

LAB 04

Miss Atika Islam

Linear search and binary search are two fundamental algorithms for finding a target element in a list or array. Here’s a breakdown of each:

**1. Linear Search**

Linear search, also called sequential search, is the simplest search algorithm. It scans each element in the list sequentially from the beginning until it finds the target or reaches the end.

* **Characteristics of Linear Search:**
* **Time Complexity**: O(n)O(n)O(n), where nnn is the number of elements in the list. Each element might need to be checked in the worst case.
* **When to Use**: Ideal when the list is small or unsorted.
* **How It Works**:
  + Start from the first element and compare it with the target.
  + Move to the next element and repeat until the target is found or the end is reached.

#include <iostream>

using namespace std;

int linearSearch(int arr[], int size, int target) {

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

if (arr[i] == target) {

return i; // Target found

}

}

return -1; // Target not found

}

int main() {

int arr[] = {2, 4, 0, 1, 9};

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

int target = 1;

int result = linearSearch(arr, size, target);

if (result != -1) {

cout << "Element found at index " << result << endl;

} else {

cout << "Element not found" << endl;

}

return 0;

}

### Binary Search

Binary search is a more efficient search algorithm but requires the list to be sorted. It reduces the search space by half with each step, dividing and conquering to quickly locate the target.

#### 1- Characteristics of Binary Search:

* **Time Complexity**: O(logn), since it halves the search space each step.
* **When to Use**: Best for large, sorted lists or arrays.
* **How It Works**:
  + Start with pointers at the beginning (left) and end (right) of the list.
  + Calculate the midpoint and compare it with the target.
  + If the midpoint value is equal to the target, return the midpoint index.
  + If the target is smaller, move the end pointer just before the midpoint.
  + If the target is larger, move the start pointer just after the midpoint.
  + Repeat until the target is found or the pointers cross.

#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int target) {

int left = 0, right = size - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid; // Target found

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

left = mid + 1; // Search in the right half

} else {

right = mid - 1; // Search in the left half

}

}

return -1; // Target not found

}

int main() {

int arr[] = {1, 2, 4, 5, 9}; // Must be sorted for binary search

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

int target = 5;

int result = binarySearch(arr, size, target);

if (result != -1) {

cout << "Element found at index " << result << endl;

} else {

cout << "Element not found" << endl;

}

return 0;

}

**LAB TASKS:**

**1- Find First and Last Occurrence**:

* **Problem**: Given an array, find the first and last occurrences of a given target value.
* **Example**:

Input: arr = [2, 4, 2, 6, 2, 3, 4], target = 2

Output: First occurrence = 0, Last occurrence = 4

* **Hint**: Use linear search to iterate through the array, updating the positions when the target is found.

**2- Count Occurrences of a Number**:

* **Problem**: Given an array, count how many times a given target number appears.
* **Example**:

Input: arr = [1, 2, 2, 3, 2, 5], target = 2

Output: 3

**3- Check for Pair with Given Sum**:

* **Problem**: Given an array and a target sum, determine if any two numbers add up to the target.
* **Example**:

Input: arr = [3, 1, 4, 6, 5], target\_sum = 9

Output: True (because 4 + 5 = 9)

* **Hint**: Use nested loops or a hash table to store elements as you search.