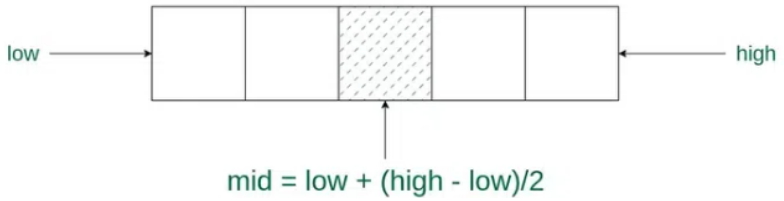
I. Binary Search

1. Algorithm
   - Array must be sorted
   - Divide the search data into 2 halves



   - Compare the middle element of the data search with the key (key = array[mid])
   - if key is not found:
     +) key < array[mid] then left side is used to for next search
     +) key > array[mid] then right side is used to for next search
   - Until: key is found or low > high
2. Time Complexity
   O(log(n))
3. Pros and Cons
   - Pros: faster than linear search, more efficient than interpolation search and exponential search
   - Cons: the array must be sorted

II. Regular Expression
[cheet-sheet-syntax for regex]
[cheet-sheet-syntax for regex 2]

III. Prefix Sum - Counting Sort

1. Prefix Sum
   a) Definition
   - Prefix sum: cumulative sum, compute the sum of elements in a sequence up to a certain index
   - Prefix sum of an array is a new array where the value at each index represents the sum of all elements from the beginning of the original array up to that index

   b) Example
   - Given array: [1, 3, 5, 7, 9]
   - Then the prefix sum array would be: [1, 4, 9, 16, 25].

     At index 0, the prefix sum is 1 (sum of elements up to index 0).At index 1, the prefix sum is 1 + 3 = 4 (sum of elements up to index 1). And So on…

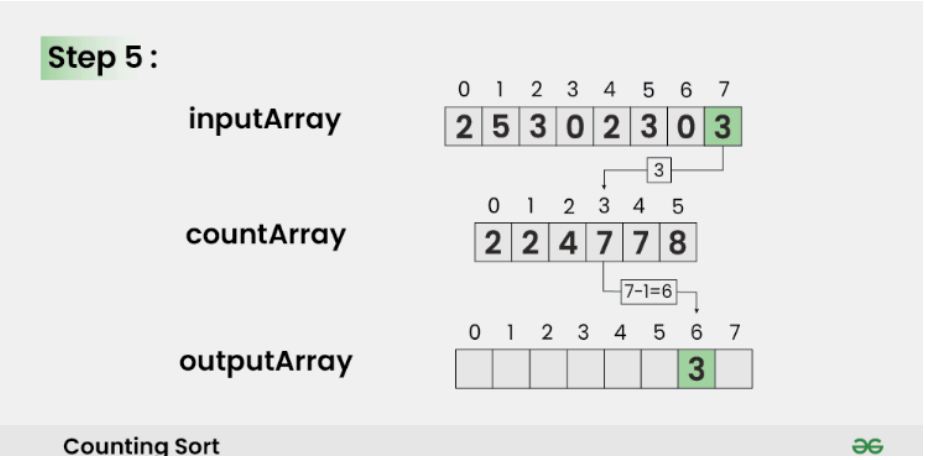
2. Counting Sort

a) Definition

- Counting Sort is a non-comparison-based sorting algorithm that works well when there is a limited range of input values.
- The basic idea behind Counting Sort is to count the frequency of each distinct element in the input array and use that information to place the elements in their correctly sorted positions.

b) Algorithms

- Declare an auxiliary array countArray[] of size max(inputArray[])+1 and initialize it with 0
  where countArray[] is an auxiliary array to hold the count of each unique element in the input array
- Calculate the prefix sum at every index of array inputArray[]
- Create an array outputArray[] of size N.
- Traverse array inputArray[] from end and update outputArray[countArray[inputArray[i]] – 1] = inputArray[i]. Also, update countArray[inputArray[i]] = countArray[inputArray[i]] (as shown in Fig below)



- Detail steps in: https://www.geeksforgeeks.org/counting-sort/

c) Time Complexity

- O(N+M), where N and M are the size of inputArray[] and countArray[] respectively.

d) Pros and Cons

- Pros: if range values is small, Counting sort generally performs faster than all comparison-based sorting algorithms, such as merge sort and quicksort.
  Easy to code
- Cons: is inefficient if the range of values is very large. It uses extra space for sorting the array elements(not In-place sorting)