**Arrays:**

**1.Kids with greatest number of candies**

<https://leetcode.com/problems/kids-with-the-greatest-number-of-candies/>

Approach: Simple approach, first iterate over the array to find the maximum number of candies in the candies array, and then again iterate over the candies array and make the sum candies available and extra candies and compare the summation to the maximum candy got in the first iteration.

**2.Remove duplicates from sorted array**

<https://leetcode.com/problems/remove-duplicates-from-sorted-array/>

Approach: If the current element is not equal to the next element then replace the current element with the next element.

Time Complexity: O(log N) N is the number of elements in the array

**3.Richest customer wealth**

<https://leetcode.com/problems/richest-customer-wealth/>

Approach: Make the sum amounts in all the banks of the customer and track the maximum amount.

**4.Two sum**

<https://leetcode.com/problems/two-sum/>

Approach 1: (Bruteforce) Iterate over the array and check for each element whether there exists another element which upon summation it equals target or not.

Approach 2: (Efficient using HashMap)

**5. Sum of Infinite array**

<https://www.codingninjas.com/codestudio/problem-details/sum-of-infinite-array_873335>

Approach 1: (Bruteforce)

Approach 2: (Efficient using some math modular arithmetic)

**6.Build array from permutation**

<https://leetcode.com/problems/build-array-from-permutation/>

Approach: How to solve the problem is itself provided in the question.

**7.Concatination of array**

<https://leetcode.com/problems/concatenation-of-array/>

Approach: The naive approach is that first copy the elements of nums and then starts copying the same elements once again till 2\*n size. Better approach is to use the modulus operator such that

for (int i = 0; i < 2 \* n; i++) {

ans[i] = nums[i % n];

}

**8. Minimum number of distinct elements after k removals**

<https://leetcode.com/problems/least-number-of-unique-integers-after-k-removals/>

Approach 1: The first approach is to count all the occurrences of each integers in hashmap, sort the hashmap according to the least frequency of the elements in ascending order, and go on subtracting the k value as sorted frequencies of integers from the sorted map. And go on subtracting k value and remove the element from the unsorted map if and only if the frequency of the removing element is less than or equal to k.

Time complexity: O(n log(n))

Approach 2: The second approach is to use the priority queue and hashmap the approach is same as that of the Approach [1], but this approach is efficient that previous as the priority queue makes the frequency of the integer from the input array in descending order. And go on removing the elements from the priority queue until the k >= 0.

Time complexity: O(n log(n))

**Binary Search:**

**1.Find peak element**

<https://leetcode.com/problems/find-peak-element/>

Approach 1: (Bruteforce) Iterate over the array and check whether the element exists such that its left and right elements are less than that element.

Time complexity: O(N) N is the number of elements in array

Approach 2: (Binary Search) Apply the binary search such that,

if (nums[mid] <nums[mid + 1]) {

start = mid + 1;

} else {

end = mid - 1;

}

then return the end value.  
Time complexity: O(log N) N is the number of elements in array

**2.Find first and the last position of element in the sorted array**

<https://leetcode.com/problems/find-first-and-last-position-of-element-in-sorted-array/>

Approach 1: (Bruteforce) Iterate over the first occurrence of the element and then iterate from the last index of the array and store both of the results in the two variables.

Approach 2: (Binary Search) Apply the binary search for the first occurrence such that,

while (low <= high) {

mid = (low + high) / 2;

if (A[mid] > B) {

high = mid - 1;

} else if (A[mid] < B) {

low = mid + 1;

} else {

res = mid;

high = mid - 1;

}

}

Apply the binary search for the last occurrence such that,

while (low <= high) {

mid = (low + high) / 2;

if (A[mid] > B) {

high = mid - 1;

} else if (A[mid] < B) {

low = mid + 1;

} else {

res = mid;

low = mid + 1;

}

}

Time Complexity: O(log N) N is the number of elements in array

**3.Intersection of two arrays**

<https://leetcode.com/problems/intersection-of-two-arrays/>

Approach 1: (Bruteforce) Iterate over both the arrays and find the elements which are common in both of the arrays and maintain the Set data structure for not making the duplicates of the common elements.

Approach 2: (Binary Search) Use the set data structure, sort the array1, and apply the Binary search operation only on the array2(pass the element of the array 2 as the key to search in the binary search function), and return the elements of the set.

Time Complexity: O(log N) N is the number of elements in the length of the array.

**4.Search insert position**

<https://leetcode.com/problems/search-insert-position/>

Approach 1: (Bruteforce) Iterate over the entire array and if the element found such that current element is greater than or equal to target then return current element index. and if the iterator is reached to the last element then if it is less than target then return i+1th index.

Approach 2: (Binary Search) Apply the binary search such if the mid element greater than target element make end=mid-1 and if target is more than mid element then make start=mid+1 or else return mid.

**5.Two sum - II**

<https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/>

Approach: (Binary search) as the input array itself is the sorted and apply the binary search such that the elements in the start and end makes the summation equal to the target then the answer is indices start and end if the summation is less than target then start=start+1 else end=end-1

**Sorting:**

**1.Sort array by parity**

<https://leetcode.com/problems/sort-array-by-parity/>

Approach 1: (Bruteforce) get the evens and odds in separate lists and make the new array ans[] of length of number of evens and odds and evens first and then odds after inserting the evens in the answer array and then return ans array.

Approach 2: (Efficient in terms of time compl.) make two pointers even and odd and assign them to start and end of the given array respectively. Then check whether the even has occurred in an array pointed by an even pointer, if found leave as it is and also same with odd in which the odd occurs in the last position of then leave it as it is. If even and odd found simultaneously then swap them.

**String:**

**1.First unique character**

<https://leetcode.com/problems/first-unique-character-in-a-string/>

Approach 1: Traverse the entire string and maintain the frequency array of the each and every character. And again traverse the string and compare the frequency which is having 1 and occurs first in the string is itself is the first unique character in the string. Compare the string with the character and not with the ASCII values otherwise the operation of comparison becomes slower.

**2.Longest Common prefix**

<https://leetcode.com/problems/longest-common-prefix/>

Approach 1: In the given list of string and make the first string as the reference string and go on comparing this reference string with the other strings in the list for their common prefixes, by matching the reference string with the other strings in the list character by character.

**3.Valid palindrome**

Approach 1: (Bruteforce) Iterate the entire string and check the string character by character and then compare whether that character is alphabet special character or digit.

Approach 2: (Efficient)

**Recursion:**

**1.Maximum and minimum in array using recursion**

<https://practice.geeksforgeeks.org/problems/find-minimum-and-maximum-element-in-an-array4428/1>

Approach: Recursively iterate the entire array and keep the track of the current maximum and find whether the current element is maximum or minimum and if they are greater or smaller than current max or min elements and if is true update current max or min.