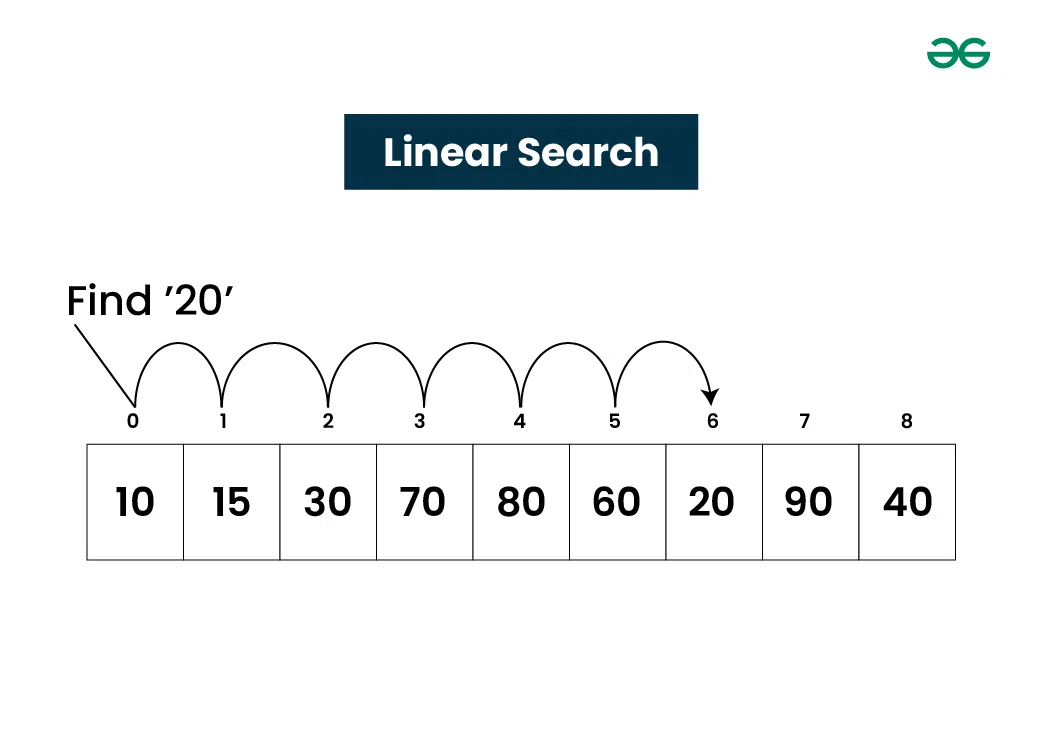
**Searching Algorithms**

* **Searching**is the fundamental process of locating a specific element or item within a collection of data. This collection of data can take various forms, such as arrays, lists, trees, or other structured representations.

1. Linear Search
2. Binary Search
3. Ternary Search
4. **Linear Search**

* Linear Search, also known as Sequential Search, is one of the simplest and most straightforward searching algorithms.
* It works by sequentially examining each element in a collection of data(array or list) until a match is found or the entire collection has been traversed.

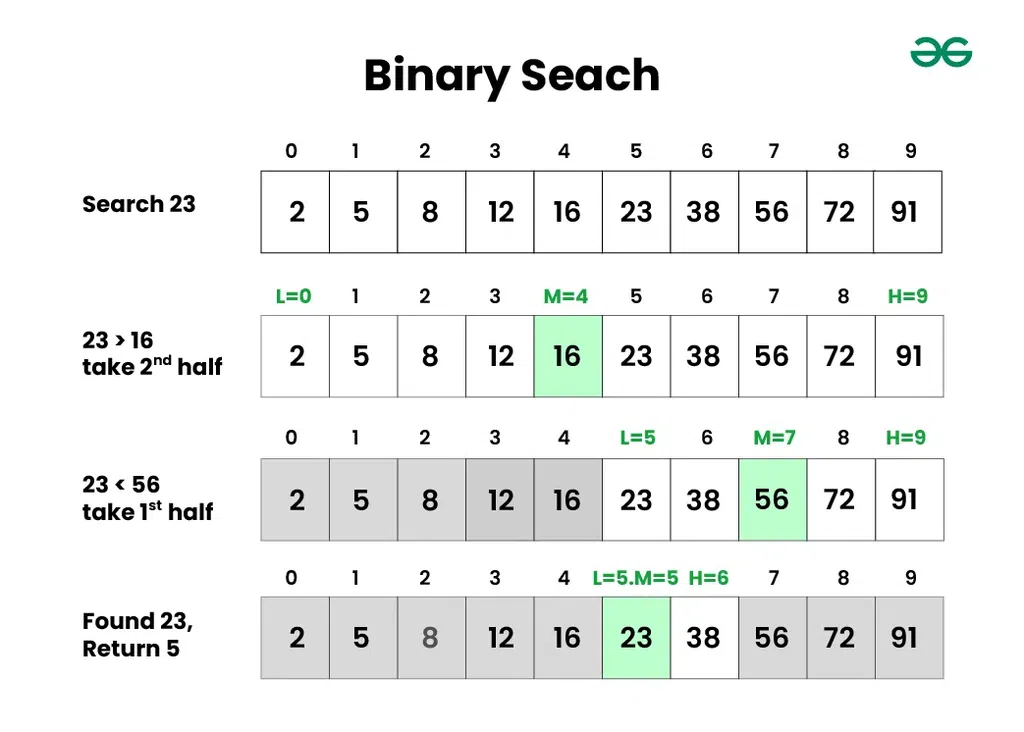


**Algorithm of Linear Search:**

* The Algorithm examines each element, one by one, in the collection, treating each element as a potential match for the key you're searching for.
* If it finds any element that is exactly the same as the key you're looking for, the search is successful, and it returns the index of key.
* If it goes through all the elements and none of them matches the key, then that means "No match is Found".

1. **Binary search :**

* Binary Search is defined as a searching algorithm used in a**sorted array**by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to **O(log N).**



**Algorithm of Binary Search:**

* Divide the search space into two halves by finding the middle index “**mid**”.
* Compare the middle element of the search space with the **key**.
* If the **key** is found at middle element, the process is terminated.
* If the **key** is not found at middle element, choose which half will be used as the next search space.
  + If the key is smaller than the middle element, then the **left** side is used for next search.
  + If the key is larger than the middle element, then the **right** side is used for next search.
* This process is continued until the key is found or the total search space is exhausted.

**Pseudo Code for Binary Search:**

binarySearch(collection, key):

left = 0

right = length(collection) - 1

while left <= right:

mid = (left + right) // 2

if collection[mid] == key:

return mid

elif collection[mid] < key:

left = mid + 1

else:

right = mid - 1

return "Not found"

**3.Ternary Search:**

* Ternary Search is a searching algorithm that divides the search space into **three parts** instead of two, as in**Binary Search**.

**Algorithm Ternary Search:**

* In Ternary Search, start with two midpoints, **oneThird** and **twoThirds**, which divide the collection into three roughly equal parts.
* Compare the elements at **oneThird** and **twoThirds** with the target key you're searching for.
* Three Possibilities:
  + If **oneThird** contains the key, you're done and return the index of **oneThird**.
  + If **twoThirds** contains the key, you're done and return the index of **twoThirds**.
  + If the **key** is less than the element at **oneThird**, eliminate the rightmost one-third of the collection and focus on the left two-thirds.
* If the key is greater than the element at **twoThirds**, eliminate the leftmost one-third of the collection and focus on the right two-thirds.
* Repeat this process iteratively until either **key** is found or determine that it's not present in the collection.

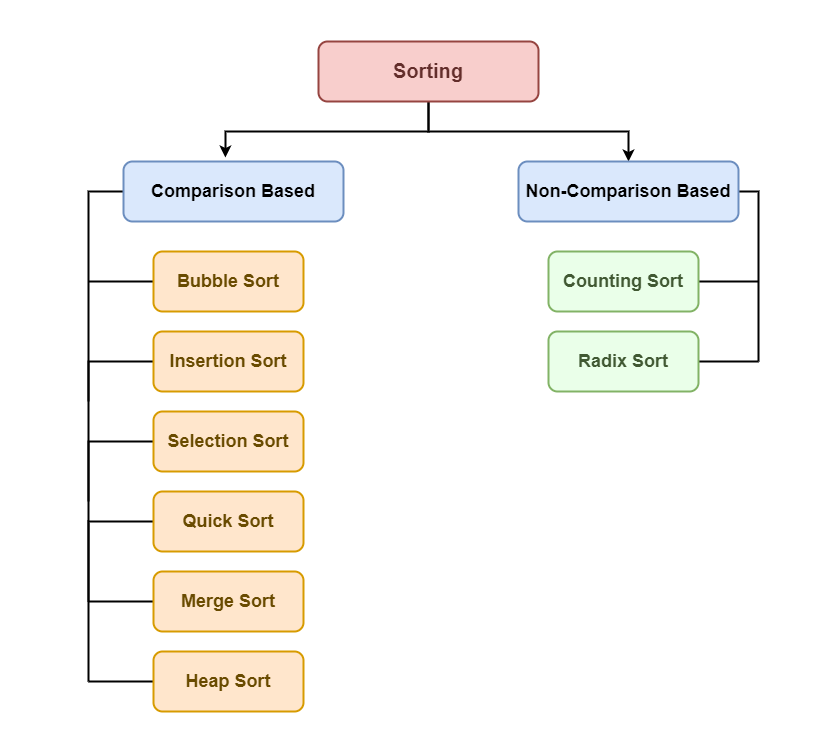
**Sorting:**

* **Sorting**refers to rearrangement of a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

**Types of Sorting Techniques**

There are various sorting algorithms are used in data structures. The following two types of sorting algorithms can be broadly classified:

* **Comparison-based:**We compare the elements in a comparison-based sorting algorithm)
* **Non-comparison-based:**We do not compare the elements in a non-comparison-based sorting algorithm)



**BUBBLE SORT**

- It is also known as Sinking sort

- We repeatedly compare each pair of adjacent items and swap them if they are in the wrong order

- If the first iteration is in the increasing order then last element(MAX) will be in right position

To swap two elements in an array using bubble sort

#include <stdio.h>

// Function to swap two elements

void swap(int \*x, int \*y) {

int temp = \*x;

\*x = \*y;

\*y = temp;

}

// Bubble Sort function

void bubbleSort(int arr[], int n) {

// Outer loop for number of passes

for (int i = 0; i < n - 1; i++) {

// Inner loop for each pass comparison

for (int j = 0; j < n - i - 1; j++) {

// Swap if the element is greater than the next element

if (arr[j] > arr[j + 1]) {

swap(&arr[j], &arr[j + 1]);

}

}

}

}

// Function to print the array

gvoid printArray(int arr[], int size) {

for (int i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr[] = {64, 34, 25, 12, 22, 11, 90};

int n = sizeof(arr) / sizeof(arr[0]);

printf("Original array: \n");

printArray(arr, n);

// Call bubble sort

bubbleSort(arr, n);

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}