**EXPERIMENT1: Title: BRUTE FORCE TECHNIQUES**

1. Sort a given set of elements using bubble and selection sort and hence find the time required to sort elements .

Code:

#include<stdio.h>

#include<time.h>

void bubblesort(int arr[],int n)//fun to perform bubble sort

{

    for(int i=n-1;i>=0;i--)

    {

        for(int j=0;j<=i-1;j++)

        {

            if (arr[j]>arr[j+1])

            {//swap elements after comparison if needed

                int temp=arr[j+1];

                arr[j+1]=arr[j];

                arr[j]=temp;

            }

        }

    }

}

void selectionsort(int arr[],int n) //fun to perform selection sort

{

    for (int i=0;i<n-1;i++)

    {

        int mini=i;

        for (int j=i+1;j<n;j++)

        {   // find index of the minimum element

            if(arr[j]<arr[mini])

            {

                mini=j;

            }

        }

        //swap the minimum elemnt with first element

        int temp=arr[mini];

        arr[mini]=arr[i];

        arr[i]=temp;

    }

}

int main()

{

    double time\_spend = 0.0;

    clock\_t begin = clock();

    int arr[] = {32, 13, 56, 35, 24, 76, 34, 23,67,43,23,20};

    int n = sizeof(arr) / sizeof(arr[0]);

    printf("Unsorted array: ");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    // Perform Bubble Sort

    bubblesort(arr, n);

    printf("Sorted array using bubble sort: ");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

    // Perform Selection Sort

    selectionsort(arr, n);

    printf("Sorted array using selection sort: ");

    for (int i = 0; i < n; i++) {

        printf("%d ", arr[i]);

    }

    printf("\n");

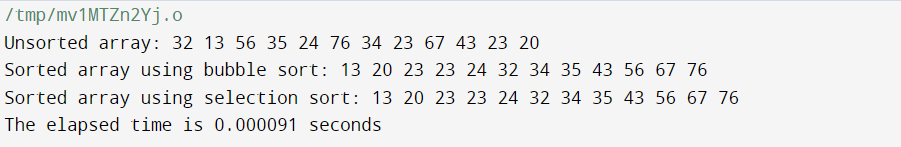
    clock\_t end = clock();

    time\_spend += (double)(end - begin) / CLOCKS\_PER\_SEC;

    printf("The elapsed time is %f seconds", time\_spend);

    return 0;

}



1. Perform linear search and find the time required to search an element

Code:

#include<stdio.h>

#include<time.h>

int main()

{

    double time\_spent=0.0;

    clock\_t begin=clock();

    int i,item,n,q;

    int a[100]={3,6,7,8,3,5,2};

    printf("enter item to serach: ");

    scanf("%d",&item);

    for (i = 0; i < q; i++) {

        if (item == a[i]) {

            // Element found

            printf("Element is present\n");

            break; // Exit the loop if the element is found

        }

    }

    // Check if the loop completed without finding the element

    if (i == q) {

        printf("Element is not present\n");

    }

    // Calculate and display the elapsed time

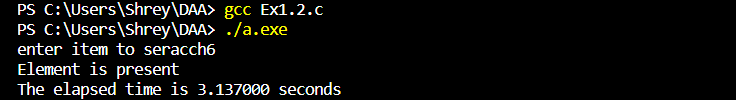
    clock\_t end = clock();

    time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

    printf("The elapsed time is %f seconds", time\_spent);

    return 0;

}



1. Given a string called TEXT with ‘n’ characters and another string called PATTERN with ‘m’ characters (m<=n) .Write a program which implements brute force string matching to search for a given pattern in the text. If the pattern is present then find the position of first occurrences of Pattern in that Text.

Code:

#include <stdio.h>

#include <string.h>

#include <time.h>

// Function to perform pattern matching in a given text

int txtmatch( const char\* text,const char\* pattern) {

    int n = strlen(text);

    int m = strlen(pattern);

    // Iterate through the text for potential pattern matches

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

        int j;

        // Compare characters of text and pattern

        for (j = 0; j < m; j++) {

            if (text[i + j] != pattern[j]) {

                break; // Exit loop if mismatch is found

            }

        }

        // If entire pattern is matched, return the position

        if (j == m) {

            return i;

        }

    }

    // Return -1 if pattern is not found in the text

    return -1;}

int main() {

    double time\_spend = 0.0;

    clock\_t begin = clock();

    const char text[200];

    const char pattern[200];

    // Input: Get the text from the user

    printf("Enter the text: ");

    gets(text);

    // Input: Get the pattern to search from the user

    printf("Enter the text to search: ");

    gets(pattern);

    // Call the text matching function

    int position = txtmatch(text, pattern);

    // Display the result of the pattern matching

    if (position != -1) {

        printf("Pattern found at position: %d\n", position);

    } else {

        printf("Pattern not found in the text.\n");

    }

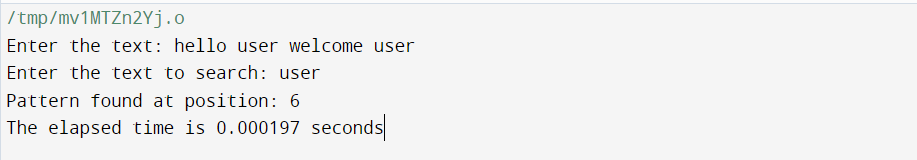
    clock\_t end = clock();

    time\_spend += (double)(end - begin) / CLOCKS\_PER\_SEC;

    printf("The elapsed time is %f seconds", time\_spend);

    return 0;

}



**EXPERIMENT-2: Title: DIVIDE AND CONQUER-I**

1. Implement Binary search and linear search and determine the time required to search an element. Repeat the experiment for different values of n, the number of elements in the list to be searched and plot a graph of the time taken versus n.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Linear search function

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

    for (int i = 0; i < n; i++) {

        if (arr[i] == x) {

            return i; // Element found, return its index

        }

    }

    return -1; // Element not found

}

// Binary search function (array must be sorted)

int binarySearch(int arr[], int left, int right, int x) {

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (arr[mid] == x) {

            return mid; // Element found, return its index

        }

        if (arr[mid] < x) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1; // Element not found

}

int main() {

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

    int n\_values[] = {100, 500, 1000, 5000, 10000}; // Different values of n

    int num\_values = sizeof(n\_values) / sizeof(n\_values[0]);

    for (int k = 0; k < num\_values; k++) {

        int n = n\_values[k];

        int arr[n];

        int x = rand() % (n \* 2); //give random num to search allow duplicates

        for (int i = 0; i < n; i++) {

            arr[i] = rand() % (n \* 2); //give random values to the array

        }

        clock\_t start, end;

        // Linear Search

        start = clock();

        int linear\_result = linearSearch(arr, n, x);

        end = clock();

        double linear\_search\_time = (double)(end - start) / CLOCKS\_PER\_SEC;

        // Binary Search (assuming the array is sorted)

        start = clock();

        int binary\_result = binarySearch(arr, 0, n - 1, x);

        end = clock();

        double binary\_search\_time = (double)(end - start) / CLOCKS\_PER\_SEC;

        printf("n = %d:\n", n);

        printf("Element to search for: %d\n", x);

        printf("Linear Search Result: %s\n", linear\_result != -1 ? "Found" : "Not Found");

        printf("Linear Search Time: %f seconds\n", linear\_search\_time);

        printf("Binary Search Result: %s\n", binary\_result != -1 ? "Found" : "Not Found");

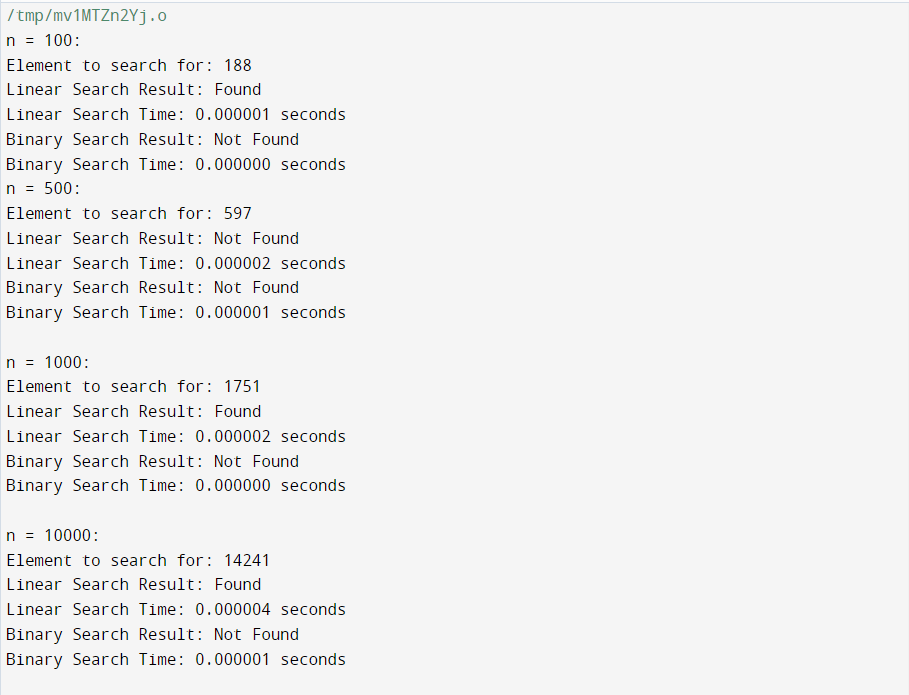
        printf("Binary Search Time: %f seconds\n", binary\_search\_time);

        printf("\n");

    }

    return 0;

}



1. Search a elements using the Binary search method and determine the time required to search the element. Repeat the experiment for different values of n, to search for the element in the list and plot a graph of the time taken versus n.

Code:

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

// Binary search function

int binarySearch(int arr[], int n, int x) {

    int left = 0, right = n - 1;

    while (left <= right) {

        int mid = left + (right - left) / 2;

        if (arr[mid] == x) {

            return mid; // Element found, return its index

        }

        if (arr[mid] < x) {

            left = mid + 1;

        } else {

            right = mid - 1;

        }

    }

    return -1; // Element not found

}

int main()

{

    int n,item; // Size of the array and Element to search for

    printf("Enter the size of the array: ");

    scanf("%d", &n);

    int \*arr = (int \*)malloc(n \* sizeof(int)); // Dynamically allocate memory for the array

    if (arr == NULL) {

        printf("Memory allocation failed\n");

        return 1;

    }

    // Populate the sorted array

    for (int i = 0; i < n; i++) {

        arr[i] = i \* 2;

    }

    printf("Enter the element to search for: ");

    scanf("%d", &item);

    double time\_spent=0.0;

    clock\_t begin=clock();

   // Binary Search

    int binary\_result = binarySearch(arr, n, item);

    clock\_t end = clock();

    time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

    printf("The elapsed time is %f seconds", time\_spent);

    if (binary\_result != -1) {

        printf("Element %d found at index %d\n", item, binary\_result);

    } else {

        printf("Element %d not found\n", item);

    }

    free(arr);

    return 0;

}

