

|  |  |
| --- | --- |
| Submitted By  Name: Muhammd Mehedi Hasan  Reg: 23201143  Section: C  Subject: CSE 108 | Submitted To  Name: Zaima Sartaj Taheri  Lecturer  Department of Computer Science and Engineering, [University of Asia Pacific](https://uap-bd.edu/) |

**Sorting Algorithms**

**i. Bubble Sort**

**Concept:  
Bubble sort is a simple comparison-based algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted. Each pass moves the largest element to its correct position, akin to "bubbling up" the largest value.**

**Numerical Example:  
Input: [5, 3, 8, 6, 2]**

* **Pass 1: [3, 5, 6, 2, 8]**
* **Pass 2: [3, 5, 2, 6, 8]**
* **Pass 3: [3, 2, 5, 6, 8]**
* **Pass 4: [2, 3, 5, 6, 8]**

**Output: [2, 3, 5, 6, 8]**

**Code:**

#include <stdio.h>

void bubbleSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j+1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

int main() {

int arr[] = {5, 3, 8, 6, 2};

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

printf("ID: 23201143\n");

printf("Original Array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

bubbleSort(arr, n);

printf("Sorted Array (Bubble Sort): ");

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

printf("%d ", arr[i]);

}

printf("\n");

return 0;

}

Input/Output:



**ii. Selection Sort**

**Concept:**  
Selection sort divides the array into two parts: a sorted part and an unsorted part. It repeatedly selects the smallest element from the unsorted part and swaps it with the first element of the unsorted part.

**Numerical Example:**  
Input: [29, 10, 14, 37, 13]

* Pass 1: [10, 29, 14, 37, 13]
* Pass 2: [10, 13, 14, 37, 29]
* Pass 3: [10, 13, 14, 37, 29]
* Pass 4: [10, 13, 14, 29, 37]

Output: [10, 13, 14, 29, 37]

**Code:**

#include <stdio.h>

void selectionSort(int arr[], int n) {

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap the found minimum element with the first element

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

int main() {

int arr[] = {29, 10, 14, 37, 13};

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

printf("ID: 23201143\n");

printf("Original Array: ");

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

printf("%d ", arr[i]);

}

printf("\n");

selectionSort(arr, n);

printf("Sorted Array (Selection Sort): ");

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

printf("%d ", arr[i]);

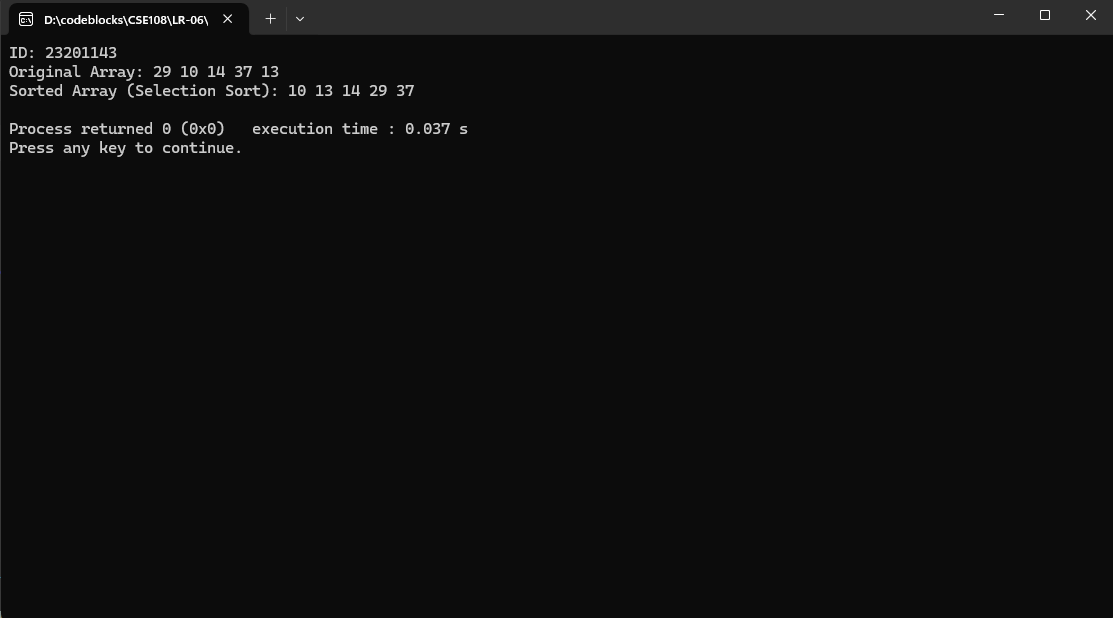
}

printf("\n");

return 0;

}

Input/Output:



**Searching Algorithms**

**i. Linear Search**

**Concept:**  
Linear search sequentially checks each element of the array until the desired element is found or the list ends.

**Numerical Example:**  
Array: [4, 2, 8, 6, 1], Search: 6

* Step 1: Compare 4 → Not found
* Step 2: Compare 2 → Not found
* Step 3: Compare 8 → Not found
* Step 4: Compare 6 → Found

**Code:**

#include <stdio.h>

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

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

if (arr[i] == key) {

return i; // Return index of key

}

}

return -1; // Return -1 if not found

}

int main() {

int arr[] = {4, 2, 8, 6, 1};

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

int key = 6;

printf("ID: 23201143\n");

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

if (result != -1) {

printf("Element %d found at index %d.\n", key, result);

} else {

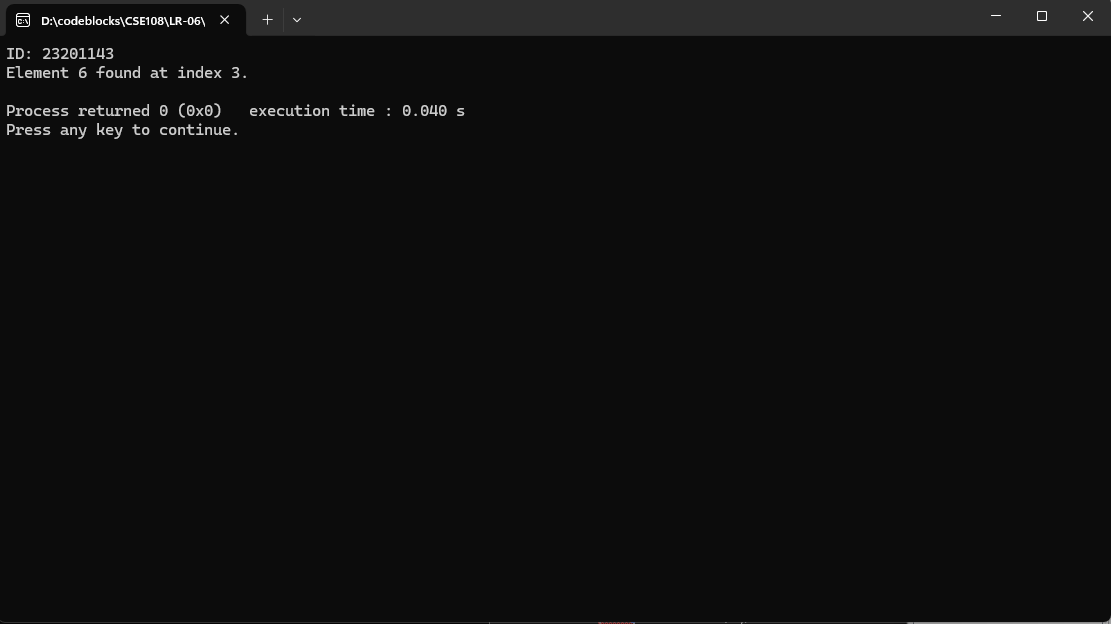
printf("Element %d not found.\n", key);

}

return 0;

}

Input/Output:



**ii. Binary Search**

**Concept:**  
Binary search divides the array into two halves and repeatedly checks the middle element. It eliminates half the elements in each step until the target is found or the search ends.

**Numerical Example:**  
Sorted Array: [1, 3, 5, 7, 9], Search: 7

* Step 1: Compare middle (5) → Go right
* Step 2: Compare middle (7) → Found

**Code:**

#include <stdio.h>

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

int left = 0, right = n - 1;

while (left <= right) {

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

if (arr[mid] == key) {

return mid; // Key found

} else if (arr[mid] < key) {

left = mid + 1; // Search right half

} else {

right = mid - 1; // Search left half

}

}

return -1; // Key not found

}

int main() {

int arr[] = {1, 3, 5, 7, 9};

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

int key = 7;

printf("ID: 23201143\n");

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

if (result != -1) {

printf("Element %d found at index %d.\n", key, result);

} else {

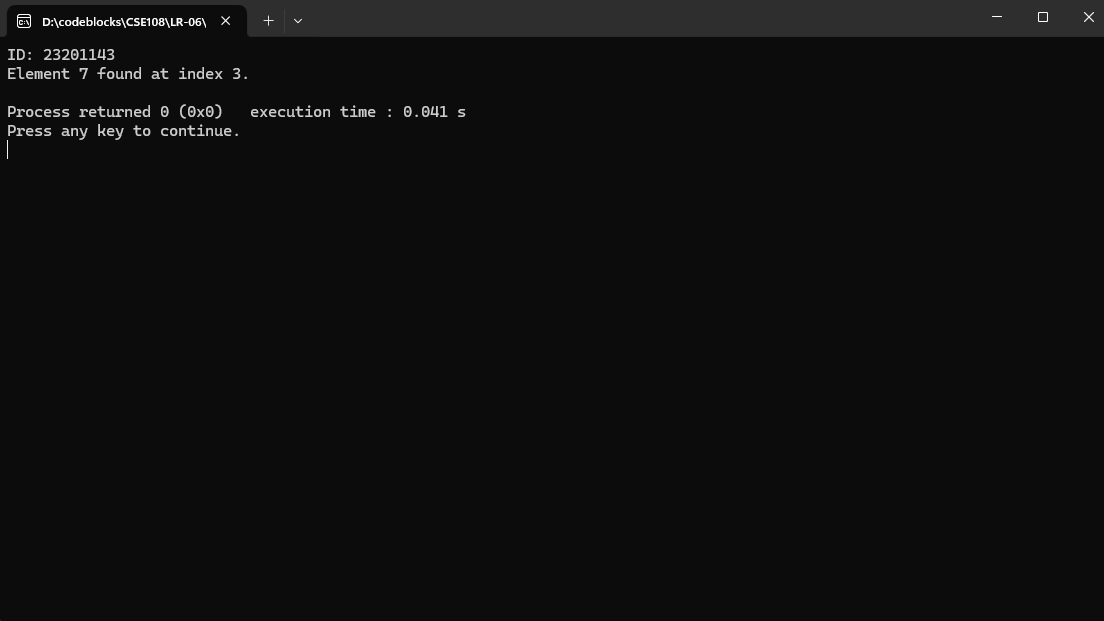
printf("Element %d not found.\n", key);

}

return 0;

}

Input/Output:



**Discussion**

**Differences Between Bubble Sort and Selection Sort**

1. **Approach**:
   * Bubble Sort repeatedly swaps adjacent elements.
   * Selection Sort selects the smallest element in the unsorted part and swaps it with the current position.
2. **Efficiency**:
   * Bubble Sort may require more swaps.
   * Selection Sort minimizes swaps, performing at most n-1 swaps.
3. **Performance**:
   * Both are O(n2)O(n^2)O(n2) in worst case, but Selection Sort often performs fewer operations.

**Binary Search vs. Linear Search**

1. **Steps**:
   * Linear Search checks every element sequentially.
   * Binary Search divides the array and eliminates half in each step.
2. **Efficiency**:
   * Linear Search: O(n)O(n)O(n) for nnn elements.
   * Binary Search: O(log⁡n)O(\log n)O(logn), much faster for large datasets.
3. **Requirement**:
   * Binary Search requires a **sorted array**, while Linear Search does not.