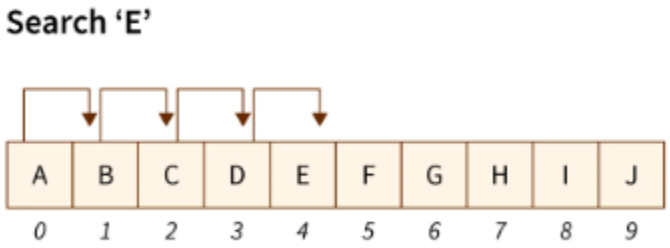
Searching

Finding a given element's occurrence in a list is the process of searching. In any data structure where any element is stored, search algorithms are made to check or retrieve that element. If the element being searched for is found in the list, the search is considered successful; otherwise, it is considered unsuccessful.
When it comes to searching for an element in an array, two common approaches are there, linear search and binary search. Both methods have their own strengths and weaknesses, making them suitable for different scenarios.

Linear Search:
Linear search, also known as sequential search, involves traversing the array one element at a time until the desired element is found. This algorithm is simple to implement and works well for small arrays or unsorted data. However, it becomes inefficient for larger datasets due to its linear time complexity.
In a linear search, each element is compared with the target element until a match is found or the end of the array is reached. This search method is like flipping through pages of a book to find a specific word.



Algorithm
1. Assume we need to find an element that is in an array in random order.
2. Start with the first position and compare it to the target in order to search for a target element.
3. If the current element matches the target element, return the position of the current element.
4. If not, move on to the next one until we reach the very end of an array.
5. If still unable to find the target, return -1.

Program for Linear Search

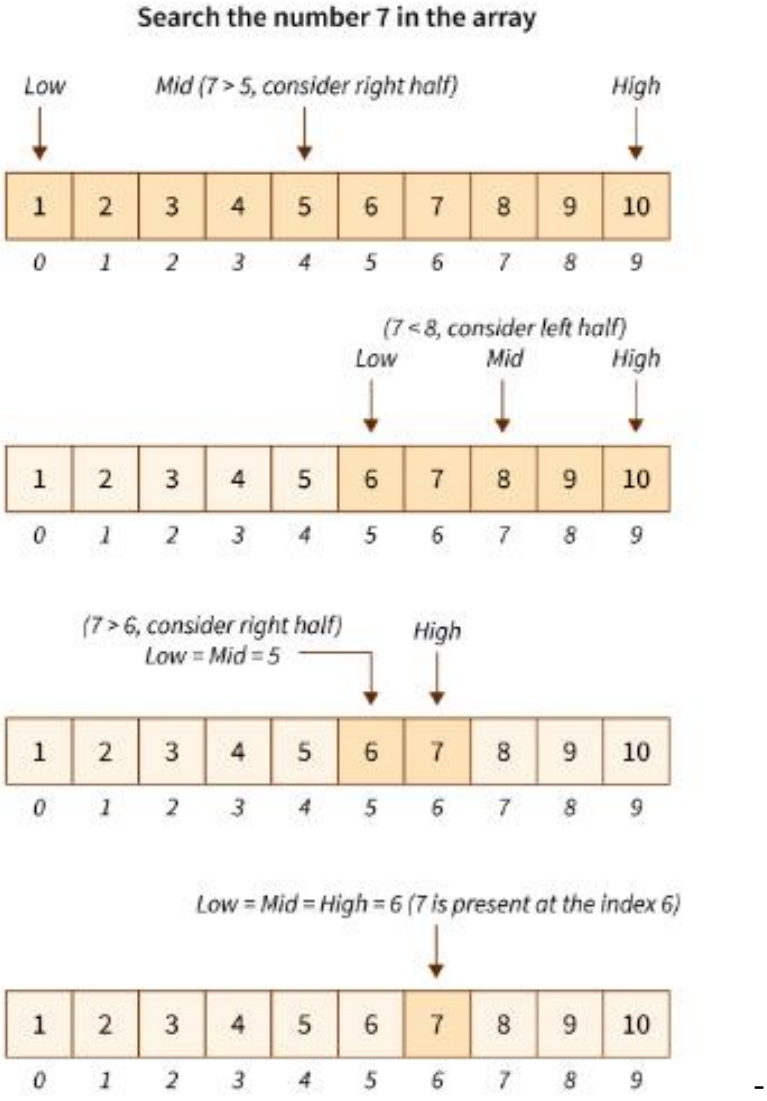
```
#include <iostream>
using namespace std;

int linearSearch(int arr[], int n, int target) {
   for (int i = 0; i < n; ++i) {
      if (arr[i] == target) {
         return i; // Element found at index i
      }
   }
   return -1; // Element not found
}

int main() {
   int arr[] = {10, 25, 30, 45, 50, 60};
   int n = sizeof(arr) / sizeof(arr[0]);
   int target = 45;
   int result = linearSearch(arr, n, target);
   if (result != -1) {
      cout << "Element found at index: " << result << endl;
   } else {
      cout << "Element not found." << endl;
   }
   return 0;
}
```

Binary Search:

Binary search, on the other hand, is a more efficient algorithm that works specifically with sorted arrays. It follows a divide-and-conquer approach, repeatedly dividing the search space in half. This results in a significantly reduced number of comparisons compared to linear search, leading to a faster search process.

Binary search takes advantage of the sorted nature of the array. It compares the middle element with the target and eliminates half of the remaining elements based on whether the target is greater or smaller than the middle element. This

process continues until the target element is found or the search space is exhausted.



Algorithm

1. Start with the middle element of the array as a current key.
2. If the middle element value matches the target element then return the index of the middle element.
3. Else check if the target element is lesser than the middle element, then continue the search in the left half.
4. If the target element is greater than the middle element, then continue the search in the right half.

5. Check from the second point repeatedly until the value is found or the array is empty.
6. If still unable to find the target, return -1.

The **three main cases** that are used in the binary search:
Case 1: arr[mid] == target // then return the index
Case 2: arr[mid] < target // then right=mid-1
Case 3: arr[mid] > target // then left = mid+1.

Program for Binary search

```
#include <iostream>
using namespace std;

int binarySearch(int arr[], int left, int right, int target) {
   while (left <= right) {
      int mid = left + (right - left) / 2;

      if (arr[mid] == target) {
         return mid; // Element found at index mid
      }

      if (arr[mid] < target) {
         left = mid + 1;
      } else {
         right = mid - 1;
      }
   }
   return -1; // Element not found
}

int main() {
   int arr[] = {10, 25, 30, 45, 50, 60};
   int n = sizeof(arr) / sizeof(arr[0]);
   int target = 45;
```

```
    int result = binarySearch(arr, 0, n - 1, target);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found." << endl;
    }
    return 0;
}
```

Key Differences between Linear and Binary Searching Algorithms:

1. Linear search works for both sorted and unsorted arrays, while binary search requires a sorted array.
2. Linear search has a linear time complexity of O(n), making it less efficient for larger datasets. Binary search boasts a logarithmic time complexity of O(log n), making it faster for larger datasets.
3. Binary search requires a sorted array, which might add overhead if the array needs frequent updates. Linear search can handle dynamic data without requiring re-sorting.