Index - 3

1. Count number of subset with given sum

```
public int perfectSum(int a[], int n, int sum)
{
   Arrays.sort(a);
   int dp[][] = new int[n+1][sum+1];

   int mod = 1000000007;

   for(int i=0;i<dp[0].length ;i++)
      dp[0][i] = 0;

   for(int i=0;i<dp.length; i++)
      dp[i][0] = 1;


   for(int i=1;i<=n;i++){
      for(int j=1;j<=sum; j++){
         if(j<a[i-1])
            dp[i][j] = dp[i-1][j];

         else
            dp[i][j] = (dp[i-1][j]%mod + dp[i-1][j-a[i-1]]%mod)%mod;
      }
   }
   return dp[n][sum];
}
```

t.c = n X sum
s.c = n X sum

2. Partition Equal Subset Sum

```
static int equalPartition(int n, int a[])
    {
        int sum = 0;
        for(int i=0;i<n;i++)
            sum = sum + a[i];

        if((sum&1)!=0)
            return 0;

        sum = sum/2;
        return checkTheSum(a, n, sum)?1:0;
    }
    static boolean checkTheSum(int a[],int n, int sum){
        boolean dp[][] = new boolean[n+1][sum+1];

        for(int i=0;i<=sum ;i++)
            dp[0][i] = false;

        for(int i=0;i<=n;i++)
            dp[i][0] = true;

        for(int i=1;i<=n;i++){
            for(int j=1;j<=sum; j++){
                if(j<a[i-1])
                    dp[i][j] = dp[i-1][j];
                else
                    dp[i][j] = dp[i-1][j] || dp[i-1][j-a[i-1]];
            }
        }
        return dp[n][sum];
    }
```

t.c = n X sum
s.c = n X sum

3. Minimum sum partition

```
public int minDiffernce(int a[], int n)
{
               int sum = 0;
               for(int i=0;i<n;i++)
                   sum+=a[i];

              int s = sum;
              sum/=2;
              boolean dp[][] = new boolean[n+1][sum+1];
```

```
            checkSum(a,n ,sum, dp);
            int second=0, first=0;

            for(int i=sum ;i>=0;i--)
                if(dp[n][i]){
                    first = i;
                    break;
                }

            second = s - first;
            return Math.abs(first-second);
}
void checkSum(int a[],int n, int sum, boolean dp[][]){

             for(int i=0;i<=sum ;i++)
                dp[0][i] = false;

             for(int i=0;i<=n;i++)
                dp[i][0] = true;

            for(int i=1;i<=n;i++)
            {
               for(int j=1;j<=sum ;j++){
                  if(j<a[i-1])
                      dp[i][j] = dp[i-1][j];
                  else
                      dp[i][j] = dp[i-1][j] || dp[i-1][j-a[i-1]];
               }
            }
}
```

t.c = n X sum
s.c = n X sum

4) Target Sum

You are given a list of non-negative integers, a1, a2, ..., an, and a target, S. Now you have 2 symbols + and −. For each integer, you should choose one from + and − as its new symbol.
Find out how many ways to assign symbols to make sum of integers equal to target S.

i/p: 0 0 0 0 0 0 0 0 1
        1
o/p : 256

(if 0's are not taken care then o/p is one)
(also (sum+-diff)%2 = 0 for all valid conditions)

```
  public int findTargetSumWays(int[] nums, int diff) {
        int sum = 0, count0=0;
        int n = nums.length;
        for(int i=0;i<n;i++){
            sum+=nums[i];
            if(nums[i]==0)
                count0++;
        }
        if(sum<diff || (sum-diff)%2==1)
            return 0;

        int s1 = (diff+sum)/2;
        int ans = countWays(nums, n, s1);
        return (int)Math.pow(2, count0)*ans;
    }
    public int countWays(int a[],int n, int sum){
        int dp[][] = new int[n+1][sum+1];

        for(int i=0;i<=sum ;i++)
            dp[0][i] = 0;
        for(int i=0;i<=n;i++)
            dp[i][0] = 1;

        for(int i=1;i<=n;i++){
            for(int j=1;j<=sum ;j++){
                if(a[i-1]==0)
                    dp[i][j] = dp[i-1][j];
                else if(j<a[i-1])
                    dp[i][j] = dp[i-1][j];
                else
                    dp[i][j] = dp[i-1][j] + dp[i-1][j-a[i-1]];
            }
        }
        return dp[n][sum];
    }
```

t.c = n X sum
s.c = n X sum

5) Josephus Problem

Recursive

```
int josephus(int n, int k)
{
        if(n==1)
                return 1;

        return (josephus(n-1, k) + k -1) %n+1;
}
```

Iterative

```
List<Integer>

int josephus(List<Integer> a, int start, int k)
{
	if(a.size()==0)
		return a.get(0);

	start = (start + k)%a.size();
	a.remove(start);

	return josephus(a, start, k);
}
```

6. Breadth First Search

```
List<Integer> bfs(ArrayList<ArrayList<Integer>> adj, int v)
{
	List<Integer> answer = new ArrayList<>();
	Queue<Integer> q = new LinkedList<>();

	boolean visited[] = new boolean[v];
	q.add(0);
	visited[0] = true;

	while(!q.isEmpty())
	{
		int cur = q.remove();
		answer.add(cur);

		for(int i=0; i<adj.get(cur).size(); i++)
		{
			if(!visited[adj.get(cur).get(i)])
			{
				q.add(adj.get(cur).get(i));
				visited[adj.get(cur).get(i)] = true;
			}
		}
	}

	return answer;
}
```

T.C = O(V+E)
S.C = O(V)

7) Depth First Search

```
public ArrayList<Integer> dfsOfGraph(int v, ArrayList<ArrayList<Integer>> adj)
   {
      ArrayList<Integer> answer = new ArrayList<>();
```

```
        boolean visited[] = new boolean[v];
        dfs(adj, answer, 0, visited);
        return answer;
    }

    public void dfs(ArrayList<ArrayList<Integer>> adj, ArrayList<Integer> answer ,int src, 
boolean v[]){
        v[src] = true;
        answer.add(src);

        for(int i=0;i<adj.get(src).size();i++){
            int cur = adj.get(src).get(i);
            if(!v[cur]){
                dfs(adj, answer, cur, v);
            }
        }
    }
```

T.C = O(V+E)
S.C = O(V)

8) Balance Parenthesis

```
public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();

        for(int i=0;i<s.length();i++){
            char c = s.charAt(i);
            if(c=='(' || c=='[' || c=='{')
                stack.push(c);
            else if(stack.isEmpty())
                return false;
            else{
                if(c==')' && stack.peek()=='(')
                    stack.pop();
                else if(c==']' && stack.peek()=='[')
                    stack.pop();
                else if(c=='}' && stack.peek()=='{')
                    stack.pop();
                else
                    return false;
            }
        }
        if(!stack.isEmpty())
            return false;
        return true;
    }
```

T.C = s.length()
S.C = s.length()

9) Next Greater Element (Variations)

i. NGS in Linear array

```
public static long[] nextLargerElement(long[] a, int n) {
        Stack<Integer> stack = new Stack<>();
        long ans[] = new long[n];
        ans[n-1] = -1;
        stack.push(n-1);
        for(int i=n-2;i>=0;i--){
            while(!stack.isEmpty() && a[stack.peek()]<=a[i])
                stack.pop();
            if(stack.isEmpty())
                ans[i] = -1;
            else
                ans[i] = a[stack.peek()];
            stack.push(i);
        }
        return ans;
    }
```

T.C = n
S.C = n

ii. NGS in Circular array (Leetcode)

Given a circular array find the next greater element of each index

Eg: i/p: 8 6 0 1 3

o/p: -1 8 1 3 8

Approachs:

a. If it is a Circular array then copy the elements of the into another array twice.
double array: 8 6 0 1 3 8 6 0 1 3
Now find NGS of this using brute force.

double array has copied elements

```
for(int i=0; i<a.length; i++)
{
        res[i] = -1;

        for(int j = i+1; j<double.length; j++)
        {
                if(double[i]<double[j])
                {
                    res[i] = double[j];
                    break;
                }
            }
        }
```

t.c = n^2
s.c = n

b. Similar to above approach but don’t use extra space

```
        for(int i=0;i<a.length; i++)
        {
            res[i] = -1;
            for(int j=1; j<a.length; j++)
            {
                if(a[i] < a[(i+j)%a.length])
                {
                    res[i] = a[(i+j]%a.length];
                    break;
                }
            }
        }
```

t.c = n^2
s.c = 1 (if result is not considered)

c. Do the exact same stack approach but do it twice

```
        for(int i = n-1; i>=0; i- -)
        {
            while(!s.isEmpty() && a[i]>=a[s.peek()])
                s.pop();

            res[i] = s.isEmpty()?-1:a[s.peek()];
            s.push(i);
        }

        for(int i = n-1; i>=0; i- -)
        {
              while(!s.isEmpty() && a[i]>=a[s.peek()])
                s.pop();

            res[i] = s.isEmpty()?-1:a[s.peek()];
            s.push(i);
        }
```

t.c = n + n
s.c = n

d. Doing it in one Pass

```
        for(int i=2*n-1 ; i>=0; i- -)
        {
                while(!s.isEmpty() && a[s.peek()]<=a[i%a.length])
                        s.pop();
                res[i%a.length] = s.isEmpty()?-1:a[s.peek()];
                s.push(i%a.length);
        }
```

t.c = n
s.c = n

10). LRU Cache

```
   static Deque<Integer> dequeue;
   static HashMap<Integer,Integer> map;
   static int capacity;
   LRUCache(int cap)
   {
      dequeue = new LinkedList<>();
      map = new HashMap<>();
      capacity = cap;
   }

   // this function should return value corresponding to key
   static int get(int key)
   {
      if(!map.containsKey(key))
         return -1;

      int value = map.get(key);
      dequeue.remove(key);
      dequeue.addFirst(key);
      return value;
   }

   // storing key, value pair
   static void set(int key, int value)
   {
      if(!map.containsKey(key)){
         if(dequeue.size()==capacity){
            int last = dequeue.removeLast();
            map.remove(last);
         }
      }
      else{
         dequeue.remove(key);
      }
```

```
        deque.addFirst(key);
        map.put(key, value);
    }
```

T.C = O(1)
S.C = O(n)

11) Largest rectangle in histogram

i. Brute Force

Calculate every subarray and then in every iteration find the min and multiply it with the length.

```
{
int answer = 0;

for(int i=0; i<n; i++)
{
        int min = Integer.MAX_VALUE;
        for(int j=i; j<n; j++)
        {
                min = Math.min(a[i], min);
                int len = j - i +1;

                ans = Math.max(ans, len*min);
        }
}

return ans;
}
```

T.C = O(n^2)
S.C = O(1)

ii. Using Stack

```
 public static long getMaxArea(long a[], long n)  {

        Stack<Integer> s = new Stack<>();

          int i = 0;
          long ans = 0;
          while(i<a.length)
          {
                  while(!s.isEmpty() && a[s.peek()]>a[i])
                  {
                          int top = s.pop();
                          long cur = a[top];
```

```
            if(s.isEmpty())
                ans = Math.max(ans, i*cur);
            else
                ans = Math.max(ans, cur*(i-s.peek()-1));
        }
        s.push(i++);
    }

    while(!s.isEmpty())
    {
        long cur = a[s.pop()];
        if(s.isEmpty())
            ans = Math.max(ans, i*cur);
        else
        {
            int len = i - s.peek() - 1;
            ans = Math.max(ans, len*cur);
        }
    }
    return ans;
  }
```

T.C = O(n)
S.C = O(n)

12) Sliding Window maximum

i. Brute Force Approach

```
int[] slidingWindow(int a[], int k, int n)
{
    int answer[] = new int[n-k+1];

    for(int i=0; (i+k)<n; i++)
    {
        int max = Integer.MIN_VALUE;
        for(int j=i; j<i+k; j++)
        {
            max = Math.max(max, a[j]);
        }
        answer[index++] = max;
    }

    return answer;
}
```

T.C = O(n*k)
S.C = O(n)

ii. Using Deque

```
int slidingWindow(int a[],  int k, int n)
{
        Deque<Integer> dq = new LinkedList<>();
        int answer[] = new int[n-k+1];
        int index = 0;
        int i=0;
        while(i<k)
        {
                while(!dq.isEmpty() && a[dq.peekLast()]<=a[i])
                        dq.pollLast();
                dq.offerLast(i);
                i++;
        }

        while(i<n)
        {
                int cur = dq.peekFirst();
                answer[index++] = a[cur];

                while(!dq.isEmpty() && dq.peekFirst()<=(i-k))
                        dq.pollFirst();

                while(!dq.isEmpty() && a[dq.peekLast()]<=a[i])
                        dq.pollLast();
                dq.offerFirst(i);
                i++;
        }

        answer[index] = dq.peekFirst();
        return answer;
}
```

T.C = O(n)
S.C = O(n)

13) Implement Min Stack

i. Using Another stack

It takes space of n

T.C = n
S.C = n

ii. By putting decoded values into the same stack

```
class MinStack {

    long min = Integer.MAX_VALUE;
    Stack<Long> s;
    public MinStack() {
        s = new Stack<>();
    }

    public void push(int val) {
        if(s.isEmpty()){

            s.push((long)val);
            min = val;

        }
        else if(val>=min){
            s.push((long)val);
        }
        else{
            s.push((long)val+val-min);
            min = val;
        }
    }

    public void pop() {
        if(s.peek()>=min)
            s.pop();
        else
            min = min + min -s.pop();
    }

    public int top() {
        if(s.peek()>min)
            return (int)(long)s.peek();
        return (int)min;
    }

    public int getMin() {
        return (int)min;
    }
}
```

T.C = 1
S.C = n

14) Rotten Oranges

i. Use brute force approach

Traverse the array and every time u encounter change all it's neighbouring 1's to -1 and i't value to 0. Now in the second iteration change all neighbours of -1 to -2 and so on do it till all the oranges are rotten. Every time you change keep a count variable to track . Finally return the count.

T.C = n^2
S.C = 1

ii. Using BFS

```
 class Pair{
        int x;
        int y;
        int t;
        Pair(int a, int b, int c){
            x=a;
            y=b;
            t=c;
        }
    }
    public boolean isSafe(int i, int j, int n, int m, int a[][]){
        if(i>=n || i<0 || j>=m || j<0)
            return false;
        if(a[i][j]==0 || a[i][j]==2)
           return false;

            a[i][j] = 2;
            return true;
    }
    public boolean check(int a[][]){

        for(int i=0;i<a.length; i++)
            for(int j=0;j<a[0].length; j++)
                if(a[i][j]==1)
                    return false;

        return true;
    }
    public int orangesRotting(int[][] a)
    {
        Queue<Pair> q = new LinkedList<>();

        for(int i=0;i<a.length; i++)
            for(int j=0;j<a[0].length; j++)
                if(a[i][j]==2)
                    q.add(new Pair(i, j, 0));

        int count = 0;
        while(!q.isEmpty()){
            Pair temp = q.remove();
            int i = temp.x;
            int j = temp.y;
            int time = temp.t;

            if(isSafe(i-1,j, a.length, a[0].length ,a) ){
                q.add(new Pair(i-1 ,j, time+1));
            }
            if(isSafe(i+1,j, a.length, a[0].length, a) ){
                q.add(new Pair(i+1 ,j, time+1));
            }
            if(isSafe(i, j-1, a.length ,a[0].length ,a) ){
                q.add(new Pair(i, j-1, time+1));
            }
            if(isSafe(i, j+1, a.length, a[0].length, a)){
                q.add(new Pair(i, j+1, time+1));
            }

            count = Math.max(count, time);
        }

        if(!check(a))
            return -1;

        return count;
    }
```

T.C = O(n^2)
S.c = O(n)

15) Sort Stack Using recursion

```
static void sortStack(Stack<Integer> stack, int n){
            if(stack.size()<=1)
                return;

            int top = stack.pop();
            sortStack(stack, n-1);
            insert(stack, top);
            return;
}
static void insert(Stack<Integer> stack, int element){
            if(stack.size()==0 || stack.peek()<=element){
                stack.push(element);
                return;
            }

            int curTop = stack.pop();
            insert(stack, element);
            stack.push(curTop);
            return;
}
```

T.C = O(n^2)
S.C = O(1)