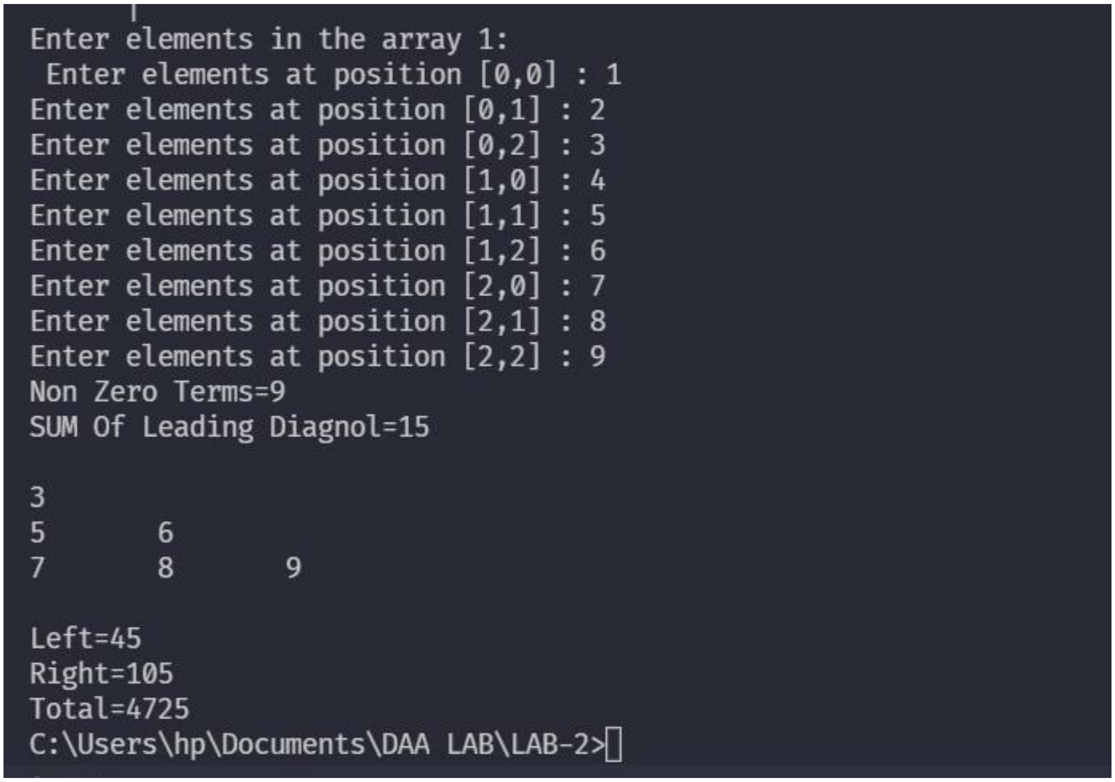


```
{
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        sum += a[i][i];
    }
    return sum;
}

void displayMinorD(int **a)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (j >= n - i - 1)
            {
                printf("%d \t", a[i][j] );
            }
        }
        printf("\n");
    }
}

void productD(int **a)
{
    int proL = 1, proR = 1;
    for (int i = 0; i < n; i++)
    {
        proL = proL * a[i][i];
        proR = proR * a[i][n - 1 - i];
    }

    printf("Left=%d\nRight=%d\nTotal=%d", proL, proR, proL * proR);
}

int main()
{
    int **A = (int *)malloc(n * sizeof(int *));
    for (int i = 0; i < n; i++)
        A[i] = (int *)malloc(n * sizeof(int));
    printf("Enter elements in the array 1: \n ");
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("Enter elements at position [%d,%d] : ", i, j);
            scanf("%d", &A[i][j]);
        }
    }
    printf("Non Zero Terms=%d\n",nonZero(A));
    printf("SUM Of Leading Diagnol=%d\n\n",sumLeadingD(A));
```



```
    displayMinorD(A);
    printf("\n");
    productD(A);

    return 0;
}
```

Output



```
Enter elements in the array 1:
 Enter elements at position [0,0] : 1
Enter elements at position [0,1] : 2
Enter elements at position [0,2] : 3
Enter elements at position [1,0] : 4
Enter elements at position [1,1] : 5
Enter elements at position [1,2] : 6
Enter elements at position [2,0] : 7
Enter elements at position [2,1] : 8
Enter elements at position [2,2] : 9
Non Zero Terms=9
SUM Of Leading Diagnol=15

3
5       6
7       8       9

Left=45
Right=105
Total=4725
C:\Users\hp\Documents\DAA LAB\LAB-2>
```

3. WAP in C to store 1 million integers in an array. To search an element in that array and find out its time complexity (best, worst, and average).

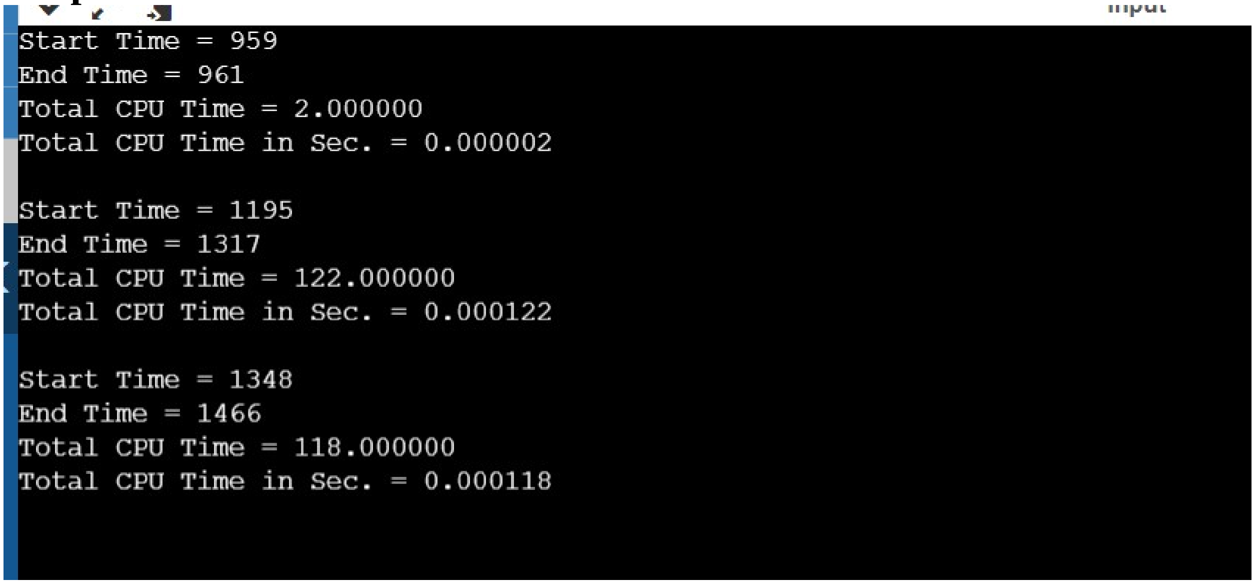
Program

```
*********************************************************
**********************/
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main(){
   int n=1000000;
   int a[n];
   for(int i=0; i<n; i++){
         a[i]=i+1;
```

```
  }
  int c1=a[0];
  int c2=a[n-1];
  clock_t start,end;
  double total_cputime;

  ///FOR BEST CASE
  start=clock();
  for(int i=0; i<n; i++){
      if(a[i]==c1){
          end=clock();
          printf("Start Time = %ld\n",start);
          printf("End Time = %ld\n",end);
          total_cputime=(double)(end-start);
          printf("Total CPU Time = %f\n",total_cputime);
          total_cputime=((double)(end-
start))/CLOCKS_PER_SEC;
          printf("Total CPU Time in Sec. =
%f\n",total_cputime);
      }
  }
  printf("\n");

  ///FOR WORST CASE
  start=clock();
  for(int i=0; i<n; i++){
      if(a[i]==c2){
          end=clock();
          printf("Start Time = %ld\n",start);
          printf("End Time = %ld\n",end);
          total_cputime=(double)(end-start);
          printf("Total CPU Time = %f\n",total_cputime);
          total_cputime=((double)(end-
start))/CLOCKS_PER_SEC;
          printf("Total CPU Time in Sec. =
%f\n",total_cputime);
      }
  }
  printf("\n");

  ///FOR AVERAGE CASE
  int c3=rand()%n;
  start=clock();
```

```
    for(int i=0; i<n; i++){
            if(a[i]==c3){
                    end=clock();
                    printf("Start Time = %ld\n",start);
                    printf("End Time = %ld\n",end);
                    total_cputime=(double)(end-start);
                    printf("Total CPU Time = %f\n",total_cputime);
                    total_cputime=((double)(end-
start))/CLOCKS_PER_SEC;
                    printf("Total CPU Time in Sec. =
%f\n",total_cputime);
            }
    }
    printf("\n");
}
```

Output

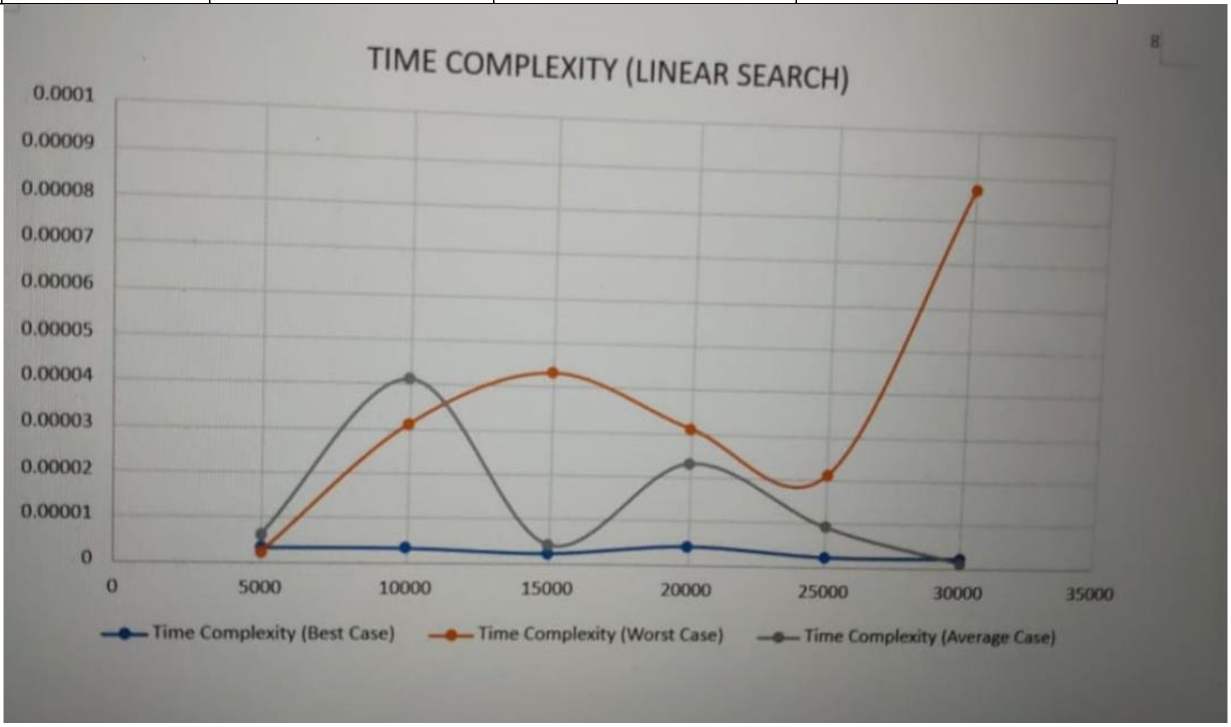


```
Start Time = 959
End Time = 961
Total CPU Time = 2.000000
Total CPU Time in Sec. = 0.000002

Start Time = 1195
End Time = 1317
Total CPU Time = 122.000000
Total CPU Time in Sec. = 0.000122

Start Time = 1348
End Time = 1466
Total CPU Time = 118.000000
Total CPU Time in Sec. = 0.000118
```

Draw the graph as the time found in each case.

Sl No.	No. of element	Time Complexity (Best Case)	Time Complexity (Worst Case)	Time Complexity (Average Case)
1	5000	0.000002	0.000001	0.000001
2	10000	0.000001	0.000001	0.000001
3	15000	0.000001	0.000001	0.000002
4	20000	0.000004	0.000002	0.000002

5	**25000**	**0.000005**	**0.000001**	**0.000002**
6	**30000**	**0.000001**	**0.000002**	**0.000001**



4. WAP in C to store 1 million integers in an array. To search an element in that array and find out its time complexity using binary search (best, worst, and average).

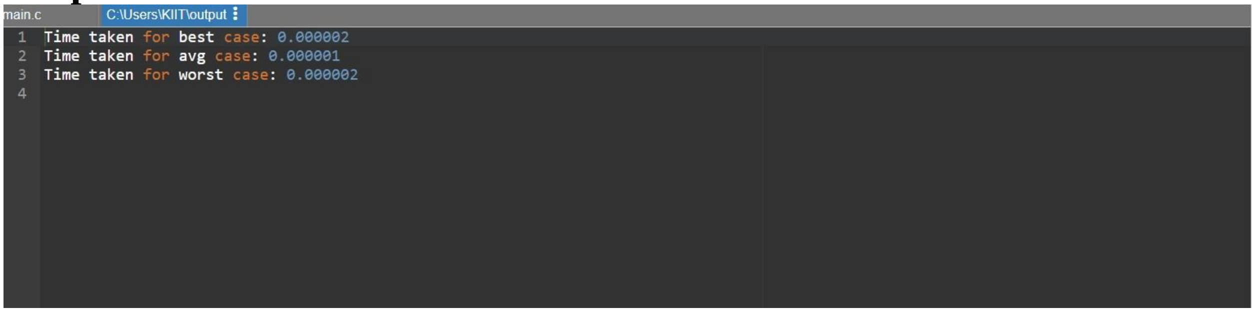
Program

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


#define sf(x)            scanf("%d", &x)
#define pf(x)            printf("%d ", x)
#define pfn(x)           printf("%d\n", x)
#define pfc(x)           printf("%d, ", x)
#define f(i,x,y)         for(int i = x; i < y; i++)
#define fi(i,x,y,inc)    for(int i = x; i < y; i += inc)
#define rf(i,x,y)        for(int i = x; i >= y; i--)


void c_() {

#ifndef ONLINE_JUDGE
    freopen("C:\\Users\\KIIT\\input", "r", stdin);
```

```
	freopen("C:\\Users\\KIIT\\output", "w", stdout);
#endif
}

int main() {

	c_();

	/* *********************** Your Main function Code
Below ************************* */

	int n = 100000;
	int arr[n];

	f(i, 0, n) {
		//arr[i] = 1 + rand() % 100;
		arr[i] = i + 1;
	}


	int best = arr[(n - 1) / 2];
	int worst = arr[1];
	int avg = arr[n / 16];
	time_t strt, end;

	int lo = 0, hi = n - 1;

	strt = clock();
	while (lo < hi)
	{
		int mid = (lo + hi) / 2;

		if (arr[mid] == best) {
			end = clock();
			double t = end - strt;
			printf("Time taken for best case: %f\n", (t /
CLOCKS_PER_SEC));
			break;
		}

		if (arr[mid] > best)
		{
			hi = mid;
```

```
        }
        else
        {
            lo = mid + 1;
        }
    }


    lo = 0, hi = n - 1;

    strt = clock();
    while (lo < hi)
    {
        int mid = (lo + hi) / 2;

        if (arr[mid] == avg) {
            end = clock();
            double t = end - strt;
            printf("Time taken for avg case: %f\n", (t /
CLOCKS_PER_SEC));
            break;
        }

        if (arr[mid] > avg)
        {
            hi = mid;
        }
        else
        {
            lo = mid + 1;
        }
    }


    lo = 0, hi = n - 1;

    strt = clock();
    while (lo < hi)
    {
        int mid = (lo + hi) / 2;

        if (arr[mid] == worst) {
            end = clock();
            double t = end - strt;
            printf("Time taken for worst case: %f\n", (t / CLOCKS_PER_SEC));
            break;
        }

        if (arr[mid] > worst)
        {
            hi = mid;
        }
        else
        {
            lo = mid + 1;
        }
    }
return 0;
}
```

Output



```
Time taken for best case: 0.000002
Time taken for avg case: 0.000001
Time taken for worst case: 0.000002
```

Draw the graph as the time found in each case.

Sl No.	No. of element	Time Complexity (Best Case)	Time Complexity (Worst Case)	Time Complexity (Average Case)
1	5000	0.000003	0.000001	0.000001
2	10000	0.000001	0.000001	0.000001
3	15000	0.00004	0.000001	0.000001
4	20000	0.000002	0.000002	0.000001
5	25000	0.000002	0.000001	0.000002
6	30000	0.000002	0.000001	0.000002