# Day-2

ASSIGNMENT-CSA 0321

NAME:D.Rohab

REG NO:192311359

# BINARY SEARCH:

Linear search is a straightforward algorithm that checks each element in a list one by one until it finds the target. Its time complexity is O(n), making it efficient for small datasets but less so for larger ones.

**Steps of the Linear Search Algorithm**

1. **Start at the beginning** of the list or array.
2. **Check each element** one by one:
   * If the current element matches the target, return its index.
   * If it doesn't match, move to the next element.
3. **Repeat** until you find the target or reach the end of the list.
4. If the target is not found after checking all elements, return a value (commonly -1) indicating that the target is not in the list.

CODE:

#include <stdio.h>

int main()

{

int a[10]={2,3,4,5,67,9,6,5,3,7};

int target=5;

for(int i=0;i<10;i++)

{

if(a[i]==target)

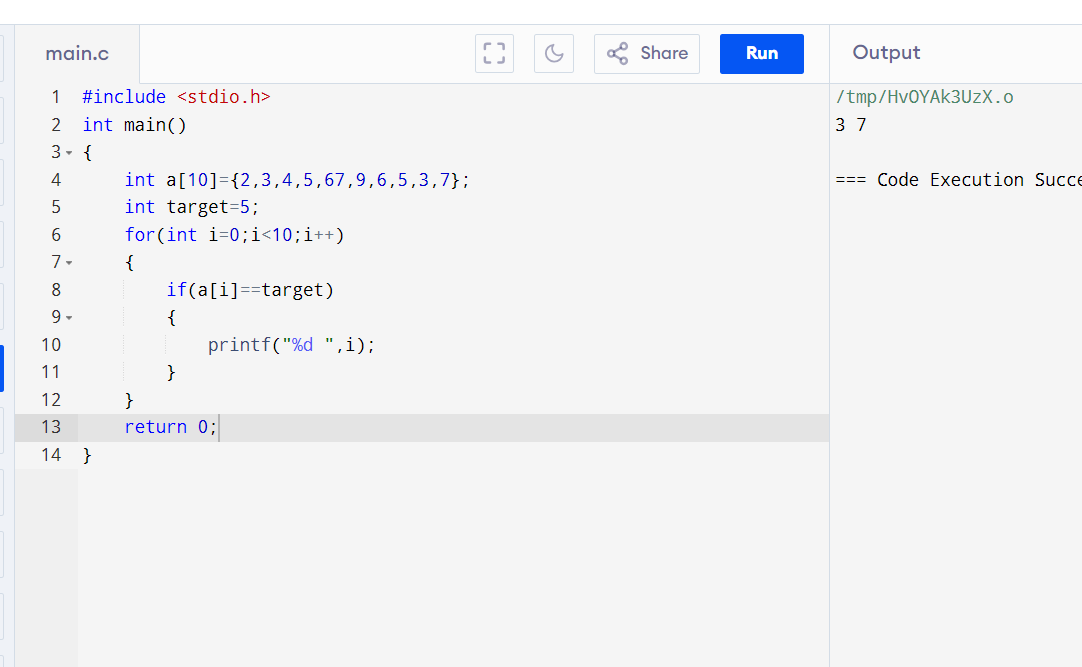
{

printf("%d ",i);

}

}

return 0;

}

# BINARY SEACH :

**Definition**: Binary search is an efficient algorithm for finding a target value within a **sorted** array or list. It works by repeatedly dividing the search interval in half.

**Steps of the Binary Search Algorithm**

1. **Start with two pointers**: one at the beginning (low) and one at the end (high) of the sorted array.
2. **Calculate the middle index**: mid = (low + high) / 2.
3. **Compare the middle element** with the target:
   * If the middle element is equal to the target, return the index.
   * If the middle element is less than the target, narrow the search to the upper half by updating low = mid + 1.
   * If the middle element is greater than the target, narrow the search to the lower half by updating high = mid - 1.
4. **Repeat** the process until low exceeds high.
5. If the target is not found, return a value (commonly -1) indicating that the target is not in the list.

CODE:

#include <stdio.h>

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

int low = 0, high = size - 1;

while (low <= high) {

int mid = low + (high - low) / 2;

if (arr[mid] == target) return mid;

if (arr[mid] < target) low = mid + 1;

else high = mid - 1;

}

return -1;

}

int main() {

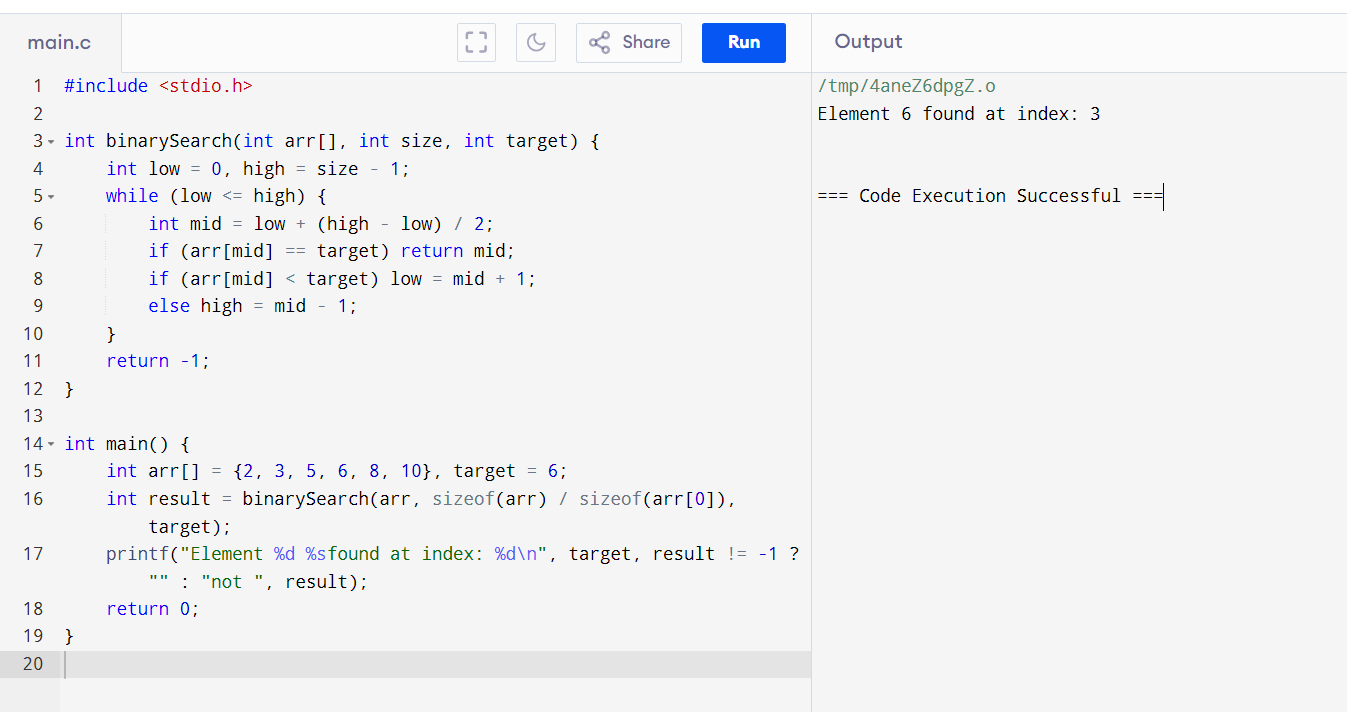
int arr[] = {2, 3, 5, 6, 8, 10}, target = 6;

int result = binarySearch(arr, sizeof(arr) / sizeof(arr[0]), target);

printf("Element %d %sfound at index: %d\n", target, result != -1 ? "" : "not ", result);

return 0;

}



# DIFFERENTIATION BETWEEN LINEAR AND BINARY SEARCH

Linear Search

\*\*Definition\*\*: Linear search, also known as sequential search, is a basic searching algorithm that checks each element in a list or array sequentially until the target element is found or the list ends.

\*\*How It Works\*\*:

1. Start from the first element of the array.

2. Compare each element with the target value.

3. If a match is found, return the index of that element.

4. If the end of the list is reached without finding the target, return a value indicating the target is not present (commonly -1).

\*\*Characteristics\*\*:

- \*\*Time Complexity\*\*: O(n), where n is the number of elements. The worst-case scenario occurs when the target is at the end or not present.

- \*\*Space Complexity\*\*: O(1) for the iterative version since it uses a fixed amount of space.

- \*\*Efficiency\*\*: Suitable for small datasets or unsorted lists due to its simplicity.

### Binary Search

\*\*Definition\*\*: Binary search is an efficient algorithm for finding a target value within a sorted array or list. It reduces the search space by half with each comparison.

\*\*How It Works\*\*:

1. Begin with two pointers, `low` and `high`, representing the current search range.

2. Calculate the middle index: `mid = (low + high) / 2`.

3. Compare the middle element with the target:

- If equal, return the index.

- If less than the target, narrow the search to the upper half (update `low`).

- If greater than the target, narrow the search to the lower half (update `high`).

4. Repeat the process until the target is found or the search space is exhausted.

\*\*Characteristics\*\*:

- \*\*Time Complexity\*\*: O(log n) because the search space is halved with each comparison.

- \*\*Space Complexity\*\*: O(1) for the iterative version; O(log n) for the recursive version due to call stack usage.

- \*\*Efficiency\*\*: Ideal for large datasets that are already sorted. It dramatically reduces the number of comparisons needed compared to linear search.

Summary of Differences

- \*\*Data Requirement\*\*: Linear search works on both sorted and unsorted data, while binary search requires the data to be sorted.

- \*\*Performance\*\*: Linear search is slower for large datasets, with a time complexity of O(n), whereas binary search is faster, with a time complexity of O(log n).

- \*\*Use Cases\*\*: Linear search is often used in simple or small data scenarios, while binary search is favored in performance-critical applications where data is sorted.

Both algorithms serve important roles in computer science and software development, and the choice between them depends on the specific requirements of the problem at hand.