**Approach & Basics:**

* Calculate time complexity using constraints, a typical machine does 10^8 operations per sec (in general).
* TC can give you hint regarding which algorithm or data structure to use.
* MC can be overlooked in coding round but must be taken care of in interviews.
* 10^18=O(logN) ; 10^8=O(N) ; 10^6=O(NlogN) ; 10^4=O(N^2) ; and so on like O(N^3) ; O(2^N) ; O(N!) .
* For debugging:- Test custom cases, comment out code, stub the code or function (hard i/p).
* System.out.println(Integer.MAX\_VALUE);//Max value is of the order 2\*10^9//as int is 4bytes
* System.out.println(Long.MAX\_VALUE);//Max value is of the order 9\*10^19//and long is 8bytes
* Below 4 algorithms are mostly asked in Interviews but the sliding window algorithm is very useful.

**1. Swap Sort:**

* A very useful algorithm which can detect missing and duplicate elements in array of 1toN. TC = O(N);MC = O(1)

//This problem can be solved by using brute force,using HashSet or HashMap,by sorting or by using maths and  
//making equations, but it will get complex if we have to find more than one missing or duplicate number.  
//Disclaimer : if the array is read-only swap-sort then it won't work (ofc).  
//Must consider this example for easier understanding:  
//Array index : {0,1,2,3,4}   
//Array elements : {4,2,3,1,1}  
public static void swapSort(int[] arr) {  
 int i = 0;//initialize pointer to traverse array  
 while (i < arr.length) {  
 if (arr[i] == arr[arr[i] - 1]) {//this will check if the element at ith index is what it should be  
 i++; //if it is increase the pointer  
 } else {  
 *swap*(arr, i, arr[i] - 1);//else we swap the number at ith index and the num. at arr[i]-1 index  
 }  
 }  
 i = 0;  
 while (i < arr.length) {//after that's done loop  
 if (arr[i] != i + 1) {//and if value at ith index is not same as i+1  
 System.out.println("Duplicate: " + arr[i]);//we found the duplicate  
 System.out.println("Missing: " + (i + 1));//we found the missing  
 }  
 i++;  
 }  
 System.out.println("=========");  
}  
public static void swap(int[] arr, int x, int y) {  
 int temp = arr[x];  
 arr[x] = arr[y];  
 arr[y] = temp;  
}

**2. Query Processing:**

* Query processing is used in questions when you are already given various types of generalized inputs/queries that your code is required to process somehow and then give the answer.There is no general way or algorithm used in this you have to use your mind to pre-process the queries in order to reduce search time by making Maps,Sets,Sorting or something like that.

**3. Multi Pointers:**

* Here we make multiple pointers which points the positions of elements in the array and then solve the question.
* It is generally of 2 types: 2 pointer algorithm, 3 pointer algorithm.(Generally used in arrays).
* Boilerplate is nothing specific just initializing the pointers and rest is pretty much mind game.

**4. Sliding Window: [Best question for this topic:** <https://leetcode.com/problems/minimum-window-substring/>**]**

* Applied when questions deals with sub-arrays(continuous ofc). It is of 2 types: Fixed and Variable.
* Both types require initaliziation like i,j(pointers to keep track of window length),sum etc.
* Biolerplate for Fixed Size Window:

//find max subArray for length k.  
//kadane's algorithm is for max sum sub-array in an array i.e. k is not specified.  
public static void maxSumForWindowSizeK(int[] array, int k) {  
 int sum = 0; int ans = Integer.MIN\_VALUE; int i = 0; int j = 0; //initializations

while (j < array.length) {//while j pointer is less than window size  
 sum = sum + array[j]; //do these operations in each iteration (default operation)

if (j - i + 1 < k) { //if the gap is less than window size.

j++; //do this till we hit window size.  
 } else if (j - i + 1 == k) { //once we hit window size do the required operations  
 ans = Math.*max*(sum, ans);  
 sum = sum - array[i];  
 i++; //now increase j and i by 1 to slide the window  
 j++;  
 }  
 }  
 System.out.println(ans);  
}

* In counting stuff or pattern questions generally string related most likely HashMap would be used.
* Biolerplate for Variable Size Window:

//Given a string you need to print the size of the longest possible substring that has exactly K unique characters. If there is no possible substring then print -1

public static void unique(String s, int k) {  
 HashMap<Character, Integer> map = new HashMap<>();//initializations  
 int j = 0; int i = 0; int ans = Integer.MIN\_VALUE;//initializations  
  
 while (j < s.length()) {//while j pointer is less than window size  
 map.put(s.charAt(j), map.getOrDefault(s.charAt(j), 0) + 1);//do these ops. in each iteration (dop.)  
 if (map.size() < k) {//if the condition is less than window size  
 j++; //do this to hit window size  
 } else if (map.size() == k) { //once we hit window size  
 ans = Math.*max*(ans, j - i + 1); //do the req. operations  
 j++;  
 } else {  
 while (map.size() > k) { // as it is variable size window, if the condition overcomes window size we need to keep remove stuff till the condition is not met again  
 map.put(s.charAt(i), map.get(s.charAt(i)) - 1); //do ops. to remove calcs. of ith pointer  
 if (map.get(s.charAt(i)) == 0) {  
 map.remove(s.charAt(i));  
 }  
 i++;//keep increasing i by 1 in each iteration  
 }  
 j++;// once it is done now increase j in order to calculate next window operation.  
 }  
 }  
 System.out.println(ans);  
}

* In general it can be solved using biolerplate and using DS like HashMaps,ArrayLists,PriorityQueues etc.
* Google’s favourite topic.

**4. Binary Search:**

* Used on sorted array has TC= O(logn) and MC = O(1). The below code is how to implement BinarySearch using recursion, but the best way is to use it using **Collections.binarySearch(List,element);** returns the index if the element is present in the list if not it returns a negative value. To reverse the list use **Collections.reverse(List);**

public static int binarySearch(ArrayList<Integer> al, int number, int low, int high) {  
 //Order agnostic search (binary search no matter how sorting is done)  
 //this is done using recursion very easy to implement, can be done using while loop too and then using break;  
 if (al.get(0) <= al.get(al.size() - 1) && low <= high) {  
 int mid = low + ((high - low) / 2);  
 if (al.get(mid) == number) {  
 return mid;  
 } else if (al.get(mid) > number) {  
 return *binarySearch*(al, number, low, mid - 1);  
 } else {  
 return *binarySearch*(al, number, mid + 1, high);  
 }  
 } else if (al.get(0) >= al.get(al.size() - 1) && low <= high) {  
 int mid = low + ((high - low) / 2);  
 if (al.get(mid) == number) {  
 return mid;  
 } else if (al.get(mid) < number) {  
 return *binarySearch*(al, number, low, mid - 1);  
 } else {  
 return *binarySearch*(al, number, mid + 1, high);  
 }  
 }  
 return -1;  
}