CHAUDHARY HAMDAN

1905387

**Algorithm Lab. Class Assignment-2**

**CSE Group 1**

**Date: - 16th July 2021**

Q1.

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.

Code:

#include <stdio.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

}

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

int main() {

c\_();

int a, b, c;

sf(a); sf(b); sf(c);

pfc(a); pfc(b); pfn(c);

swap(&a, &b);

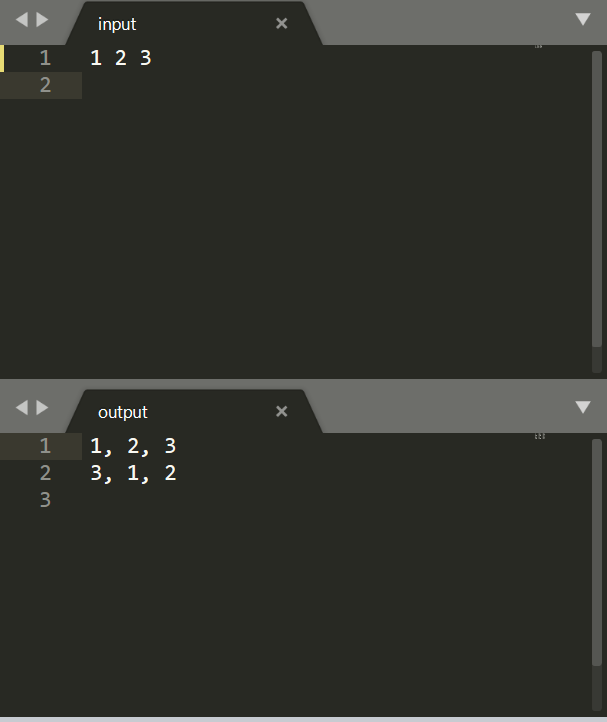
swap(&a, &c);

pfc(a); pfc(b); pfn(c);

return 0;

}

Output:



Q2.

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.

Code:

#include <stdio.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\_();

int n;

sf(n);

int non\_zero = 0;

int sum\_above = 0;

int pro\_diag = 1;

int mat[n][n];

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

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

sf(mat[i][j]);

if (mat[i][j]) {

non\_zero++;

}

if (i == j) {

pro\_diag \*= mat[i][j];

}

if (j > i) {

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

}

}

}

printf("Number of non zero elements: %d\n", non\_zero);

printf("Sum of elements above diagonal: %d\n", sum\_above);

printf("Product of diagonal elements: %d\n", pro\_diag);

printf("Minor diagonal elements: ");

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

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

sf(mat[i][j]);

if (i + j == n - 1) {

pfc(mat[i][j]);

}

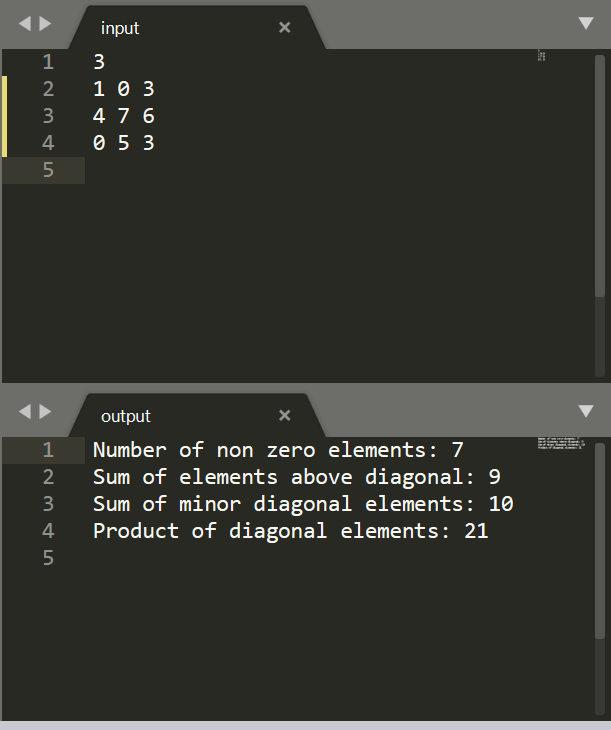
}

}

return 0;

}

Output:



Q3.

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).

Code:

#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\_();

int n = 100000;

int arr[n];

f(i, 0, n) {

//arr[i] = 1 + rand() % 100;

arr[i] = i + 1;

}

int best = arr[0];

int worst = arr[n - 1];

int avg = arr[n / 2];

time\_t strt, end;

strt = clock();

f(i, 0, n) {

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();

f(i, 0, n) {

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();

f(i, 0, n) {

if (worst == arr[i]) {

end = clock();

double t = end - strt;

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

break;

}

}

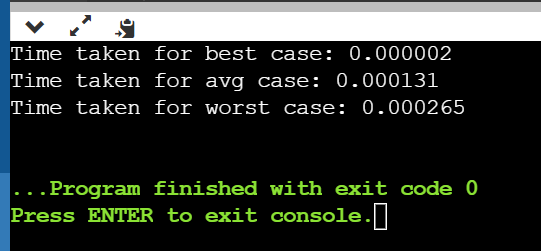
return 0;

}

return 0;

}

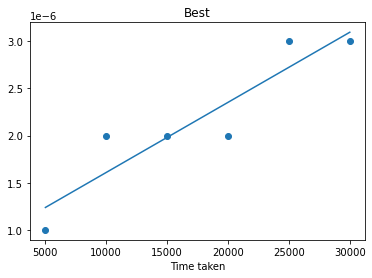
Output:

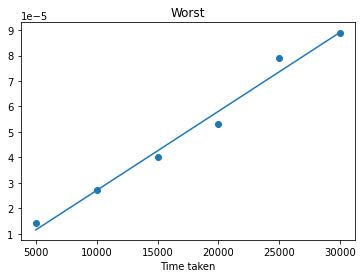


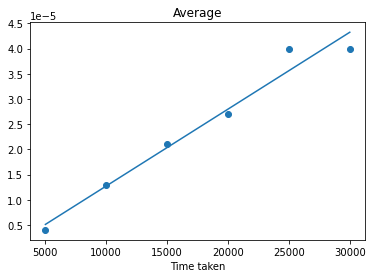
Tables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl No.** | **No. of element** | **Time Complexity ( Best Case)** | **Time Complexity (Worst Case)** | **Time Complexity**  **(Average Case)** |
| **1** | **5000** | **0.000001** | **0.000014** | **0.000004** |
| **2** | **10000** | **0.000002** | **0.000027** | **0.000013** |
| **3** | **15000** | **0.000002** | **0.000040** | **0.000021** |
| **4** | **20000** | **0.000002** | **0.000053** | **0.000027** |
| **5** | **25000** | **0.000003** | **0.000079** | **0.000040** |
| **6** | **30000** | **0.000003** | **0.000089** | **0.000040** |

Graphs:







Q4.

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).

Code:

#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\_();

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:

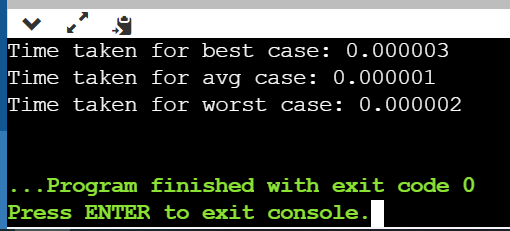


Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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** | **0.000003** |
| **3** | **15000** | **0.000001** | **0.000004** | **0.000004** |
| **4** | **20000** | **0.000002** | **0.000004** | **0.000004** |
| **5** | **25000** | **0.000002** | **0.000005** | **0.000005** |
| **6** | **30000** | **0.000002** | **0.000006** | **0.000005** |

Graph:

