Binary Search

Binary Search is a fast and efficient algorithm for finding a target element in a sorted list.
It repeatedly divides the search space in half until the element is found (or the range is empty).

Think of it like this:
While looking for a word in a dictionary, we don't start from page one; you open near the middle, decide which half to continue in, and keep narrowing down.

- **Algorithmic Steps (in pseudocode / algorithm format)**

Algorithm: BinarySearch(A, n, key)
Input: Sorted array A of n elements, value key to be searched
Output: Index of key if found, else -1

1. low ← 0

2. high ← n - 1

3. while low ≤ high do
    4. mid ← (low + high) / 2
    5. if A[mid] == key then
    6. return mid // key found
    7. else if A[mid] < key then
    8. low ← mid + 1 // search in right half
    9. else
    10. high ← mid - 1 // search in left half

4. end while

5. return -1 // key not found

Implementation in C++

```
#include <iostream>

#include <iostream>

using namespace std;


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

    int low = 0, high = n - 1;



    while (low <= high) {

        int mid = (low + high) / 2;  // or low + (high - low)/2 to avoid
overflow



        if (arr[mid] == key)

            return mid;  // found

        else if (arr[mid] < key)

            low = mid + 1;  // go right

        else

            high = mid - 1; // go left

    }



    return -1;  // not found

}
```

Time Complexity

Case	Comparisons	Time
Best (middle element first)	1	**O(1)**
Average	$\log_2(n)$	**O(log n)**
Worst	$\log_2(n)$	**O(log n)**

Space complexity: O(1) (for iterative version)
Sorted array needed? ✅ Yes, absolutely.
Stable? ❌ Not applicable (not a sorting algorithm).
Approach: Divide and Conquer

Summary

Binary Search beats Linear Search hands down for large **sorted** datasets. But if your data isn't sorted, you're better off with Linear Search, or sort first and then apply Binary Search.