Arrays & Hashing

- Concatenation of Array

```java
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

```java
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

```
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

```
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

```
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;
    }
}
```

- Design hashset

```
class MyHashSet {

    ArrayList<Integer> list;    // accessible arraylist

    public MyHashSet() {
        list=new ArrayList<>(); // created the list
    }

    public void add(int key) {
        if(!list.contains(key))
            list.add(key);
    }

    public void remove(int key) {
        list.remove(Integer.valueOf(key));  // otherwise it treats
key as an index
    }

    public boolean contains(int key) {
        return list.contains(key);
    }
}
```

Or

```
public class MyHashSet {
```

```
    private boolean[] data;

    public MyHashSet() {
        data = new boolean[1000001];
    }

    public void add(int key) {
        data[key] = true;
    }

    public void remove(int key) {
        data[key] = false;
    }

    public boolean contains(int key) {
        return data[key];
    }
}
```

- Design HashMap

```
class MyHashMap {

    int[] arr;
    public MyHashMap() {
        arr=new int[1000001];
        Arrays.fill(arr,-1);
    }

    public void put(int key, int value) {
        arr[key]=value;
    }
```

```
    public int get(int key) {
        return arr[key];
    }

    public void remove(int key) {
        arr[key]=-1;
    }
}
```

- Sort an array

```
class Solution {
    public int[] sortArray(int[] nums) {
        int l=nums.length;
        mergesort(nums,0,l-1);
        return nums;
    }

    static void mergesort(int[] arr,int l, int r)
    {
        if(l>=r)
            return;

        int mid=(l+r)/2;
        mergesort(arr,l,mid);
        mergesort(arr,mid+1,r);
        merge(arr,l,mid,r);
    }

    static void merge(int[] arr,int l,int mid,int r)
    {
        int n1,n2;
```

```
        n1=mid-l+1;
        n2=r-mid;

        int a[]=new int[n1];
        int b[]=new int[n2];

        for(int i=0;i<n1;i++)
            a[i]=arr[l+i];

        for(int i=0;i<n2;i++)
            b[i]=arr[mid+i+1];

        int i=0,j=0,k=l;

        while(i<n1 && j<n2)
        {
            if(a[i]<b[j])
                arr[k++]=a[i++];
            else
                arr[k++]=b[j++];
        }

        while(i<n1)
            arr[k++]=a[i++];
        while(j<n2)
            arr[k++]=b[j++];
    }
}
```

- Sort colors

```java
class Solution {
    public void sortColors(int[] nums) {
        int r=0,w=0,b=nums.length-1;    // 3 pointers

        while(w<=b)
        {
            if(nums[w]==0)  // 0 always left
            {
                int temp=nums[w];
                nums[w]=nums[r];
                nums[r]=temp;
                w++;
                r++;
            }
            else if(nums[w]==1) // 1 always mid
                w++;
            else    // 2 always right
            {
                int temp=nums[w];
                nums[w]=nums[b];
                nums[b]=temp;
                b--;
            }
        }
    }
}
```

- Top K frequent elements

```java
class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        HashMap<Integer,Integer> map=new HashMap<>();
```

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

        int[] n=new int[map.size()];
        int a=0;

        for(int i:map.keySet())
        {
            n[a++]=map.get(i);
        }

        Arrays.sort(n);

        a=n.length-k;
        int[] ans=new int[k];
        int x=0;

        for(int i=a;i<n.length;i++)
        {
            for(int j:map.keySet())
            {
                if(map.get(j)==n[i])
                {
                    ans[x++]=j;
                    map.remove(j);
                    break;
                }
            }
        }

        return ans;
    }
```

```
}
```

Or

```
class Solution {
    public int[] topKFrequent(int[] nums, int k) {

        // Step 1: Count frequency of each number using HashMap
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i : nums) {
            map.put(i, map.getOrDefault(i, 0) + 1);
        }

        // Step 2: Create buckets where index = frequency
        // Each bucket[i] holds a list of numbers that appear
exactly i times
        ArrayList<Integer>[] bucket = new ArrayList[nums.length +
1];

        // Step 3: Fill buckets based on frequency
        for (int key : map.keySet()) {
            int freq = map.get(key);
            if (bucket[freq] == null) {
                bucket[freq] = new ArrayList<>();
            }
            bucket[freq].add(key);
        }

        // Step 4: Collect top K frequent elements starting from
highest frequency
        List<Integer> res = new ArrayList<>();
```

```
        for (int i = bucket.length - 1; i >= 0 && res.size() < k; i--) {
            if (bucket[i] != null)
                res.addAll(bucket[i]);  // Add all numbers with this frequency
        }

        // Step 5️⃣: Convert result list to int[] for the final answer
        int[] ans = new int[k];
        for (int i = 0; i < k; i++)
            ans[i] = res.get(i);

        return ans; // ✅ O(n) overall time complexity
    }
}
```

- Encode and Decode strings

```
class Solution {

    public String encode(List<String> strs) {
        String wd="";
        for(String s:strs)
        {
            wd+=s+"/";
        }
        return wd;
    }

    public List<String> decode(String str) {
```

```
        ArrayList<String> wd=new ArrayList<>();
        String w="";
        for(int i=0;i<str.length();i++)
        {
            char ch=str.charAt(i);
            if(ch!='/')
                w+=ch;
            else
            {
                wd.add(w);
                w="";
            }
        }
        return wd;
    }
}
```

Or

```
class Solution {

    public String encode(List<String> strs) {
        StringBuilder sb = new StringBuilder();
        for (String s : strs) {
            sb.append(s.length()).append('#').append(s);
        }
        return sb.toString();
    }

    public List<String> decode(String str) {
        List<String> result = new ArrayList<>();
```

```
        int i = 0;
        while (i < str.length()) {
            int j = i;
            while (str.charAt(j) != '#') j++;
            int len = Integer.parseInt(str.substring(i, j));
            j++;
            result.add(str.substring(j, j + len));
            i = j + len;
        }
        return result;
    }

}
```

- Range sum query 2d immutable

```
class NumMatrix {

    int[][] matrix;

    public NumMatrix(int[][] matrix) {
        this.matrix=matrix;;
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        int sum=0;
        for(int i=row1;i<=row2;i++)
        {
            for(int j=col1;j<=col2;j++)
            {
                sum+=matrix[i][j];
```

```
            }
        }
        return sum;
    }
}
```

Or

```
class NumMatrix {

    int [][] prefix;
    public NumMatrix(int[][] matrix) {
        int r=matrix.length,c=matrix[0].length;
        prefix=new int[r+1][c+1];
        for(int i=1;i<=r;i++)
        {
            for(int j=1;j<=c;j++)
            {
                prefix[i][j]=matrix[i-1][j-1]
                            +prefix[i-1][j]
                            +prefix[i][j-1]
                            -prefix[i-1][j-1];
            }
        }
    }

    public int sumRegion(int row1, int col1, int row2, int col2) {
        return prefix[row2+1][col2+1]
              - prefix[row1][col2 + 1]
              - prefix[row2 + 1][col1]
              + prefix[row1][col1];
```

```
    }
}
```

- Products of arrays except self

```
class Solution {
    public int[] productExceptSelf(int[] nums) {
        int p=1,zero=0;
        for(int i:nums)
        {
            if(i!=0)
                p*=i;
            else
                zero++;
        }

        for(int i=0;i<nums.length;i++)
        {
            if(zero>1)
                nums[i]=0;
            else if(zero==1)
                nums[i]=(nums[i]==0) ? p : 0;
            else
                nums[i]=p/nums[i];
        }

        return nums;
    }
}
```

- Valid sudoku

```
class Solution {
    public boolean isValidSudoku(char[][] board) {
        // row check
        for(int i=0;i<9;i++)
        {
            if(checkRow(board[i])==false)
                return false;
        }
        // column check
        for(int i=0;i<9;i++)
        {
            if(checkCol(board,i)==false)
                return false;
        }
        // sub-boxes check
        for(int i=0;i<9;i++)
        {
            if(checkBox(board,i)==false)
                return false;
        }

        return true;
    }

    private static boolean checkRow(char[] a)
    {
        boolean[] f=new boolean[10];
        for(char c: a)
        {
            if(c=='.')
                continue;
```

```
            int digit=c-'0';
            if(f[digit]==true)
                return false;
            f[digit]=true;
        }
        return true;
    }

    private static boolean checkCol(char[][] a,int j)
    {
        boolean[] f=new boolean[10];
        for(int i=0;i<9;i++)
        {
            char c=a[i][j];
            if(c=='.')
                continue;
            int digit=c-'0';
            if(f[digit]==true)
                return false;
            f[digit]=true;
        }
        return true;
    }

    private static boolean checkBox(char[][] a,int b)
    {
        int i=3*(b/3);
        int j=3*(b%3);

        boolean[] f=new boolean[10];
        for(int x=i;x<(i+3);x++)
        {
            for(int y=j;y<(j+3);y++)
            {
```

```
                char c=a[x][y];
                if(c=='.')
                    continue;
                int digit=c-'0';
                if(f[digit]==true)
                    return false;
                f[digit]=true;
            }
        }
        return true;
    }
}
```

Or

```
class Solution {
    public boolean isValidSudoku(char[][] board) {
        boolean[][] r=new boolean[9][9];
        boolean[][] c=new boolean[9][9];
        boolean[][] b=new boolean[9][9];

        for(int i=0;i<9;i++)
        {
            for(int j=0;j<9;j++)
            {
                char ch=board[i][j];

                if(ch=='.')
                    continue;

                int num=ch-'1';   // 0 based indexing
```

```
                int box=(i/3)*3+(j/3);

                if(r[i][num] || c[j][num] || b[box][num])
                    return false;
                r[i][num] = c[j][num] = b[box][num] = true;
            }
        }
        return true;
    }
}
```

- Longest consecutive sequence

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

        HashSet<Integer> set=new HashSet<>();
        for(int i:nums)
            set.add(i);


        int max=0;

        for(int i:set)
        {
            if(set.contains(i-1)==false)
            {
                int curr=i;
                int streak=1;
```

```
                while(set.contains(curr+1))
                {
                    curr++;
                    streak++;
                }
                max=Math.max(max,streak);
            }

        }

        return max;
    }
}
```

- Best time to buy and sell stock – II

```
class Solution {
    public int maxProfit(int[] prices) {
        int profit=0;
        for(int i=0;i<prices.length-1;i++)
        {
            if(prices[i]<prices[i+1])
                profit+=prices[i+1]-prices[i];
        }
        return profit;
    }
}
```

- Majority element II

```java
class Solution {
    public List<Integer> majorityElement(int[] nums) {

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

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

        int n=nums.length;
        ArrayList<Integer> list=new ArrayList<>();

        for(int i:map.keySet())
        {
            if(map.get(i)>(n/3))
                list.add(i);
        }

        return list;
    }
}
```

Or

```java
class Solution {
    public List<Integer> majorityElement(int[] nums) {
        int c1=0,c2=0,e1=0,e2=0;    //  Atmost 2 candidates can be there

        // Boyer Moore Voting algo
```

```
        for(int i:nums)
        {
            if(i==e1)
                c1++;
            else if(i==e2)
                c2++;
            else if(c1==0)
            {
                e1=i;
                c1=1;
            }
            else if(c2==0)
            {
                e2=i;
                c2=1;
            }
            else
            {
                c1--;
                c2--;
            }
        }

        c1=0;
        c2=0;
        for(int i:nums)
        {
            if(i==e1)
                c1++;
            else if(i==e2)
                c2++;
        }
```

```
        ArrayList<Integer> list=new ArrayList<>();
        int n=nums.length/3;
        if(c1>n)
            list.add(e1);
        if(c2>n)
            list.add(e2);

        return list;
    }
}
```

- Subarray sum equals k

```
class Solution {
    public int subarraySum(int[] nums, int k) {
        int c=0;
        HashMap<Integer,Integer> map=new HashMap<>();
        map.put(0,1);   // for index 0 subarray

        int presum=0;
        for(int i:nums)
        {
            presum+=i;
            // Check if there exists a prefixSum such that
currentSum - oldSum = k
            if(map.containsKey(presum-k))
                c+=map.get(presum-k);
            map.put(presum,map.getOrDefault(presum,0)+1);
        }

        return c;
```

```
    }
}
```

- First missing positive

```
class Solution {
    public int firstMissingPositive(int[] nums) {
        Arrays.sort(nums);
        int miss=1;
        for(int i:nums)
        {
            if(i>0 && i==miss)
                miss++;
        }
        return miss;
    }
}
```

Or

```
class Solution {
    public int firstMissingPositive(int[] nums) {
        // Pigeon hole principle

        // Replace invalid numbers with a dummy value (n+1)
        int n=nums.length;
        for(int i=0;i<n;i++)
        {
            if(nums[i]<=0 || nums[i]>n)
```

```
                nums[i]=n+1;
        }

        // Mark presence of numbers within [1, n]
        for(int i=0;i<n;i++)
        {
            int val=Math.abs(nums[i]);  // Absolute value if
marked earlier
            if(val==n+1)
                continue;
            int seat=val-1;
            if(nums[seat]>0)
                nums[seat]=-nums[seat];
        }

        // First positive index + 1 is the missing number
        for(int i=0;i<n;i++)
        {
            if(nums[i]>0)
                return i+1;
        }

        // All numbers 1..n present
        return n+1;
    }
}
```