Arrays

- Concatenation of Array

```
class Solution {
    public int[] getConcatenation(int[] nums) {
        int n=nums.length;
        int[] ans=new int[n*2];
        for(int i=0;i<n*2;i++)
        {
            int x=i%n;
            ans[i]=nums[x];
        }
        return ans;
    }
}
```

- Contains duplicate

```
class Solution {
    public boolean hasDuplicate(int[] nums) {
        Arrays.sort(nums);
        for(int i=0;i<nums.length-1;i++)
        {
            if(nums[i]==nums[i+1])
            {
                return true;
            }
        }
        return false;
    }
}
```

Or

```
class Solution {
    public boolean hasDuplicate(int[] nums) {
        Set<Integer> s = new HashSet<>();   // Using hashset
        for (int i : nums)
        {
            if(s.add(i)==false) // returns false when duplicate elements are
present
                return true;
        }
        return false;
    }
}
```

- Valid anagram

```
class Solution {
    public boolean isAnagram(String s, String t) {

        if(s.length() != t.length())
            return false;

        HashMap<Character,Integer> map1=new HashMap<>();
        HashMap<Character,Integer> map2=new HashMap<>();

        for(int i=0;i<s.length();i++)
        {
            char ch=s.charAt(i);
            char ch1=t.charAt(i);
            map1.put(ch,map1.getOrDefault(ch,0)+1);
            map2.put(ch1,map2.getOrDefault(ch1,0)+1);
        }

        for (Character key : map1.keySet())
        {
            if (!map1.get(key).equals(map2.get(key)))
                return false;
        }
        return true;
```

```
    }
}
```

Or

```
class Solution {
    public boolean isAnagram(String s, String t) {
        if(s.length() != t.length())
            return false;

        HashMap<Character,Integer> map=new HashMap<>();

        for(int i=0;i<s.length();i++)
        {
            char ch=s.charAt(i);
            map.put(ch,map.getOrDefault(ch,0)+1);
        }

        for(char c : t.toCharArray())
        {
            if(map.containsKey(c)==false)
                return false;
            map.put(c,map.get(c)-1);
            if(map.get(c)==0)
                map.remove(c);
        }
        return map.isEmpty();
    }
}
```

Or

```
class Solution {
    public boolean isAnagram(String s, String t) {
        if(s.length() != t.length())
```

```
            return false;

        char[] ss=s.toCharArray();
        char[] tt=t.toCharArray();
        Arrays.sort(ss);
        Arrays.sort(tt);
        return Arrays.equals(ss,tt);
    }
}
```

- Two sum

```
class Solution {
    public int[] twoSum(int[] nums, int target) {

        HashMap<Integer,Integer> map=new HashMap<>();
        int[] arr=new int[2];

        for(int i=0;i<nums.length;i++)
        {
            int x=target-nums[i];
            if(map.containsKey(x))
            {
                arr[0]=map.get(x);
                arr[1]=i;
                break;
            }
            map.put(nums[i],i);
        }
        return arr;
    }
}
```

- Longest common prefix

```java
class Solution {
    public String longestCommonPrefix(String[] strs) {

        if(strs.length==0)
            return "";
        else if(strs.length==1)
            return strs[0];

        String w=strs[0];   // Storing first string as initial prefix
        int l=w.length(),f=0; // flag

        while(!w.equals(""))    // until w!=""
        {
            f=0;    // resetting flag each time

            for(int i=1;i<strs.length;i++)
            {
                if(strs[i].length()<w.length()) // checking if current string is smaller that the prefix
                {
                    f=1;
                    break;
                }

                String wd=strs[i].substring(0,w.length());

                if(!w.equals(wd))
                {
                    f=1;
                    break;
                }
            }

            if(f==0)    // if flag stays 0 it means prefix is ccommon in all strings
                return w;

            if(w.length()==0)   // if the length of prefix is 0 we break to avoid substring error
                break;
```

```
            w=strs[0].substring(0,l--);
        }

        return "";
    }
}
```

- Group anagrams

```
class Solution {
    public List<List<String>> groupAnagrams(String[] strs) {

        if(strs==null || strs.length==0)
            return new ArrayList<>();

        HashMap<String,List<String>> map=new HashMap<>();

        for(String s: strs)
        {
            char[] c = s.toCharArray(); // converting string to character array
            Arrays.sort(c); // sorting anagrams give common string
            String wd=new String(c);    // reverting back to string

            if(!map.containsKey(wd))    // searching if the common strings is
present if not a list is created for it
                map.put(wd,new ArrayList<>());

            map.get(wd).add(s); // adding anagrams in the key's list
        }

        return new ArrayList<>(map.values());
    }
}
```

- Remove element

```java
class Solution {
    public int removeElement(int[] nums, int val) {

        int i=0,j=nums.length-1;

        while(i<=j)
        {
            if(nums[i]==val)
            {
                if(nums[j]!=val)
                {
                    int t=nums[i];
                    nums[i]=nums[j];
                    nums[j]=t;
                    i++;
                }
                j--;
            }
            else
                i++;

        }
        return j+1;
    }
}
```

Or

```java
class Solution {
    public int removeElement(int[] nums, int val) {

        int k=0;

        for(int i=0;i<nums.length;i++)
        {
            if(nums[i]!=val)
                nums[k++]=nums[i];
        }
```

```
        return k;
    }
}
```

Or

```
class Solution {
    public int removeElement(int[] nums, int val) {

        int k=0;
        ArrayList<Integer> list=new ArrayList<>();

        for(int i: nums)
        {
            if(i!=val)
                list.add(i);
        }

        for(int i=0;i<list.size();i++)
        {
            nums[i]=list.get(i);
        }
        return list.size();
    }
}
```

- Majority element

```
class Solution {
    public int majorityElement(int[] nums) {

        HashMap<Integer,Integer> map=new HashMap<>();

        for(int i: nums)
            map.put(i,map.getOrDefault(i,0)+1);

        int max=-1,val=Integer.MIN_VALUE;
```

```
        for(int i: map.keySet())
        {
            if(map.get(i)>max)
            {
                max=map.get(i);
                val=i;
            }
        }

        return val;
    }
}
```

Or

```
public class Solution {
    public int majorityElement(int[] nums) {
        Arrays.sort(nums);
        return nums[nums.length / 2];
    }
}
```

Or

```
public class Solution {
    public int majorityElement(int[] nums) {

        int candidate=0,count=0;    // Moore voting algorithm

        for(int i: nums)
        {
            if(count==0)
                candidate=i;
            count=count+((i==candidate)? 1 : -1);
        }
```

```
        return candidate;
    }
}
```

-