Quick Sort

```
int[] A;
int[] sortArray(int[] A) {
    this.A = A;
    sort(0, A.length - 1);
    return A;
}
void sort(int l, int r) {
    if (l >= r) return;
    int p = part(l, r);
    sort(l, p - 1);
    sort(p + 1, r);
}
int part(int l, int r) {
    int p = A[r];
    //note here i is initially less than l as it will preincremented and used
    int i = l - 1;
    for (int j = i + 1; j < r; ++j)
        if (A[j] < p)
            swap(++i, j);
    swap(i + 1, r);
    return i + 1;
}
void swap(int i, int j) {
    int t = A[i];
    A[i] = A[j];
    A[j] = t;
}
```

Quick Sort two Pivot

```

```

Linked List Reversal Recursive

```
public ListNode reverseList(ListNode head) {
        if(head==null || head.next==null) return head;
        ListNode p= reverseList(head.next);
        head.next.next= head;
        head.next= null;
        return p;
    }
```

Linked List Reversal Iterative

```
public ListNode reverseList(ListNode head) {
        ListNode prev=null, next=null, cur=head;
        while(cur!=null)
        {
            next=cur.next;
            cur.next=prev;
            prev=cur;
            cur=next;
        }
        return prev;
    }
```

Merge Sort

```
void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        int n1 = m - l + 1;
        int n2 = r - m;
        /* Create temp arrays */
        int L[] = new int [n1];
        int R[] = new int [n2];
        /*Copy data to temp arrays*/
        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];
        /* Merge the temp arrays */
        // Initial indexes of first and second subarrays
        int i = 0, j = 0;
        // Initial index of merged subarry array
        int k = l;
        while (i < n1 && j < n2) {
            if (L[i] <= R[j]) {
                arr[k] = L[i];
                i++;
            }
            else {
                arr[k] = R[j];
                j++;
            }
            k++;
        }
        /* Copy remaining elements of L[] if any */
        while (i < n1) {
            arr[k] = L[i];
            i++;
            k++;
        }
        /* Copy remaining elements of R[] if any */
        while (j < n2) {
            arr[k] = R[j];
```

```
            j++;
            k++;
        }
    }

    // Main function that sorts arr[l..r] using merge()
    void sort(int arr[], int l, int r)
    {
        if (l < r) {
            // Find the middle point
            int m = (l+r)/2;
            // Sort first and second halves
            sort(arr, l, m);
            sort(arr , m+1, r);
            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }
```

Longest Substring Without Repeating Characters [LINK]

```
    public int lengthOfLongestSubstring(String s) {
        Map<Character,Integer> map= new HashMap<>();
        int b=0,e=0,counter=0,max=0;
        while(e<s.length()){
            char c= s.charAt(e);
            map.put(c, map.getOrDefault(c,0)+1);
            if(map.get(c)>1)
                counter++;
            e++;
            while(counter>0) {
                char cc= s.charAt(b);
                if(map.get(cc)>1)
                    counter--;
                map.put(cc,map.get(cc)-1);
                b++;
            }
            if(max<e-b)
                max=e-b;
        }
        return max;
    }
```

Reverse Nodes in k-Group [LINK]

```
    public ListNode reverseKGroup(ListNode head, int k) {
        ListNode cur=head;
        int count=0;
        while(cur!=null&&count!=k) {
            count++;
            cur=cur.next;
        }
```

```
        if(count==k) {
            cur=reverseKGroup(cur,k);
            while(count-->0) {
                ListNode tmp=head.next;
                head.next=cur;
                cur=head;
                head=tmp;
            }
            head=cur;
        }
        return head;
    }
```

Count and Say [LINK]

```
    public String countAndSay(int n) {
        if(n==1) return "1";
        String prev= countAndSay(n-1);
        StringBuilder sb= new StringBuilder();

        for(int i=0;i<prev.length();i++)
        {
            char c= prev.charAt(i);
            int count=0;
            while(i<prev.length()&&c==prev.charAt(i))
            {
                i++;
                count++;
            }
            i--;
            sb.append(Integer.toString(count)+c);
        }
        return sb.toString();
    }
```

Multiple Strings [LINK]

```
public String multiply(String num1, String num2) {
        if(num1.equals("0") || num2.equals("0"))
            return "0";
      int m=num1.length(), n=num2.length();
        int[] pos =  new int[m+n];

        for(int i=m-1;i>=0;i--)
            for(int j=n-1;j>=0;j--)
            {
                int mul =
Character.getNumericValue(num1.charAt(i))*Character.getNumericValue(num2.charAt(j));
                int p1=i+j, p2=i+j+1;
                int sum = mul+ pos[p2];
```

```
                pos[p1]+= sum/10;
                pos[p2]= sum%10;
            }

        StringBuilder str = new StringBuilder();

        for(int p : pos)
            if(!(str.length()==0 && p==0))
            str.append(p);

        return str.toString();
    }
```

Rotate Image LINK

```
    int n;
    public void rotate(int[][] matrix) {
        n=matrix.length;
            if(n==0) return;

            for(int i=0;i<n/2;i++)
                for(int j=i;j<n-1-i;j++)
                    swap(matrix, i,j);
            return;

    }

    public void swap(int[][] mat, int i, int j)
    {
        int tmp= mat[n-1-j][i];
        mat[n-1-j][i] = mat[n-1-i][n-1-j];
        mat[n-1-i][n-1-j] = mat[j][n-1-i];
        mat[j][n-1-i] = mat[i][j];
        mat[i][j]= tmp;
    }
```

Pow (x,n) LINK

```
  public double myPow(double x, int n) {
        if(n==0) return 1;
        double tmp= myPow(x,n/2);
        if(n%2==0)
            return  tmp*tmp;
        if(n>0)
            return tmp*tmp*x;
        return tmp*tmp/x;
    }
```

Edit Distance LINK

```
  public int minDistance(String word1, String word2) {
        int n=word1.length(), m=word2.length();
```

```
        if(n==0||m==0) return m+n;
        int[][] dp= new int[n+1][m+1];
        for(int i=0;i<=n;i++)
            dp[i][0]=i;
        for(int j=0;j<=m;j++)
            dp[0][j]=j;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=m;j++)
                if(word1.charAt(i-1)==word2.charAt(j-1))
                    dp[i][j]=dp[i-1][j-1];
                else
                    dp[i][j] = Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-1][j-1]))+1;
        return dp[n][m];
    }
```

Sort Colors LINK

```
    public void sortColors(int[] nums) {
        int s=0,e=nums.length-1;
        for(int i=0;i<nums.length;i++)
        {
            while(nums[i]==2&&i<e)
            {
                int tmp=nums[i];
                nums[i]=nums[e];
                nums[e--]=tmp;
            }

            while(nums[i]==0&&i>s)
            {
                int tmp= nums[i];
                nums[i]=nums[s];
                nums[s++]=tmp;
            }
        }
    }
```

Single Number II : Every number occurs thrice except two LINK

```
public int singleNumber(int[] nums) {
        int ans=0;
        for(int i=0;i<32;i++)
        {
            int sum=0;
            for(int num: nums)
            {
                if(((num>>i)&1)==1)
                    sum++;
            }
            sum=sum%3;
            if(sum!=0)
                ans|=(sum<<i);
```

```
        }
        return ans;
    }
```

Single Number III : All twice and two once LINK

```
public int[] singleNumber(int[] nums) {
        int totxor=0;
        for(int i:nums)
            totxor^=i;
        int hbit=0;
        for(int i=31;i>=0;i--)
        {
            if(((totxor>>i)&1)==1)
                {hbit=i;
                break;}
        }
    List<Integer> setlist= new ArrayList<>();
    List<Integer> notsetlist= new ArrayList<>();
        for(int i: nums)
        {
            if(((i>>hbit)&1)==1)
                setlist.add(i);
            else
                notsetlist.add(i);
        }

    int f=0,s=0;
        for(int i: setlist)
            f^=i;
        for(int i:notsetlist)
            s^=i;
        return new int[]{f,s};
    }
```

Remove Duplicates from a Sorted List II LINK

```
    public ListNode deleteDuplicates(ListNode head) {
        ListNode res= new ListNode(0);
        ListNode ans= res;
        res.next=head;
        ListNode prev=res,cur=head;
        while(cur!=null)
        {
            while(cur.next!=null && cur.val==cur.next.val)
                cur=cur.next;

            if(prev.next== cur)
                prev=prev.next;
            else
                prev.next=cur.next;
            cur=cur.next;
```

```
        }
        return res.next;
    }
```

Reverse Linked List II m to n LINK

```
TreeNode prev=null;
    public void flatten(TreeNode root) {
        if(root==null) return;
        flatten(root.right);
        flatten(root.left);
        root.right=prev;
        root.left=null;
        prev=root;
    }
```

Copy List with Random Pointer LINK

```
 public Node copyRandomList(Node head) {
        if(head==null) return head;
        Node ptr=head;
        while(ptr!=null)
        {
            Node clone= new Node(ptr.val);
            clone.next= ptr.next;
            ptr.next=clone;
            ptr= ptr.next.next;
        }

        ptr=head;
        while(ptr!=null)
        {
            ptr.next.random = ptr.random!=null?ptr.random.next:null;
            ptr=ptr.next.next;
        }
        ptr=head;
        Node clonehead= head.next;
        Node clone= head.next;
        while(ptr!=null)
        {
            ptr.next= clone.next;
            clone.next=  clone.next!=null?clone.next.next:null;
            ptr=ptr.next;
            clone=clone.next;
        }
        return clonehead;
    }
```

Intersections of two Lists LINK

```
public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if(headA==null||headB==null) return null;
```

```
        ListNode a=headA, b=headB;
        while(a!=b){
            if(a!=null)
                a=a.next;
            else
                a=headB;
            if(b!=null)
                b=b.next;
            else
                b=headA;
        }
        return a;
    }
```

Odd Even Linked List LINK

```
public ListNode oddEvenList(ListNode head) {
        if(head==null) return head;
        ListNode odd=head;
        ListNode evenHead=head.next;
        ListNode even=head.next;
        ListNode old=null;
        while(even!=null && even.next!=null)
        {
            old=odd;
            odd.next= even.next;
            even.next= odd.next.next;
            odd=odd.next;
            even=even.next;
        }
            odd.next=evenHead;
        return head;
    }
```

Maximum Product Subarray LINK

```
    public int maxProduct(int[] nums) {
    int max=nums[0],min=nums[0],res= nums[0];
    for(int i=1;i<nums.length;i++)
    {
        int tmp=max;
        max= Math.max(
                        Math.max(max*nums[i],min*nums[i]),
                        nums[i]);
        min= Math.min(
                        Math.min(tmp*nums[i],min*nums[i]),
                        nums[i]);
        if(res<max)
            res=max;
    }
    return res;
```

```
}
```

Majority Element LINK

```
//Boyer-Moore Voting Algorithm
public int majorityElement(int[] nums) {
   int count=0;
    int ans=0;
        for(int i:nums)
        {
            if(count==0)
                ans=i;
            if(ans==i)
                count++;
            else
                count--;
        }
    return ans;
    }
```

House Robber LINK

```
 public int rob(int[] nums) {

        // return curr;
        // if(nums.length==0) return 0;
        // if(nums.length == 1) return nums[0];
        // int[] dp = new int[nums.length];
        // dp[0]=nums[0];
        // dp[1]=Math.max(nums[0],nums[1]);
        // for(int i=2;i<nums.length;i++)
        //     dp[i] = Math.max(dp[i-1],dp[i-2]+nums[i]);
        // return dp[nums.length-1];

        // int prev=0,cur=0;
        // for(int i:nums)
        // {
        //     int tmp= cur;
        //     cur= Math.max(cur, prev+i );
        //     prev= tmp;
        // }
        // return cur;

        int last=0,lastlast=0;
        for(int i:nums)
        {
            int tmp= last;
            last= Math.max(last,lastlast+i);
            lastlast= tmp;
        }
        return last;
```

```
    }
```

Paint House LINK

```
public int minCost(int[][] costs) {
        if(costs.length==0) return 0;
        int houses= costs.length;
        int color=3;
        int[][] dp= new int[houses][color];
        for(int i=0;i<color;i++)
            dp[0][i]=costs[0][i];
        int min=0;
        for(int i=1;i<houses;i++)
            for(int j=0;j<color;j++)
            {
                if(j==0)
                    min= Math.min(dp[i-1][1],dp[i-1][2]);
                if(j==1)
                    min= Math.min(dp[i-1][2],dp[i-1][0]);
                if(j==2)
                    min= Math.min(dp[i-1][1],dp[i-1][0]);
                dp[i][j]= min+costs[i][j];
            }
        return Math.min(dp[houses-1][0],Math.min(dp[houses-1][1],dp[houses-1][2]));
    }
```

Search in a 2D Matrix II LINK

```
public boolean searchMatrix(int[][] matrix, int target) {
        int r=matrix.length-1,c=0;
        while(r>=0&&c<matrix[0].length)
        {
            // System.out.println(r+" "+c);
            if(target==matrix[r][c])
                return true;
            if(target>matrix[r][c])
                c++;
            else
                r--;
        }
        return false;
    }
```

Lowest Common Ancestor of a Binary Tree LINK

```
    TreeNode x;
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if(p==q) return q;
        helper(root,p,q);
        return x;
    }
```

```
    public int helper(TreeNode root, TreeNode p, TreeNode q){
        if(root==null) return 0;
        int tmp=0;
        if(root==p||root==q)
            tmp++;
        int l= helper(root.left, p,q);
        int r= helper(root.right, p,q);
        if(tmp+l+r>=2)
            x=root;
        return Math.max(tmp,Math.max(l,r));
    }
```

Closest Binary Search Tree Value II LINK

```
public List<Integer> closestKValues(TreeNode root, double target, int k) {
        Queue<Integer> q= new LinkedList<>();
        inorder(root,target, k, q);
        List<Integer> res= new ArrayList<>();
        while(!q.isEmpty())
            res.add(q.poll());
        return res;
    }

    public void inorder(TreeNode root, double target, int k, Queue<Integer> q)
    {
        if(root==null) return;
        inorder(root.left, target, k , q);
        if(q.size()<k)
            q.offer(root.val);
        else{
            if(Math.abs(root.val-target) <= Math.abs(q.peek()-target))
            {
                q.poll();
                q.offer(root.val);
            }
        }
        inorder(root.right, target, k , q);
    }
```

Leaves of a Binary Tree LINK

```
List<List<Integer>> res=  new ArrayList<>();
    public List<List<Integer>> findLeaves(TreeNode root) {
        helper(root);
        return res;
    }

    public int helper(TreeNode root)
    {
        if(root==null) return -1;
        int h = 1+ Math.max(helper(root.left) ,helper(root.right));
```

```
        if(res.size()==h)
            res.add(new ArrayList<>());
        res.get(h).add(root.val);
        return h;
    }
```

Reverse Linked List II [LINK]

```
public ListNode reverseBetween(ListNode head, int m, int n) {

        if(head==null) return null;
        ListNode res= new ListNode(0);
        res.next= head;
        ListNode ans= res;
        for(int i=0;i<m-1;i++)
            ans=ans.next;
        ListNode cur=ans.next;
        ListNode next= cur.next;
        for(int i=0;i<n-m;i++)
        {
            cur.next= next.next;
            next.next=ans.next;
            ans.next= next;
            next= cur.next;
        }
        return res.next;
    }
```

Unique Binary Search Trees [LINK]

```
public int numTrees(int n) {
        int[] G= new int[n+1];
        G[0]=1;
        G[1]=1;
        for(int i=2;i<=n;i++)
            for(int j=1;j<=i;j++)
                G[i]+=G[j-1]*G[i-j];
        return G[n];
     //G[n] = sigma (G[k-1]*G[n-k])
    }
```

Flatten Binary Tree to Linked List [LINK]

```
TreeNode prev=null;
    public void flatten(TreeNode root) {
        if(root==null) return;
        flatten(root.right);
        flatten(root.left);
        root.right=prev;
        root.left=null;
        prev=root;
    }
```

Counting Bits LINK

```
    public int[] countBits(int num) {
        int ans[] = new int[num+1];
        for(int i=1;i<=num;i++)
            ans[i]= ans[i&(i-1)]+1;
        return ans;
    }
```

Sum of two integer LINK

```
public int getSum(int a, int b) {
       return b==0?a:getSum(a^b,(a&b)<<1);
    }
```

Battleships in a Board LINK

```
public int countBattleships(char[][] board) {
        int count=0;
        for(int i=0;i<board.length;i++)
            for(int j=0;j<board[0].length;j++)
            {
                if(board[i][j]=='.'||(j>0&&board[i][j-1]=='X')||(i>0&&board[i-1]
[j]=='X'))
                    continue;
                count++;
            }
        return count;
    }
```

The Maze LINK

```
int n,m;
    public boolean hasPath(int[][] maze, int[] start, int[] dest) {
        n=maze.length;
        m=maze[0].length;
        Queue<int[]> q= new LinkedList<>();
        q.offer(start);
        boolean[][] visited = new boolean[n][m];
        visited[start[0]][start[1]]=true;
        int[][] moves= {{1,0},{-1,0},{0,1},{0,-1}};
```

```
        while(!q.isEmpty())
        {
            int[] cur= q.poll();
            if(cur[0]==dest[0]&&cur[1]==dest[1])
                return true;
            for(int[] m : moves)
            {
                int x = cur[0]+m[0];
                int y= cur[1]+m[1];
                while(isValid(x,y)&&maze[x][y]==0)
                {
                    x+=m[0];
                    y+=m[1];
                }
                x-=m[0];
                y-=m[1];
                if(!visited[x][y]){
                    q.offer(new int[]{x,y});
                    visited[x][y]=true;
                }
            }

        }
        return false;
    }
```

Partition to K Equal Sum Subsets [LINK]

```
 public boolean canPartitionKSubsets(int[] nums, int k) {
        int sum=0;
        for(int i: nums)
            sum+=i;
        return helper(nums, new boolean[nums.length],k,0,0,sum/k,0 );
    }

    public boolean helper(int[] nums, boolean[] visited, int k, int cursum, int curnum,
int tar, int index)
    {
        if(k==1) return true;
        if(tar==cursum&&curnum>0) return helper(nums, visited, k-1,0,0,tar,0);
        for(int i=index;i<nums.length;i++)
        {
            if(!visited[i])
            {
                visited[i]=true;
                if(helper(nums, visited, k,cursum+nums[i], curnum++,tar, i+1)) return
true;
                visited[i]= false;
            }
        }
        return false;
```

```
    }
```

Maximum Frequency Stack [LINK]

```
Map<Integer,Integer> fmap;
   Map<Integer, Stack<Integer>> stkmap;
   int max;

   public FreqStack() {
       fmap=new HashMap<>();
       stkmap= new HashMap<>();
       max=0;
   }

   public void push(int x) {
       fmap.put(x,fmap.getOrDefault(x,0)+1);
       if(max<fmap.get(x))
       {
           max= fmap.get(x);
           stkmap.put(max, new Stack<Integer>());
       }
       stkmap.get(fmap.get(x)).push(x);
   }

   public int pop() {
       int res= stkmap.get(max).pop();
       if(stkmap.get(max).isEmpty())
           max--;
       if(fmap.get(res)==1)
           fmap.remove(res);
       else
           fmap.put(res, fmap.get(res)-1);
       return res;
   }
```

Maximum Knight Moves on infinite board [LINK]

```
public int minKnightMoves(int x, int y) {
       x=Math.abs(x);
       y=Math.abs(y);
       if (x < y) {
           int t = x;
           x = y;
           y = t;
       }
        // 2 corner cases
        if(x==1 && y == 0){
           return 3;
        }
        if(x==2 && y == 2){
           return 4;
        }
```

```
        int  delta = x-y;
        if (y > delta) {
             return (int)(delta - (2*Math.floor((float)(delta-y)/3)));
        } else {
            return (int)(delta - (2*Math.floor((delta-y)/4)));
        }
    }
```

Minimum Time to Build Blocks LINK

```
public int minBuildTime(int[] blocks, int split) {
        PriorityQueue<Integer> pq= new PriorityQueue<>();
        for(int i : blocks)
            pq.offer(i);
        while(pq.size()>1)
        {
            pq.poll();
            pq.offer(pq.poll()+split);
        }
        return pq.poll();

    }
```

Ugly Number III LINK

```
 long gcd(long a, long b)
   {
        if(a==0) return b;
        return gcd(b%a,a);
   }

   long lcm(long a,long b){
        return (a*b)/gcd(a,b);
   }

   int rank(int n,int a,int b,int c){
        return (int)(
        n/a+n/b+n/c- n/lcm(a,b) - n/lcm(c,b) - n/lcm(a,c) + n/lcm(a,lcm(b,c)));
   }

  public int nthUglyNumber(int n, int a, int b, int c) {
        int l=0,r=Integer.MAX_VALUE,res=0;
        while(l<=r)
        {
            int mid= l+(r-l)/2;
            int rank = rank(mid,a,b,c);
            if(rank>=n){
                res=mid;
                r=mid-1;}
            else
                l=mid+1;
```

```
        }
        return res;
    }
```

Regular Expression Matching [LINK]

Given an input string (`s`) and a pattern (`p`), implement regular expression matching with support for `'.'` and `'*'`.

```
'.' Matches any single character.
'*' Matches zero or more of the preceding element.
```

The matching should cover the **entire** input string (not partial).

```
public boolean isMatch(String s, String p) {
            if(s==null||p==null){
                return false;
            }
            boolean[][] dp= new boolean[s.length()+1][p.length()+1];
            dp[0][0]=true;
            for(int i=0;i<p.length();i++)
                if(p.charAt(i)=='*'&&dp[0][i-1])
                    dp[0][i+1]= true;

            for(int i=0;i<s.length();i++)
                for(int j=0;j<p.length();j++)
                {
                    if(s.charAt(i)==p.charAt(j)||p.charAt(j)=='.')
                        dp[i+1][j+1]=dp[i][j];
                     if(p.charAt(j)=='*')
                    {
                        if(s.charAt(i)!=p.charAt(j-1)&&p.charAt(j-1)!='.')
                            dp[i+1][j+1]=dp[i+1][j-1];
                        else
                            dp[i+1][j+1]= dp[i][j+1]||dp[i+1][j]||dp[i+1][j-1];
                    }
                }
            return dp[s.length()][p.length()];
```

Wildcard Matching [LINK]

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'`.

```
'?' Matches any single character.
'*' Matches any sequence of characters (including the empty sequence).
```

The matching should cover the **entire** input string (not partial).

```
 public boolean isMatch(String s, String p) {
        int m=s.length(),n=p.length();
```

```
        boolean[][] dp = new boolean[m+1][n+1];
        dp[0][0] = true;
        for(int i=1;i<=n;i++)
        {
            if(p.charAt(i-1)=='*')
                dp[0][i]= true;
            else
                break;
        }

        for(int i=1;i<=m;i++)
            for(int j=1;j<=n;j++)
            {
                if(p.charAt(j-1)!='*')
                    dp[i][j]= dp[i-1][j-1] && (s.charAt(i-1)==p.charAt(j-1)||p.charAt(j-
1)=='?');
                else
                    dp[i][j]= dp[i-1][j] || dp[i][j-1];
            }
        return dp[m][n];
    }
```

Find Median from Data Stream [LINK]

```
 PriorityQueue<Integer> min,max;

    /** initialize your data structure here. */
    public MedianFinder() {
        min= new PriorityQueue<>();
        max= new PriorityQueue<>(Collections.reverseOrder());
    }

    public void addNum(int num) {
        max.offer(num);
        min.offer(max.poll());
        if(max.size()<min.size())
            max.offer(min.poll());
    }

    public double findMedian() {
        if(max.size()==min.size())
            return (max.peek()+min.peek())/2.0;
        else
            return max.peek();
    }
}
```

Sqrt [LINK]

```
public int mySqrt(int x) {
        if(x==0) return 0;
        if(x==1) return 1;
```

```
        int l=0,r=x;
       int res=0;
        while(l<=r)
        {
            int mid= (l+r)/2;
            if(mid==x/mid) return mid;
            if(mid<x/mid){
                l=mid+1;
                res=mid;
            }
            else
                r=mid-1;
        }
        return res;
    }
```

Sliding Window Maximum [LINK]

```
  public int[] maxSlidingWindow(int[] nums, int k) {
        if(nums.length ==0) return new int[]{};
        int n= nums.length;
        int[] lmax = new int[n];
        int[] rmax = new int[n];
        lmax[0]= nums[0];
        rmax[n-1] = nums[n-1];

        for(int i=1;i<n;i++)
            lmax[i]=  i%k==0? nums[i]:Math.max(lmax[i-1],nums[i]);

        for(int i=n-2;i>=0;i--)
            rmax[i]= i%k==0? nums[i]:Math.max(rmax[i+1], nums[i]);

        int[] out = new int[n-k+1];
        for(int i=0;i<n-k+1;i++)
        {
            out[i]=  Math.max(rmax[i],lmax[i+k-1]);
        }
        return out;
    }
```

Minimum value greater than given value in BST

```
int findMinforN(Node root, int N)
{
    // If leaf node reached and is smaller than N
    if (root.left == null &&
        root.right == null && root.data < N)
        return -1;

    // If node's value is greater than N and left value
    // is null or smaller then return the node value
    if ((root.data >= N && root.left == null) ||
```

```
        (root.data >= N && root.left.data < N))
        return root.data;

    // if node value is smaller than N search in the
    // right subtree
    if (root.data <= N)
        return findMinforN(root.right, N);

    // if node value is greater than N search in the
    // left subtree
    else
        return findMinforN(root.left, N);
}
```

Excel Sheet Column Title [LINK]

```
 public String convertToTitle(int n) {
         StringBuilder sb= new StringBuilder();
        while(n>0)
        {
            sb.append((char)('A'+(n-1)%26));
            n=(n-1)/26;
        }
        return sb.reverse().toString();
    }
```

Spiral Matrix [LINK]

```
public List<Integer> spiralOrder(int[][] mat) {
        List<Integer> res= new ArrayList<>();
        if(mat.length==0) return res;

        int rs=0,re=mat.length-1,cs=0,ce=mat[0].length-1;

        while(rs<=re&&cs<=ce)
        {
            for(int i=cs;i<=ce;i++)
                res.add(mat[rs][i]);
            rs++;
            for(int j=rs;j<=re;j++)
                res.add(mat[j][ce]);
            ce--;
            if(rs<=re)
                for(int i=ce;i>=cs;i--)
                    res.add(mat[re][i]);
            re--;
            if(cs<=ce)
            for(int j=re;j>=rs;j--)
                res.add(mat[j][cs]);
            cs++;
        }
```

```
        return res;
    }
```

Remove Comments LINK

```
 public List<String> removeComments(String[] s) {

        List<String> res= new ArrayList<>();
        boolean multiopen= false;
        String tmp="";

        for(String str: s)
        {
            for(int i=0;i<str.length();i++)
            {
                //single line comment
                if(i<str.length()-1&&
str.charAt(i)=='/'&&str.charAt(i+1)=='/'&&!multiopen)
                    break;
                //multiline start
                if(i<str.length()-1&&
str.charAt(i)=='/'&&str.charAt(i+1)=='*'&&!multiopen)
                {
                    multiopen=true;
                    i++;
                }
                //multiline close
                else if (i<str.length()-1&&
str.charAt(i)=='*'&&str.charAt(i+1)=='/'&&multiopen)
                {
                    multiopen=false;
                    i++;
                }
                //code
                else if(!multiopen)
                    tmp+=str.charAt(i);
            }
            if(tmp.length()!=0&&!multiopen)
                {res.add(tmp);
                tmp="";}
        }
        return res;
    }
```

Median of Two Sorted Arrays LINK

```
public double findMedianSortedArrays(int[] A, int[] B) {
    int m=A.length,n=B.length;
        if(m>n)
        {
            int[] tmp=A;
            A=B;
```

```
            B=tmp;
            int t=m;
            m=n;
            n=t;
        }

        int imin=0,imax=m,half=(m+n+1)/2;
        while(imin<=imax)
        {
            int i=(imin+imax)/2;
            int j= half-i;
            if(i<imax&&B[j-1]>A[i])
                imin=i+1;
            else if(i>imin && B[j]<A[i-1])
                imax=i-1;
            else
            {   System.out.println(i+" "+j);
                int maxleft=0;
                if(i==0)
                    maxleft=B[j-1];
                else if(j==0)
                    maxleft = A[i-1];
                else
                    maxleft= Math.max(A[i-1],B[j-1]);
                if((m+n)%2==1) return maxleft;

                int minright=0;
                if(i==m)
                    minright=B[j];
                else if(j==n)
                    minright=A[i];
                else
                    minright= Math.min(A[i],B[j]);

                return (maxleft+minright)/2.0;
            }
        }
        return 0.0;
    }
```

Longest Common Subsequence [LINK]

```
    public int longestCommonSubsequence(String a, String b) {
        int[][] dp= new int[a.length()+1][b.length()+1];

        for(int i=1;i<=a.length();i++)
            for(int j=1;j<=b.length();j++)
            {
                if(a.charAt(i-1)==b.charAt(j-1))
                    dp[i][j]= 1+dp[i-1][j-1];
                else
                    dp[i][j]= Math.max(dp[i][j-1],dp[i-1][j]);
            }
```

```
        return dp[a.length()][b.length()];
    }
```

Longest Increasing Subsequence LINK

```
public int lengthOfLIS(int[] nums) {
    if(nums.length==0)
        return 0;
       int[] dp= new int[nums.length];
       dp[0]=1;
       int maxans=1;
       for(int i=1;i<dp.length;i++)
       {
           int maxval=0;
           for(int j=0;j<i;j++)
           {
               if(nums[i]>nums[j]){
                   maxval= Math.max(dp[j], maxval);
               }
           }
           dp[i]=maxval+1;
           maxans= Math.max(maxans, dp[i]);
       }
       return maxans;
    }
```

Partition Equal Subset Sum LINK

```
public boolean canPartition(int[] nums) {
       int sum=0;
       for(int i:nums)
           sum+=i;
       if(sum%2==1)
           return false;
       sum/=2;
       boolean dp[]= new boolean[sum+1];
       dp[0]=true;
       for(int num:nums)
           for(int i=sum;i>=0;i--)
               if(i>=num)
                   dp[i]|=dp[i-num];
       return dp[sum];
    }
```

Partition a set into two subsets such that the difference of subset sums is minimum

```
static int findMin(int arr[], int n)
    {
        // Calculate sum of all elements
        int sum = 0;
        for (int i = 0; i < n; i++)
```

```
            sum += arr[i];

        // Create an array to store
        // results of subproblems
        boolean dp[][] = new boolean[n + 1][sum + 1];

        // Initialize first column as true.
        // 0 sum is possible  with all elements.
        for (int i = 0; i <= n; i++)
            dp[i][0] = true;

        // Initialize top row, except dp[0][0],
        // as false. With 0 elements, no other
        // sum except 0 is possible
        for (int i = 1; i <= sum; i++)
            dp[0][i] = false;

        // Fill the partition table
        // in bottom up manner
        for (int i = 1; i <= n; i++)
        {
            for (int j = 1; j <= sum; j++)
            {
                // If i'th element is excluded
                dp[i][j] = dp[i - 1][j];

                // If i'th element is included
                if (arr[i - 1] <= j)
                    dp[i][j] |= dp[i - 1][j - arr[i - 1]];
            }
        }

        // Initialize difference of two sums.
        int diff = Integer.MAX_VALUE;

        // Find the largest j such that dp[n][j]
        // is true where j loops from sum/2 t0 0
        for (int j = sum / 2; j >= 0; j--)
        {
            // Find the
            if (dp[n][j] == true)
            {
                diff = sum - 2 * j;
                break;
            }
        }
        return diff;
    }
```

Serialize and Deserialize N-ary Tree [LINK]

```
// Encodes a tree to a single string.
    public String serialize(Node root) {
```

```
        List<String> lst= new ArrayList<>();
        serializeHelper(root,lst);
        return String.join(",",lst);
    }

    public void serializeHelper(Node root, List<String> lst)
    {
        if(root==null)
            return;
        lst.add(Integer.toString(root.val));
        lst.add(Integer.toString(root.children.size()));
        for(Node n: root.children)
            serializeHelper(n,lst);
    }

    // Decodes your encoded data to tree.
    public Node deserialize(String data) {
        if(data.length()==0) return null;
        Queue<String> q= new LinkedList<>(Arrays.asList(data.split(",")));
        return deserializeHelper(q);
    }

    public Node deserializeHelper(Queue<String> q)
    {
        int val= Integer.parseInt(q.poll());
        int size= Integer.parseInt(q.poll());
        List<Node> children= new ArrayList<>();
        while(size-->0)
            children.add(deserializeHelper(q));
        return new Node(val, children);
    }
```

Implement Rand10() Using Rand7() LINK

```
  public int rand10() {
        int res=40;
        while(res>=40) // will stop only when something is in range [1-40] as we want
each integr [1-10] to be equally probable
            res = 7*rand7()-rand7(); // will randomly generate number between [1-49]
        return res%10+1;
    }
```

Alien Dictionary LINK

```
  public String alienOrder(String[] words) {

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

        for(String s: words)
            for(char c: s.toCharArray())
                order.put(c,0);
```

```
        for(int i = 0;i<words.length-1;i++)
        {
            char[] cur= words[i].toCharArray();
            char[] next= words[i+1].toCharArray();
            int min= Math.min(cur.length,next.length);
            for(int j=0;j<min;j++)
            {
                if(cur[j]!=next[j]){
                    char c1= cur[j];
                    char c2= next[j];
                    Set<Character> set= map.getOrDefault(c1,new HashSet<>());
                    if(!set.contains(c2)){
                        set.add(c2);
                        map.put(c1,set);
                        order.put(c2, order.get(c2)+1);
                    }
                    break;
                }
            }
        }
        Queue<Character> q= new LinkedList<>();
        for(char c: order.keySet())
            if(order.get(c)==0)
                q.offer(c);

        String res="";

        while(!q.isEmpty()){
            char c = q.poll();
            res+=c;
            if(map.containsKey(c)){
                for(char cc: map.get(c))
                {
                    order.put(cc, order.get(cc)-1);
                    if(order.get(cc)==0)
                        q.offer(cc);
                }
            }
        }
        if(res.length()!=order.size())
            return "";
        return res;
    }
```

Substring with Concatenation of All Words [LINK]

```
public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> res= new ArrayList<>();
        if(words.length==0||s.length()==0) return res;
        Map<String, Integer> map= new HashMap<>();
        for(String str: words)
            map.put(str, map.getOrDefault(str,0)+1);
```

```
        int num= words.length, n = s.length();
        int len= words[0].length();
        for(int i=0;i<n- num*len +1;i++)
        {
            Map<String, Integer> seen= new HashMap<>();
            int j=0;
            while(j<num)
            {
                String cur= s.substring(i+j*len,i+(j+1)*len);
                if(map.containsKey(cur))
                {
                    seen.put(cur, seen.getOrDefault(cur, 0)+1);
                    if(seen.get(cur)>map.getOrDefault(cur,0))
                        break;
                }
                else
                    break;
                j++;
            }
            if(j==num)
                res.add(i);
        }
        return res;
    }
```

Search in Rotated Sorted Array [LINK]

```
public int search(int[] nums, int tar) {
        if(nums.length==0) return -1;
        int rm= realMid(nums), s=0, e= nums.length-1, mid, rmid;
        while(s<=e)
        {
            mid= (s+e)/2;
            rmid= (mid+rm)%nums.length;
            if(nums[rmid]==tar)
                return rmid;
            if(nums[rmid]<tar)
                s=mid+1;
            else
                e=mid-1;
        }
        return -1;
    }

    public int realMid(int[] nums)
    {
        int s=0,e=nums.length-1;
        while(s<e)
        {
            int mid= (s+e)/2;
            if(nums[mid]>nums[e])
                s=mid+1;
            else
```

```
                e=mid;
            }
            return s;
        }
```

SubArray sum equal K [LINK]

```
public int subarraySum(int[] nums, int k) {
        Map<Integer, Integer> map= new HashMap<>();
        int sum=0;
        map.put(0,1);
        int count=0;
        for(int i: nums){
            sum+=i;
            if(map.containsKey(sum-k))
                count+=map.get(sum-k);
            map.put(sum, map.getOrDefault(sum, 0)+1);
        }
        return count;
    }
```

Max Chunks To Make Sorted [LINK]

```
public int maxChunksToSorted(int[] arr) {
        int max=-1;
        int count=0;
        for(int i=0;i<arr.length;i++)
        {
            max= Math.max(max, arr[i]);
            if(max==i)
                count++;
        }
        return count;
    }
```

Wiggle Sort [LINK]

```
 public void wiggleSort(int[] nums) {
        boolean less=true;
        for(int i=0;i<nums.length-1;i++)
        {
            if(less){
                if (nums[i]>nums[i+1])
                swap(nums, i, i+1);}
            else if(nums[i]<nums[i+1])
                swap(nums,i,i+1);
            less=!less;
        }
    }

    public void swap(int[] nums, int i ,int j)
    {
```

```
            int t=nums[i];
            nums[i]=nums[j];
            nums[j]=t;
        }
```

Find Minimum in Rotated Sorted Array II LINK

```
    public int findMin(int[] nums) {
            int s=0,e=nums.length-1;
            while(s<e)
            {
                int mid=(s+e)/2;
                if(nums[mid]>=nums[e])
                    s=mid+1;
                else
                    e=mid;
            }
            return nums[e];
        }
```

Paint Fence LINK

```
     public int numWays(int n, int k) {
            if(n==0) return 0;
            if(n==1) return k;
            int sameCol= k;
            int diffCol= k*(k-1);
            for(int i=2;i<n;i++)
            {
                int tmp= diffCol;
                diffCol= (diffCol+sameCol)*(k-1);
                sameCol=tmp;
            }
            return diffCol+sameCol;
        }
```

Coin Change 2 LINK

You are given coins of different denominations and a total amount of money. Write a function to compute the number of combinations that make up that amount. You may assume that you have infinite number of each kind of coin.

```
public int change(int amount, int[] coins) {
        int[][] dp= new int[coins.length+1][amount+1];
        dp[0][0]= 1;
        for(int i=1;i<=coins.length;i++){
            dp[i][0]=1;
            for(int j=1;j<=amount;j++)
                dp[i][j] = dp[i-1][j] + (j>=coins[i-1]?dp[i][j-coins[i-1]]:0);
        }
        return dp[coins.length][amount];
    }
```

READ WRITE lock

The rules for read access are implemented in the `lockRead()` method. All threads get read access unless there is a thread with write access, or one or more threads have requested write access.

The rules for write access are implemented in the `lockWrite()` method. A thread that wants write access starts out by requesting write access (`writeRequests++`). Then it will check if it can actually get write access. A thread can get write access if there are no threads with read access to the resource, and no threads with write access to the resource. How many threads have requested write access doesn't matter.

```
public class ReadWriteLock{

  private int readers       = 0;
  private int writers       = 0;
  private int writeRequests = 0;

  public synchronized void lockRead() throws InterruptedException{
    while(writers > 0 || writeRequests > 0){
      wait();
    }
    readers++;
  }

  public synchronized void unlockRead(){
    readers--;
    notifyAll();
  }

  public synchronized void lockWrite() throws InterruptedException{
    writeRequests++;

    while(readers > 0 || writers > 0){
      wait();
    }
    writeRequests--;
    writers++;
  }

  public synchronized void unlockWrite() throws InterruptedException{
    writers--;
```

```
        notifyAll();
      }
    }
```

Maximum Sum of Two Non-Overlapping Subarrays LINK

```
public int maxSumTwoNoOverlap(int[] A, int L, int M) {
        for(int i=1;i<A.length;i++)
            A[i]+=A[i-1];
        int Lmax= A[L-1], Mmax= A[M-1], res= A[L+M-1];
        for(int i=L+M;i<A.length;i++)
        {
            Lmax= Math.max(Lmax, A[i-M]-A[i-L-M]);
            Mmax= Math.max(Mmax, A[i-L]-A[i-L-M]);
            res= Math.max(res, Math.max(Lmax+ A[i]-A[i-M], Mmax+ A[i]-A[i-L]));
        }
        return res;
    }
```

N Queen LINK

```
class Solution {
    int n;
    Set<Integer> row, col, diag ,adiag;
    int[] queens;
    List<List<String>> res= new ArrayList<>();
    public List<List<String>> solveNQueens(int n) {
        this.n= n;
        row= new HashSet<>();
        col= new HashSet<>();
        diag= new HashSet<>();
        adiag= new HashSet<>();
        queens= new int[n];
        backtrack(0);
        return res;
    }

    public void backtrack(int row)
    {
        for(int col=0;col<n;col++)
        {
            if(isOk(row,col))
            {
                placeQueen(row,col);
                if(row==n-1)
                    addSolution();
                else
                    backtrack(row+1);
                removeQueen(row,col);
            }
        }
    }
```

```
    public boolean isOk(int row, int col)
    {

 if(this.row.contains(row)||this.col.contains(col)||diag.contains(row+col)||adiag.contai
ns(row-col))
            return false;
        return true;
    }

    public void placeQueen(int row, int col)
    {
        queens[row]= col;
        this.row.add(row);
        this.col.add(col);
        diag.add(row+col);
        adiag.add(row-col);
    }


    public void removeQueen(int row, int col)
    {
        queens[row]= 0;
        this.row.remove(row);
        this.col.remove(col);
        diag.remove(row+col);
        adiag.remove(row-col);
    }

    public void addSolution(){
        List<String> lst=new ArrayList<>();
        for(int i=0;i<n;i++)
        {
            int col= queens[i];
            StringBuilder sb= new StringBuilder();
            for(int j=0;j<col;j++) sb.append(".");
            sb.append("Q");
            for(int j=0;j<n-col-1;j++) sb.append(".");
            lst.add(sb.toString());
        }
        res.add(lst);
    }

}
```