🔍 1. Linear Search

💡 Real-World Analogy:

Imagine you're looking for your **Piyush** in a group photo of 10 people. You start from the **left**, checking one face at a time.
That's **Linear Search** — go through each element one by one.

📜 Code:

```
#include <stdio.h>
int main() {
    int numbers[] = {5, 3, 8, 6, 2};  // A list of numbers
    int target = 6;                   // Number we want to find
    int size = sizeof(numbers) / sizeof(numbers[0]);

    // Start checking each element one-by-one
    for(int i = 0; i < size; i++) {
        if(numbers[i] == target) {
            printf("Number %d found at index %d.\n", target, i);
            return 0; // Exit the program once we find it
        }
    }

    // If we finish the loop, the number was not found
    printf("Number %d not found in the array.\n", target);
    return 0;
}
```

🧠 Step-by-Step Explanation:

1. **numbers[] = {5, 3, 8, 6, 2}** — This is like a shelf of books with numbers written on the cover.
2. **target = 6** — We're trying to find the book with number 6.
3. **for loop** — Go through each book one at a time from left to right.
4. **If match found** — Print its position (index).
5. **return 0;** — We found it, no need to search further. Exit.
6. **If we reach the end** — Print "not found".

⚡ Summary:

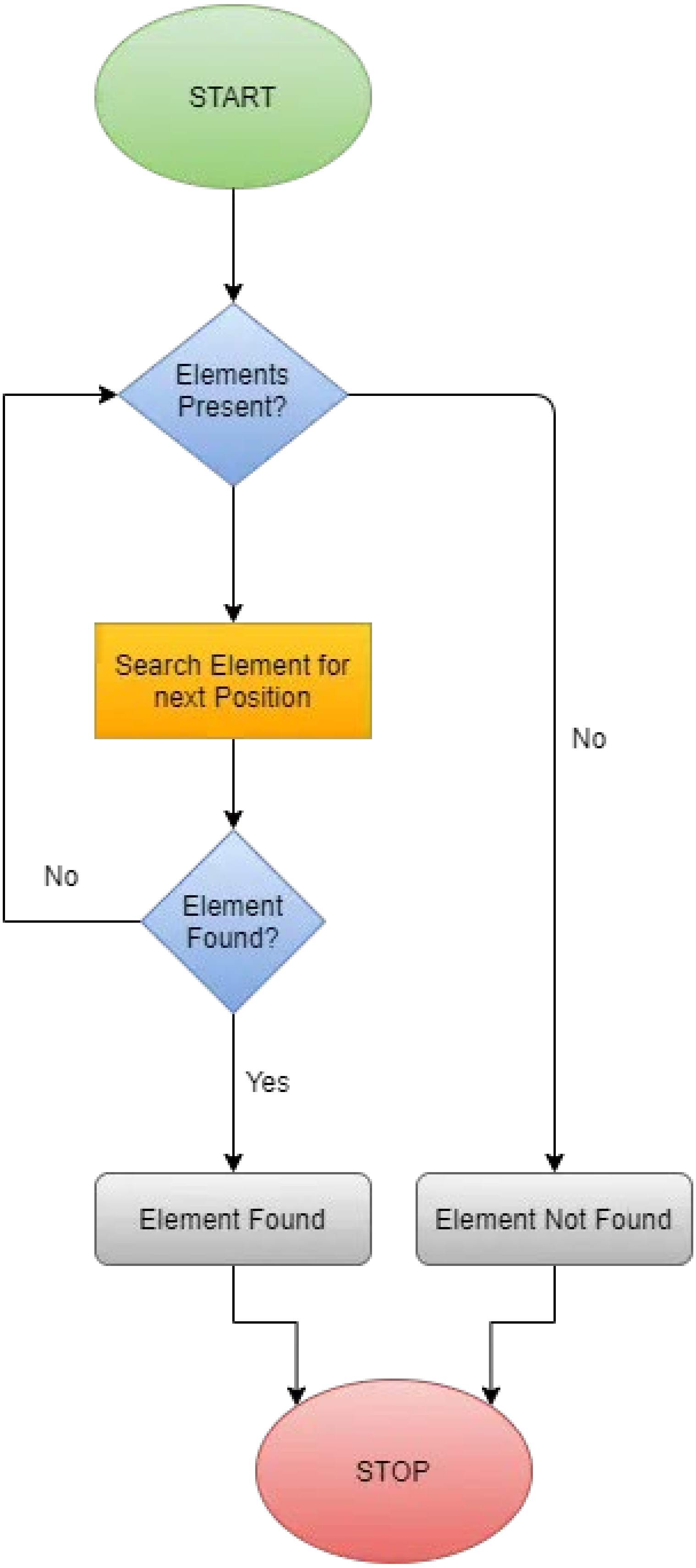
- ✅ Works for any list (sorted or not).
- ❌ Slower for large lists (like flipping pages one-by-one in a dictionary).

🧠 Explanation:

- Checks every element from left to right.
- Stops as soon as match is found.
- Simple and works on any array (unsorted or sorted).

Algorithm:

```
1. Start
2. Initialize the array and the target element to search
3. Determine the size (number of elements) in the array
4. Set index i = 0
5. Repeat the following steps while i < size:
   a. If array[i] == target:
      → Print "Element found at index i"
      → Exit
   b. Else:
      → Increment i by 1
6. If loop ends without finding:
   → Print "Element not found in the array"
7. Stop
```


START
Elements Present?
Search Element for next Position
No
Element Found?
No
Yes
Element Found
Element Not Found
STOP

🔍 2. Binary Search

💡 Real-World Analogy:

Looking for the word "Tiger" in an **alphabetical dictionary**. You don’t flip page by page. Instead, you:

1. Open the middle page.
2. If it’s “Elephant”, go right.
3. If it’s “Zebra”, go left.
4. Keep halving the search.

That's **Binary Search** — fast but requires the list to be sorted!

📜 Code:

```
#include <stdio.h>
int main() {
    int numbers[] = {2, 3, 5, 6, 8}; // Sorted array
    int target = 6;               // Number to search
    int size = sizeof(numbers) / sizeof(numbers[0]);
    int low = 0, high = size - 1;
    // Keep checking the middle element
    while(low <= high) {
        int mid = (low + high) / 2;
        if(numbers[mid] == target) {
            printf("Number %d found at index %d.\n", target, mid);
            return 0; // Exit as soon as it's found
        } else if(numbers[mid] < target) {
            low = mid + 1; // Ignore the left half
        } else {
            high = mid - 1; // Ignore the right half
        }
    }
    // If loop ends, number wasn't found
    printf("Number %d not found in the array.\n", target);
    return 0;       }
```

🧠 Step-by-Step Explanation:

1. **Sorted List Required!** — {2, 3, 5, 6, 8}.
2. **Start with the full range**: low = 0, high = 4.
3. **Find middle**: mid = (low + high)/2.
4. **Check middle element**:
   - If equal to target → print and stop.
   - If less → search right half.
   - If more → search left half.
5. **Repeat until found or range is empty**.

📈 Summary:

- ✅ Super fast for large sorted lists.
- ❌ Won't work correctly if the list isn't sorted.

🧪 Real-World Use Case Comparison:

Scenario	Use Linear Search	Use Binary Search
Friend's name in random list	✅ Yes	❌ No
Finding word in dictionary	❌ Slow	✅ Perfect
Checking small shopping list	✅ Good enough	❌ Unnecessary
Searching sorted product IDs	❌ Inefficient	✅ Super fast

🧠 Explanation:

- Starts by checking the middle of the array.
- Eliminates half the array in each step.
- Requires the array to be **sorted**.
- Way faster than linear search for big lists.

Algorithm:

```
1. Start
2. Initialize the sorted array and the target element
3. Set low = 0 and high = size - 1
4. Repeat the following steps while low <= high:
   a. Calculate mid = (low + high) / 2
   b. If array[mid] == target:
      → Print "Element found at index mid"
      → Exit
   c. Else if array[mid] < target:
      → Set low = mid + 1
   d. Else:
      → Set high = mid - 1
5. If loop ends without finding:
   → Print "Element not found in the array"
6. Stop
```