

**Project: Array Operations Efficiency**

**Course Name: Algorithms**

**Course Code: CSE246**

**Section: 08**

**Semester: Fall 24**

**Submitted to:**

Amit Mandal

Lecturer

Department of Computer Science and Engineering

**Submitted by:**

Name: Samia Jahan Mariam

ID: 2022-1-60-376

Name: Sayma Sultana

ID: 2021-3-60-105

Name: Mariya Mehajabin

ID: 2022-1-60-363

Date of Submission: 02.02.2025

**Array Operations Efficiency**

**The Algorithm used are:**

**Linear Search Algorithm**: Searching for a key in an array.

**Finding Maximum Element Algorithm**: Finding the highest value in an array.

**Sum of Array Elements Algorithm**: Calculating the sum of all elements in the array.

**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function prototypes

int linearSearch(int arr[], int n, int key);

int findMax(int arr[], int n);

int sumArray(int arr[], int n);

// High-resolution timer for Linux

#ifdef \_\_linux\_\_

#include <sys/time.h>

double get\_time() {

struct timespec ts;

clock\_gettime(CLOCK\_MONOTONIC, &ts);

return (double)ts.tv\_sec + (double)ts.tv\_nsec / 1e9;

}

#else

// Fallback to clock() for other systems

double get\_time() {

return (double)clock() / CLOCKS\_PER\_SEC;

}

#endif

int main() {

int n, i, key;

int j; // Declare loop variable at the beginning (C89/C90 compatibility)

double start, end;

double cpu\_time\_used;

// Array sizes to test

int sizes[] = { 1000000, 10000000,100000000}; // Larger input sizes

int num\_sizes = sizeof(sizes) / sizeof(sizes[0]); // Number of sizes

// Arrays to store runtime results for each algorithm

double linear\_search\_times[num\_sizes];

double max\_element\_times[num\_sizes];

double sum\_array\_times[num\_sizes];

// Number of repetitions for linear search

int repetitions = 1000000; // Repeat linear search 1,000,000 times

// Loop through each input size

for (j = 0; j < num\_sizes; j++) { // Use pre-declared 'j'

n = sizes[j]; // Current input size

// Dynamically allocate memory for the array

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

if (arr == NULL) {

printf("Memory allocation failed for size %d!\n", n);

return 1;

}

// Generate random input array

srand(time(0)); // Seed for random number generation

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

arr[i] = rand() % 1000; // Random numbers between 0 and 999

}

// Key for linear search (randomly selected from the array)

key = arr[rand() % n];

printf("Key to search: %d\n", key);

// Measure Linear Search time (repeat many times)

start = get\_time();

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

linearSearch(arr, n, key);

}

end = get\_time();

cpu\_time\_used = (end - start) / repetitions; // Average time per search

linear\_search\_times[j] = cpu\_time\_used;

// Display result of linear search

int result = linearSearch(arr, n, key);

if (result != -1) {

printf("Key found at index: %d\n", result);

} else {

printf("Key not found\n");

}

// Measure Finding Maximum Element time

start = get\_time();

findMax(arr, n);

end = get\_time();

cpu\_time\_used = end - start;

max\_element\_times[j] = cpu\_time\_used;

// Display maximum element

int max = findMax(arr, n);

printf("Maximum element in the array: %d\n", max);

// Measure Sum of Array Elements time

start = get\_time();

sumArray(arr, n);

end = get\_time();

cpu\_time\_used = end - start;

sum\_array\_times[j] = cpu\_time\_used;

// Display sum of array elements

int sum = sumArray(arr, n);

printf("Sum of array elements: %d\n\n", sum);

// Free dynamically allocated memory

free(arr);

}

// Print results in a table format

printf("Input Size\tLinear Search\tFind Max\tSum Array\n");

printf("--------------------------------------------------------\n");

for (j = 0; j < num\_sizes; j++) { // Use pre-declared 'j'

printf("%d\t\t%f\t%f\t%f\n", sizes[j], linear\_search\_times[j], max\_element\_times[j], sum\_array\_times[j]);

}

return 0;

}

// Linear Search Algorithm

int linearSearch(int arr[], int n, int key) {

int i; // Declare loop variable at the beginning (C89/C90 compatibility)

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

if (arr[i] == key) {

return i;

}

}

return -1;

}

// Finding Maximum Element Algorithm

int findMax(int arr[], int n) {

int i; // Declare loop variable at the beginning (C89/C90 compatibility)

int max = arr[0];

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

if (arr[i] > max) {

max = arr[i];

}

}

return max;

}

// Sum of Array Elements Algorithm

int sumArray(int arr[], int n) {

int i;

int sum = 0;

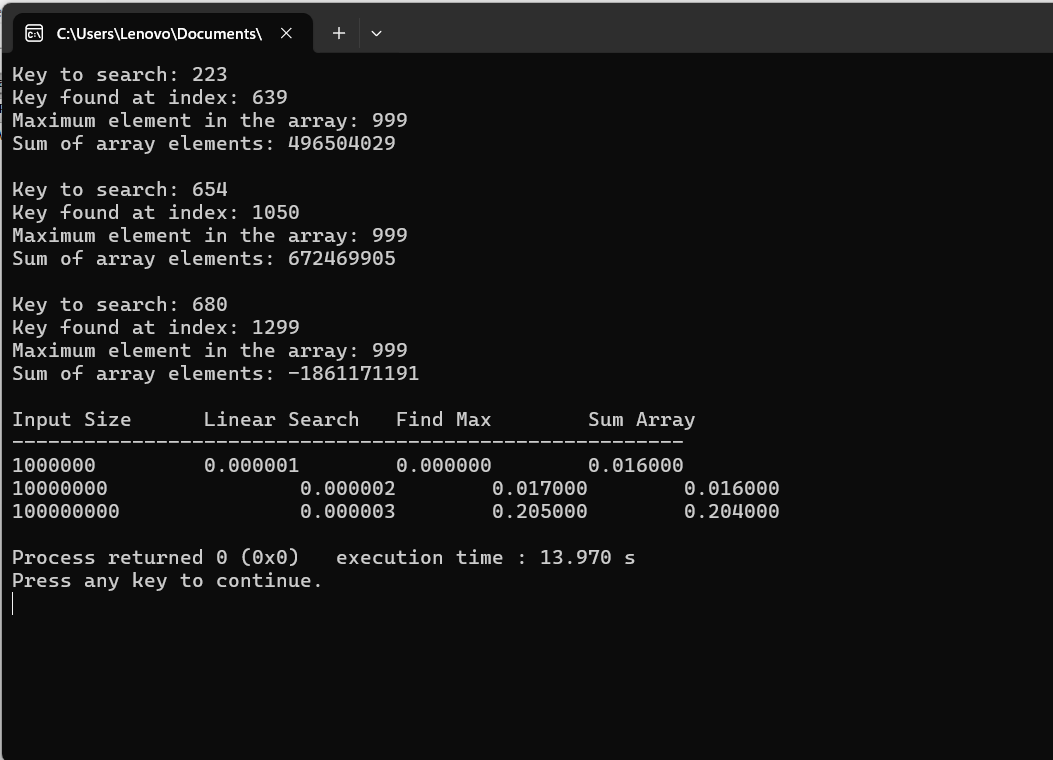
for (i = 0; i < n; i++) {

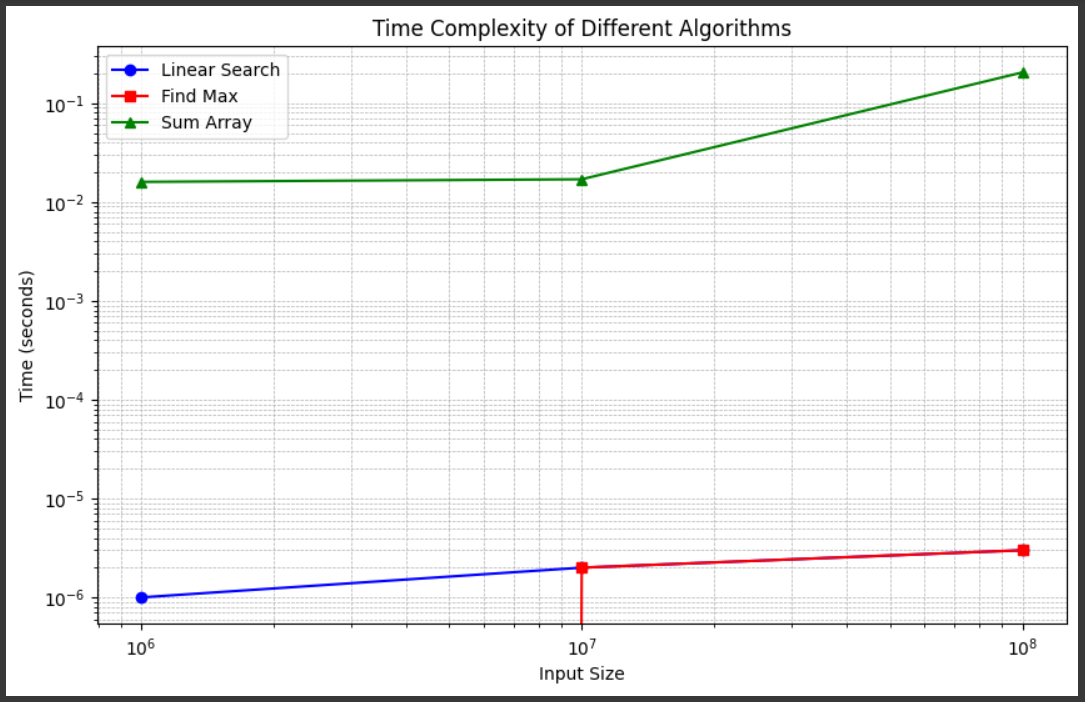
sum += arr[i];

}

return sum;

}

**Output:**

**Graph :**

This program tests the performance of three algorithms (linear search, find max, sum array) on arrays of varying sizes and reports the execution time for each operation. The get\_time() function is used to measure the execution time, and the results are displayed in a tabular format at the end.