Two Pointers

- Reverse string

```
class Solution {
    public void reverseString(char[] s) {

        int left=0,right=s.length-1;

        while(left<=right)
        {
            char temp=s[left];
            s[left]=s[right];
            s[right]=temp;

            left++;
            right--;
        }

        System.out.println(s);
    }
}
```

- Valid palindrome

```
class Solution {
    public boolean isPalindrome(String s) {
        String wd="";

        int l=0,r=s.length()-1;

        // Removing all non alpha numeric characters from the
input string
        while(l<=r)
        {
            char ch=s.charAt(l);
            if(Character.isLetterOrDigit(ch))
                wd+=ch;
            l++;
        }

        s=""+wd;
        wd="";

        l=0;
        r=s.length()-1;

        while(l<=r)
        {
            char ch=s.charAt(r);
            wd+=ch;
            r--;
        }

        wd= wd.toLowerCase();
        s=s.toLowerCase();

        return (wd.equals(s));
    }
}
```

Or

```
class Solution {
    public boolean isPalindrome(String s) {

        String wd="";

        int l=0,r=s.length()-1;

        // Removing all non alpha numeric characters from the
input string
        while(l<=r)
        {
            char ch=s.charAt(l);
            if(Character.isLetterOrDigit(ch))
                wd+=Character.toLowerCase(ch);
            l++;
        }

        l=0;
        r=wd.length()-1;

        while(l<=r)
        {
            if(wd.charAt(l)!=wd.charAt(r))
                return false;

            l++;
            r--;
        }


        return true;
    }
}
```

- Valid palindrome II

```
class Solution {
    public boolean validPalindrome(String s) {
        int c=0,l=0,r=s.length()-1;

        while(l<=r)
        {
            // Try skipping either left or right char
            if(s.charAt(l)!=s.charAt(r))
                return ispalin(s,l+1,r) || ispalin(s,l,r-1);

            l++;
            r--;
        }

        return true;
    }

    private boolean ispalin(String s,int l,int r)
    {
        while(l<=r)
        {
            if(s.charAt(l)!=s.charAt(r))
                return false;

            l++;
            r--;
        }
        return true;
    }
}
```

- Merge strings alternately

```
class Solution {
    public String mergeAlternately(String word1, String word2) {

        String str="";
        int i=0,j=0,idx=0;

        while(i<word1.length() && j<word2.length())
        {
            if(idx%2==0)
                str+=word1.charAt(i++);
            else
                str+=word2.charAt(j++);

            idx++;
        }

```

```java
        // if word1 is remaining
        while(i<word1.length())
            str+=word1.charAt(i++);

        // if word2 is remaining
        while(j<word2.length())
            str+=word2.charAt(j++);

        return str;
    }
}
```

Or

```java
public class Solution {
    public String mergeAlternately(String word1, String word2) {
        StringBuilder res = new StringBuilder();
        int i = 0, j = 0;
        while (i < word1.length() && j < word2.length()) {
            res.append(word1.charAt(i++));
            res.append(word2.charAt(j++));
        }
        res.append(word1.substring(i));
        res.append(word2.substring(j));
        return res.toString();
    }
}
```

Or

```java
public class Solution {
    public String mergeAlternately(String word1, String word2) {
        int n = word1.length(), m = word2.length();
        StringBuilder res = new StringBuilder();
        int i = 0, j = 0;
        while (i < n || j < m) {
            if (i < n) res.append(word1.charAt(i++));
            if (j < m) res.append(word2.charAt(j++));
        }
        return res.toString();
    }
}
```

- Merge Sorted array

```
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {

        int i=m-1,j=n-1,k=m+n-1;

        // considering largest element first and pushing it in
the end
        while(i>=0 && j>=0)
        {
            if(nums1[i]>nums2[j])
                nums1[k--]=nums1[i--];
            else
                nums1[k--]=nums2[j--];
        }

        while(j>=0)
```

```
            nums1[k--]=nums2[j--];
    }
}
```

- Remove duplicates from sorted array

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int c=0;    // Slow pointer -- Unique element

        for(int i=1;i<nums.length;i++)
        {
            if(nums[i]!=nums[c])
            {
                c++;
                nums[c]=nums[i];
            }
        }
        return c+1;
    }
}
```

- Two integer sum II

```
class Solution {
    public int[] twoSum(int[] numbers, int target) {
        int i=0,j=numbers.length-1;

        while(i<j)
        {
            int x=numbers[i]+numbers[j];
            if(x == target)
                break;
            else if(x > target) // It means largest element is
too big
                j--;
            else    // It means smalles element is too small
                i++;
        }

        return new int[]{i+1, j+1};
    }
}
```

- 3 sum

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> l=new ArrayList<>();
        HashSet<List<Integer>> h=new HashSet<>();

        Arrays.sort(nums);  // Array needs to be sorted for
pointers tracking
```

```
        for(int i=0;i<nums.length;i++)
        {
            int a=i+1,b=nums.length-1;

            while(a<b)  // Just like 2 integer sum - previous
question
            {
                if(nums[i]+nums[a]+nums[b] == 0)
                {
                    List<Integer> ll=new ArrayList<>();
                    ll.add(nums[i]);
                    ll.add(nums[a]);
                    ll.add(nums[b]);
                    h.add(ll);  // adding to hashset to eleminate
duplicates
                    a++;
                }
                else if(nums[i]+nums[a]+nums[b] < 0)
                    a++;
                else
                    b--;
            }
        }

        for(List<Integer> t:h)
        {
            l.add(t);
        }

        return l;
    }
}
```

- 4 sum

```
class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> l=new ArrayList<>();
        HashSet<List<Integer>> h=new HashSet<>();

        Arrays.sort(nums);  // Array needs to be sorted for
pointers tracking

        for(int i=0;i<nums.length;i++)
        {
            for(int j=i+1;j<nums.length;j++)
            {
                int a=j+1,b=nums.length-1;

                while(a<b)
                {
                    long sum = (long) nums[i] + nums[j] + nums[a]
+ nums[b];
                    if(sum == target)
                    {
                        List<Integer> ll=new ArrayList<>();
                        ll.add(nums[i]);
                        ll.add(nums[j]);
                        ll.add(nums[a]);
                        ll.add(nums[b]);
                        h.add(ll);  // adding to hashset to 
eleminate duplicates
                        a++;
                    }
                    else if(sum < target)
                        a++;
                    else
                        b--;
                }

            }

        }

        for(List<Integer> t:h)
        {
            l.add(t);
        }

        return l;
    }
}
```

- Rotate array

```
class Solution {
    public void rotate(int[] nums, int k) {
        int n = nums.length;
        k = k % n;  // handle cases where k > n

        reverse(nums,0,n-k-1);    //  4,3,2,1,5,6,7,8
        reverse(nums,n-k,n-1);  //  4,3,2,1,8,7,6,5
        reverse(nums,0,n-1);  //  5,6,7,8,1,2,3,4
    }
    private void reverse(int[] arr,int i,int j)
    {
        while(i<j)
        {
            int t=arr[i];
            arr[i]=arr[j];
            arr[j]=t;
            i++;
            j--;
        }
    }
}
```

- Container with most water

```
class Solution {
    public int maxArea(int[] heights) {
        int i=0,j=heights.length-1;

        int max=0;
        while(i<j)
        {
            int area=(j-i)*Math.min(heights[i],heights[j]);

            if(area>max)
                max=area;

            // Moving the pointer where the height is small to
increase the area
            if(heights[i]<=heights[j])
                i++;
            else
                j--;
        }

        return max;
    }
}
```

- Boats to save people

```
class Solution {
    public int numRescueBoats(int[] people, int limit) {
        Arrays.sort(people);    // sorting
        int i=0,j=people.length-1,boat=0;

        while(i<=j)
        {
            if(people[i]+people[j] <= limit)
            {
                i++;    // pair lightest + heaviest
            }
                j--; // always move heaviest pointer
                boat++;
        }

        return boat;
    }
}
```

- Trapping rain water

```
class Solution {
    public int trap(int[] height) {

        int[] prefix=new int[height.length];    // Prefix maximum
        prefix[0]=0;
        for(int i=1;i<height.length;i++)
        {
            prefix[i]=Math.max(prefix[i-1],height[i-1]);
        }

        int[] suffix=new int[height.length];    // Suffix maximum
        suffix[height.length-1]=0;
        for(int i=height.length-2;i>=0;i--)
        {
            suffix[i]=Math.max(suffix[i+1],height[i+1]);
        }

        int water=0;

        for(int i=0;i<height.length;i++)
        {
            int x=Math.min(prefix[i],suffix[i])-height[i];   //
Formula
            if(x>0)
                water += x;
        }

        return water;
    }
}
```

Or

```
class Solution {
    public int trap(int[] height) {

        int water=0,leftmax=0,rightmax=0,l=0,r=height.length-1;

        while(l<r)
        {
            leftmax = Math.max(leftmax, height[l]);
            rightmax = Math.max(rightmax, height[r]);

            if(leftmax<rightmax)
            {
                water+=leftmax-height[l];
                l++;
            }
            else
            {
                water+=rightmax-height[r];
                r--;
            }
        }

        return water;
    }
}
```