CCC 204 Data Structures and Algorithms LABORATORY
REPORT :
LAB 11# - Searching Algorithms

Samano, Rajel Johann
Information Technology Program, CABECS
Colegio San Agustin – Bacolod

I. INTRODUCTION

For Laboratory Activity Number Eleven is to follow the objectives of the exercises and answer questions. The Three main objectives as follows:

- Identify and describe searching algorithms
- Modify and run search algorithms according to requirements
- Evaluate performance of searching algorithms

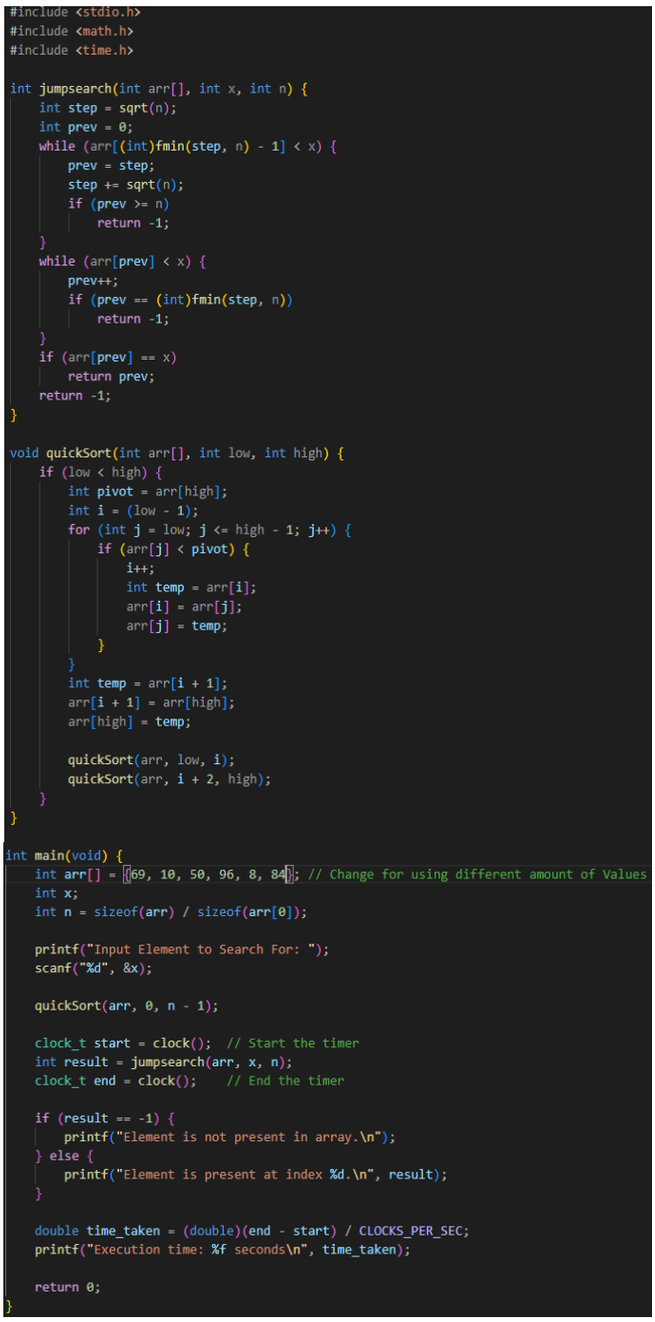
II. IMPLEMENTATION / APPROACH

```c
#include <stdio.h>

int search(int arr[], int N, int x) {
    for (int i = 0; i < N; i++)
        if (arr[i] == x)
            return i;
    return -1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        quickSort(arr, low, i);
        quickSort(arr, i + 2, high);
    }
}

int main(void) {
    int arr[] = {2, 3, 4, 10, 40}; // Change for using different amount of Values
    int x;
    int N = sizeof(arr) / sizeof(arr[0]);

    printf("Input Element to Search For: ");
    scanf("%d", &x);

    quickSort(arr, 0, N - 1);

    int result = search(arr, N, x);
    (result == -1)
        ? printf("Element is not present in array.\n")
        : printf("Element is present at index %d.\n", result);

    return 0;
}
```

Figure 1. Linear_Search.c

```c
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x) {
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
            return m;
        if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        quickSort(arr, low, i);
        quickSort(arr, i + 2, high);
    }
}
int main(void) {
    int arr[] = {2, 3, 4, 10, 40};  // Change for using different amount of Values
    int n = sizeof(arr) / sizeof(arr[0]);
    int x;

    printf("Input Element to Search For: ");
    scanf("%d", &x);

    quickSort(arr, 0, n - 1);

    int result = binarySearch(arr, 0, n - 1, x);
    (result == -1) ? printf("Element is not present in array.\n")
                   : printf("Element is present at index %d\n", result);

    return 0;
}
```

Figure 2. Binary_Search.c



```c
#include <stdio.h>
#include <math.h>
#include <time.h>

int jumpsearch(int arr[], int x, int n) {
    int step = sqrt(n);
    int prev = 0;
    while (arr[(int)fmin(step, n) - 1] < x) {
        prev = step;
        step += sqrt(n);
        if (prev >= n)
            return -1;
    }
    while (arr[prev] < x) {
        prev++;
        if (prev == (int)fmin(step, n))
            return -1;
    }
    if (arr[prev] == x)
        return prev;
    return -1;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pivot = arr[high];
        int i = (low - 1);
        for (int j = low; j <= high - 1; j++) {
            if (arr[j] < pivot) {
                i++;
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
        int temp = arr[i + 1];
        arr[i + 1] = arr[high];
        arr[high] = temp;

        quickSort(arr, low, i);
        quickSort(arr, i + 2, high);
    }
}

int main(void) {
    int arr[] = {69, 10, 50, 96, 8, 84}; // Change for using different amount of Values
    int x;
    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Input Element to Search For: ");
    scanf("%d", &x);

    quickSort(arr, 0, n - 1);

    clock_t start = clock();  // Start the timer
    int result = jumpsearch(arr, x, n);
    clock_t end = clock();    // End the timer

    if (result == -1) {
        printf("Element is not present in array.\n");
    } else {
        printf("Element is present at index %d.\n", result);
    }

    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Execution time: %f seconds\n", time_taken);

    return 0;
}
```

Figure 2. Jump_Search.c

III.EXPERIMENTAL FINDINGS / DISCUSSIONS

Searching Methods	*5 Values*	*Average*	*10 Values*	*Average*	*20 Values*	*Average*
Linear Search	*0.65sec* *0.86sec* *0.71sec*	*1.75sec*	*0.64sec* *0.77sec* *0.83sec*	*1.69sec*	*0.66sec* *0.66sec* *0.75sec*	*1.57sec*
Binary Search	*0.85sec* *0.66sec* *0.65sec*	*1.73sec*	*0.64sec* *0.80sec* *0.82sec*	*1.71sec*	*0.63sec* *0.89sec* *0.66sec*	*1.74sec*
Jump Search	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*
Jump Search using Time Library	*0.00* *0.00* *0.00*	*0.00*	*0.00* *0.00* *0.00*	*0.00*	*0.00* *0.00* *0.00*	*0.00*

***Disclaimer: Jump Search Does Not work on JDoodle so I can't check for execution time I have to implement a clock using the time library for C. I don't know if it is even accurate.**

IV. CONCLUSIONS

To conclude the eleventh laboratory activity which is searching algorithms I would say that linear is the fastest due to starting from the beginning and stops at the end of the array. But I jumping might as well be fast but due to not having any idea of its execution time then it would just be in the neutral as when I was running it on VScode it is fast as well like almost the same as linear search.

V. References:

https://www.simplilearn.com/tutorials/c-tutorial/program-for-linear-search-in-c#

https://www.geeksforgeeks.org/linear-search/

https://www.geeksforgeeks.org/binary-search/

https://www.geeksforgeeks.org/jump-search/