**Student Name:-Prameet Upadhyay**

**Student Roll No.:- 1905692**

**Algorithm Lab. Class Assignment-2**

**CSE Group 1**

**Date: - 16th July 2021**

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**

#include<stdio.h>

int swap(int \*x, int \*y)

{

int temp = \*x;

    \*x = \*y;

    \*y = temp;

}

int main()

{

    int a,b,c;

    a = 1 , b = 2, c = 3 ;

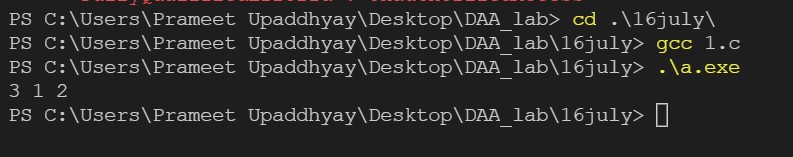
    swap(&a,&b);

    swap(&a,&c);

    printf("%d %d %d",a,b,c);

}

**Output**

****

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

**Program**

#include<stdio.h>

int non\_zero(int mat[3][3])

{

    int cnt = 0;

    for(int i=0;i<3;i++)

    {

        for(int j=0;j<3;j++)

        {

            if(mat[i][j] != 0)

                cnt++;

        }

    }

    return cnt;

}

int sum\_above\_diag(int mat[3][3])

{

    int sum = 0;

    for(int i=0;i<3;i++)

    {

        for(int j=0;j<3;j++)

        {

            if(j > i)

                sum = sum + mat[i][j];

        }

    }

    return sum;

}

int elements\_below\_diag(int mat[3][3])

{

    printf("\n\nThe elemnets below diagonal are:  ");

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            if(j < i)

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

        }

    }

}

int product\_of\_diag(int mat[3][3]){

    int product = 1;

    for(int i=0;i<3;i++){

        for(int j=0;j<3;j++){

            if(j == i){

               product = product\*mat[i][j];

            }

        }

    }

    return product;

}

int main()

{

    int mat[3][3];

    for(int i=0;i<3;i++)

    {

        for(int j=0;j<3;j++)

        {

            scanf("%d",&mat[i][j]);

        }

    }

    printf("Number of nonzero elements  : %d ", non\_zero(mat));

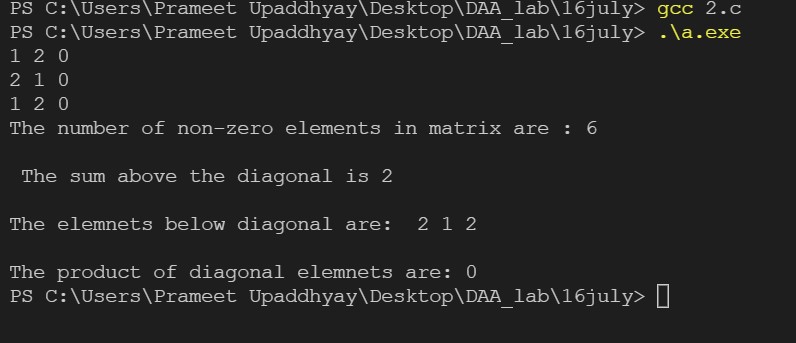
    printf("\n\n Sum above the diagonal is %d",sum\_above\_diag(mat));

    elements\_below\_diag(mat);

    printf("\nProduct of diagonal elemnets: %d",product\_of\_diag(mat));

}

**Output**

****

1. 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 <time.h>

#include <stdlib.h>

int main() {

    int n = 1000000;

    int arr[n];

    for(int i=0;i<n;i++) {

        arr[i] = i + 1;

    }

    int best = arr[0];

    int worst = arr[n - 1];

    int avg = arr[n / 2];

    time\_t strt, end;

    strt = clock();

    for(int i=0;i<n;i++) {

        if (best == arr[i]) {

            end = clock();

            double t = end - strt;

            printf("Time taken for best case: %f\n", (t / CLOCKS\_PER\_SEC));

            break;

        }

    }

    strt = clock();

    for(int i=0;i<n;i++) {

        if (avg == arr[i]) {

            end = clock();

            double t = end - strt;

            printf("Time taken for avg case: %f\n", (t / CLOCKS\_PER\_SEC));

            break;

        }

    }

    strt = clock();

    for(int i=0;i<n;i++) {

        if (worst == arr[i]) {

            end = clock();

            double t = end - strt;

            printf("Time taken for worst case: %f\n", (t / CLOCKS\_PER\_SEC));

            break;

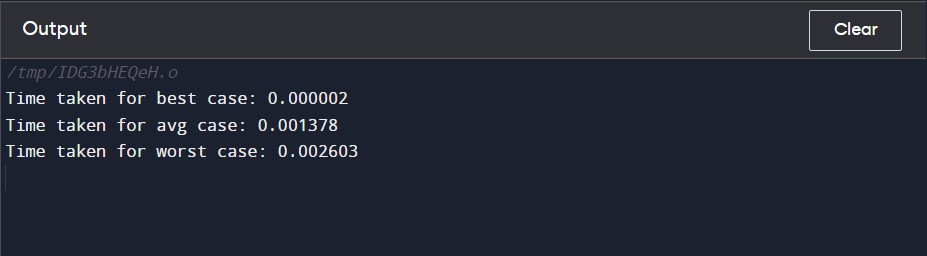
        }

    }

    return 0;

}

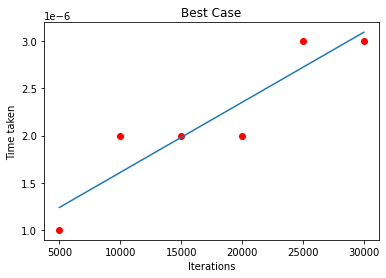
**Output**

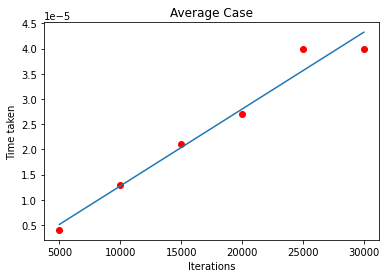
****

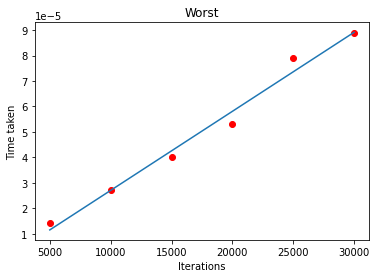
**Draw the graph as the time found in each case.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl No.** | **No. of element** | **Time Complexity ( Best Case)** | **Time Complexity (Worst Case)** | **Time Complexity**  **(Average Case)** |
| **1** | **5000** | **0.000001** | **0.000013** | **0.000004** |
| **2** | **10000** | **0.000001** | **0.000029** | **0.000014** |
| **3** | **15000** | **0.000002** | **0.000038** | **0.000020** |
| **4** | **20000** | **0.000002** | **0.000050** | **0.000025** |
| **5** | **25000** | **0.000002** | **0.000080** | **0.000038** |
| **6** | **30000** | **0.000003** | **0.000088** | **0.000040** |

**GRAPHS**

****

****

****

1. 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>

int main()

{

    int n = 100000;

    int arr[n];

    for(int i=0;i<n;i++) {

        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;

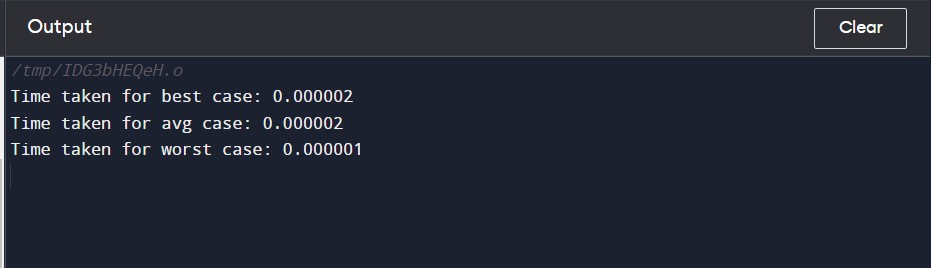
        }

    }

    return 0;

}

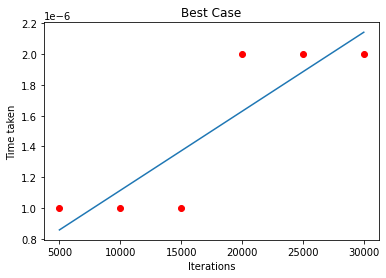
**Output**

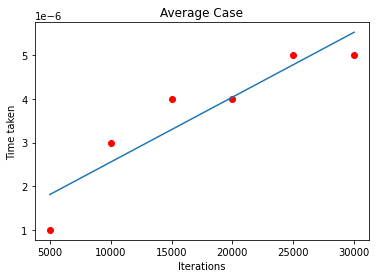
****

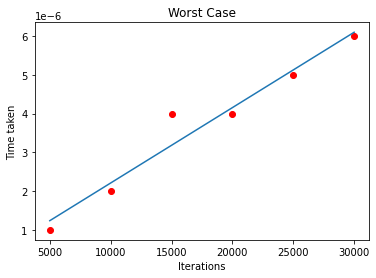
**Draw the graph as the time found in each case.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl No.** | **No. of element** | **Time Complexity ( Best Case)** | **Time Complexity (Worst Case)** | **Time Complexity**  **(Average Case)** |
| **1** | **5000** | **0.000001** | **0.000001** | **0.000001** |
| **2** | **10000** | **0.000001** | **0.000002** | **0.000002** |
| **3** | **15000** | **0.000001** | **0.000003** | **0.000002** |
| **4** | **20000** | **0.000001** | **0.000003** | **0.000003** |
| **5** | **25000** | **0.000002** | **0.000005** | **0.000004** |
| **6** | **30000** | **0.000002** | **0.000007** | **0.000005** |

**GRAPHS**

****

****

****