**INSTRUCTION**

Create a simple search algorithm using the following unsorted list. Show the simulation on how they search.

1. Linear Search – element 10 should be found in the list

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 14 | 15 | 8 | 9 | 10 | 4 | 3 | 1 |

Certainly! Here's a Java version of a linear search algorithm for your unsorted list, along with a step-by-step simulation of how it searches for the element 10:

java

Copy code

public class LinearSearch {

public static int linearSearch(int[] arr, int target) {

for (int i = 0; i < arr.length; i++) {

if (arr[i] == target) {

return i; // Return the index where the target element is found

}

}

return -1; // Return -1 if the target element is not found

}

public static void main(String[] args) {

// Your unsorted list

int[] myList = {14, 15, 8, 9, 10, 4, 3, 1};

// Element to search for

int targetElement = 10;

// Perform the linear search

int index = linearSearch(myList, targetElement);

// Display the result

if (index != -1) {

System.out.println("Element " + targetElement + " found at index " + index + ".");

} else {

System.out.println("Element " + targetElement + " not found in the list.");

}

}

}

Now, let's simulate the linear search in Java step by step:

Start with the first element (14). It does not match the target (10).

Move to the next element (15). It also does not match the target.

Continue this process for each element in the list.

Element 8 - Not a match

Element 9 - Not a match

Element 10 - Match found!

The linear search algorithm found the target element (10) in the list at index 4. Therefore, the output will be:

mathematica

Copy code

Element 10 found at index 4.

This demonstrates how a linear search works in Java by checking each element one by one until it finds the target or reaches the end of the list.

1. Binary Search – elements to be found should be 30. The elements must be sorted first before the search takes place.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 5 | 25 | 6 | 8 | 30 | 20 | 21 | 15 |

Binary search is a more efficient search algorithm, but it requires that the list be sorted in ascending order before the search takes place. Here's a Java implementation of binary search for your sorted list, along with a step-by-step simulation of how it searches for the element 30:

java

Copy code

import java.util.Arrays;

public class BinarySearch {

public static int binarySearch(int[] arr, int target) {

int left = 0;

int right = arr.length - 1;

while (left <= right) {

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

if (arr[mid] == target) {

return mid; // Return the index where the target element is found

}

if (arr[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1; // Return -1 if the target element is not found

}

public static void main(String[] args) {

// Your unsorted list

int[] myList = {10, 9, 5, 25, 6, 8, 30, 20, 21, 15};

// Sort the list in ascending order

Arrays.sort(myList);

// Element to search for

int targetElement = 30;

// Perform the binary search

int index = binarySearch(myList, targetElement);

// Display the result

if (index != -1) {

System.out.println("Element " + targetElement + " found at index " + index + ".");

} else {

System.out.println("Element " + targetElement + " not found in the list.");

}

}

}

Now, let's simulate the binary search in Java step by step:

Start with the sorted list: [5, 6, 8, 9, 10, 15, 20, 21, 25, 30].

Find the middle element (10). Since it is not equal to the target (30) and is less than the target, we discard the left half of the list.

New list to search: [15, 20, 21, 25, 30].

Find the middle element (21). It is less than the target (30), so we discard the left half again.

New list to search: [25, 30].

Find the middle element (25). It is less than the target (30), so we discard the left half again.

New list to search: [30].

Find the middle element (30). It is equal to the target (30).

The binary search algorithm found the target element (30) in the list at index 4. Therefore, the output will be:

mathematica

Copy code

Element 30 found at index 4.

This demonstrates how a binary search works in Java on a sorted list by repeatedly dividing the search interval in half until the target element is found or it is determined that the element is not in the list.