# Greedy Algorithms

1. Activity Selection 1st Approach

```java
import java.util.*;

public class ActivitySelection {
    public static void main(String[] args) {                    // Time Complexity - O(n)
        int start[] = {1, 3, 0, 5, 8, 5};                        // In this program, we have sorted activities.
        int end[] = {2, 4, 6, 7, 9, 9};
        // End time basis sorted
        int maxAct = 0;
        ArrayList <Integer> ans = new ArrayList<>();
        // First Activity
        maxAct = 1;
        ans.add(0);
        int lastEnd = end[0];
        for (int i = 0; i < end.length; i++) {
            if (start[i] >= lastEnd) {
                // Activity selection
                maxAct++;
                ans.add(i);
                lastEnd = end[i];
            }
        }
        System.out.println("Max Activites: " + maxAct);
        for (int i = 0; i < ans.size(); i++) {
            System.out.print("A" + ans.get(i) + " ");
        }
        System.out.println();
    }
}
```

2. Activity Selection 2nd Approach

```java
import java.util.*;
public class ActivitySelection2nd {
    public static void main(String[] args) {
        int start[] = {1, 3, 0, 5, 8, 5};           // Time Complexity - O(n logn)
        int end[] = {2, 4, 6, 7, 9, 9};             // We have to write the code for sorting.
        // Sorting
        int activities[][] = new int[start.length][3];
        for (int i = 0; i < start.length; i++) {
            activities[i][0] = i;
            activities[i][1] = start[i];
            activities[i][2] = end[i];
        }
        // Lambda Function - Shortform of any function
        Arrays.sort(activities, Comparator.comparingDouble(o -> o[2]));
        // End time basis sorted
        int maxAct = 0;
        ArrayList <Integer> ans = new ArrayList<>();
        // 1st Activity
        maxAct = 1;
        ans.add(activities[0][0]);
```

```
        int lastEnd = activities[0][2];
        for (int i = 1; i < end.length; i++) {
            if (activities[i][1] >= lastEnd) {
                // Activity Select
                maxAct++;
                ans.add(activities[i][0]);
                lastEnd = activities[i][2];
            }
        }
        System.out.println("Max Activities: " + maxAct);
        for (int i = 0; i < ans.size(); i++) {
            System.out.print("A" + ans.get(i) + " ");
        }
        System.out.println();
    }
}
```

3. Best time to Buy & Sell Stock

```
public class BestTimeBuySellStock {
    public static int maxProfit(int prices[], int n){
        int buy = prices[0], max_profit=0;
        for(int i=1; i<n; i++){                    //Time Complexity - O(n)
            if (buy > prices[i]) {                 //Space Complexity - O(1)
                buy = prices[i];
            }
            else if(prices[i] - buy > max_profit){
                max_profit = prices[i]-buy;
            }
        }
        return max_profit;
    }
    public static void main(String[] args) {
        int prices[] = {7, 1, 5, 3, 6, 4};
        int n = prices.length;
        int max_profit = maxProfit(prices, n);
        System.out.println(max_profit);
    }
}
```

4. Chocolate Problem

```
import java.util.*;
public class ChocolateProblem {
    public static void main(String[] args) {
        int n=4, m=6;
        Integer costVer[] = {2, 1, 3, 1, 4};    // m-1
        Integer costHor[] = {4, 1, 2};      // n-1
        Arrays.sort(costVer, Collections.reverseOrder());
        Arrays.sort(costHor, Collections.reverseOrder());
        int h=0, v=0;
        int hp = 1, vp = 1;
        int cost = 0;
        while (h < costHor.length && v < costVer.length) {
            if (costVer[v] <= costHor[h]) {     // Horizontal Cuts
                cost += (costHor[h] * vp);
```

```
                hp++;
                h++;
            } else {        // Vertical Cuts
                cost += (costVer[v] * hp);
                vp++;
                v++;
            }
        }
        while (h < costHor.length) {
            cost += (costHor[h] * vp);
            hp++;
            h++;
        }
        while (v < costVer.length) {
            cost += (costVer[v] * hp);
            vp++;
            v++;
        }
        System.out.println("Minimum Cost of Cuts: " + cost); // 42
    }
}
```

5. Fractional Knapsacks

```
import java.util.*;
public class FractionalKnapsack {
    public static void main(String[] args) {
        int val[] = {60, 100, 120};
        int weight[] = {10, 20, 30};
        int W = 50;
        double ratio[][] = new double[val.length][2];
        // 0th Column -> idx
        // 1st Column -> ratio
        for (int i = 0; i < val.length; i++) {
            ratio[i][0] = i;
            ratio[i][1] = val[i] / (double)weight[i];
        }
        // Ascending Order
        Arrays.sort(ratio, Comparator.comparingDouble(o -> o[1]));
        int capacity = W;
        int finalVal = 0;
        for (int i = ratio.length-1; i >= 0; i--) {
            int idx = (int)ratio[i][0];
            if (capacity >= weight[idx]) {      // include full item
                finalVal += val[idx];
                capacity -= weight[idx];
            } else {
                // include fractional item
                finalVal += (ratio[i][1] * capacity);
                capacity = 0;
                break;
            }
        }
        System.out.println("Final Value: " + finalVal);
    }
}
```

6. Indian Coins

```java
import java.util.*;

public class IndianCoins {
    public static void main(String[] args) {
        Integer coins[] = {1, 2, 5, 10, 20, 50, 100, 500, 2000};
        Arrays.sort(coins, Comparator.reverseOrder());
        int countOfCoins = 0;
        int amount = 590;
        ArrayList <Integer> ans = new ArrayList<>();
        for (int i = 0; i < coins.length; i++) {
            if (coins[i] <= amount) {
                while (coins[i] <= amount) {
                    countOfCoins++;
                    ans.add(coins[i]);
                    amount -= coins[i];
                }
            }
        }
        System.out.println("Total(min) coins used: " + countOfCoins);
        for (int i = 0; i < ans.size(); i++) {
            System.out.print(ans.get(i) + " ");
        }
        System.out.println();
    }
}
```

7. Job Sequencing Problem

```java
import java.util.*;

public class JobSequencingProblem {
    static class Job {
        int deadline;
        int profit;
        int id;
        public Job(int i, int d, int p) {
            id = i;
            deadline = d;
            profit = p;
        }
    }
    public static void main(String[] args) {
        int jobsInfo[][] = {{4, 20}, {1, 10}, {1, 40}, {1, 30}};
        ArrayList <Job> jobs = new ArrayList<>();
        for (int i = 0; i < jobsInfo.length; i++) {
            jobs.add(new Job(i, jobsInfo[i][0], jobsInfo[i][1]));
        }
        // Sorting of Objects
        Collections.sort(jobs, (obj1, obj2) -> obj2.profit - obj1.profit);
        // Descending order of profit
        ArrayList <Integer> seq = new ArrayList<>();
        int time = 0;
        for (int i = 0; i < jobs.size(); i++) {
            Job curr = jobs.get(i);
            if (curr.deadline > time) {
```

```
                seq.add(curr.id);
                time++;
            }
        }
        System.out.println("Max Jobs: " + seq.size());
        for (int i = 0; i < seq.size(); i++) {
            System.out.print(seq.get(i) + " ");
        }
        System.out.println();
    }
}
```

8. Kth Largest Odd Number in Range

```
public class KthLargeOddNumInRange {
    public static int kthOdd(int range[], int K){
        if (K <= 0) {           //Time Complexity - O(1)
            return 0;           //Space Complexity - O(1)
        }
        int L = range[0];
        int R = range[1];
        if ((R & 1) > 0) {
            int Count = (int) Math.ceil((R-L+1)/2);
            if (K > Count) {
                return 0;
            } else {
                return (R-2*K+2);
            }
        } else {
            int count = (R-L+1)/2;
            if (K > count) {
                return 0;
            } else {
                return (R-2*K+1);
            }
        }
    }
    public static void main(String[] args) {
        int range[] = {-10, 10};
        int k = 2;
        System.out.println(kthOdd(range, k)); // 7 will be second largest odd number in range of -
10 and 10.
    }
}
```

9. Lexicographically Small String K Sum

```
import java.util.*;
public class LexicographicallySmallStringKsum {
    public static char[] lexo_small(int n, int k){
        char arr[] = new char[n];
        Arrays.fill(arr, 'a');          //Time Complexity - O(n)
        for(int i=n-1; i>=0; i--){          //Space Complexity - O(n)
            k -= i;
            if (k >= 0) {
                if (k >= 26) {
```

```
                    arr[i] = 'z';
                    k -= 26;
                } else {
                    arr[i] = (char)(k+97-1);
                    k -= arr[i] - 'a'+1;
                }
            } else {
                break;
            }
            k += i;
        }
        return arr;
    }
    public static void main(String[] args) {
        int n=5;
        int k=42;
        char arr[] = lexo_small(n, k);
        System.out.println(new String(arr));
    }
}
```

10. Maximum Balanced String Partitions

```
public class MaxBalancedStringPartitions {
    public static int BalancedPartition(String str, int n) {
        if (n == 0) {           //Time Complexity - O(n)
            return 0;           //Time Complexity - O(1)
        }
        int r = 0;
        int l = 0;
        int ans = 0;
        for (int i=0; i<n; i++){
            if (str.charAt(i) == 'R') {
                i++;
            }
            else if(str.charAt(i) == 'L') {
                i++;
            }
            if (r == l) {
                ans++;
            }
        }
        return ans;
    }
    public static void main(String[] args) {
        String str = "LRRRRLLRLLRL";
        int n = str.length();
        System.out.print(BalancedPartition(str, n));
    }
}
```

11. Max Length Chain of Pairs

```java
import java.util.*;

public class MaxLengthChainOfPairs {
    public static void main(String[] args) {
        int pairs[][] = {{5, 24}, {39, 60}, {5, 28}, {27, 40}, {50, 90}};
        Arrays.sort(pairs, Comparator.comparingDouble(o -> o[1]));
        int chainLen = 1;
        int chainEnd = pairs[0][1];     // Last selected pair end       // Chain end
        for (int i = 0; i < pairs.length; i++) {
            if (pairs[i][0] > chainEnd) {
                chainLen++;
                chainEnd = pairs[i][1];
            }
        }
        System.out.println("Maximum Length of Chain: " + chainLen + " set of pairs");
    }
}
```

12. Minimum Absolute Difference Pairs

```java
import java.util.*;

public class MinAbsoluteDifferencePairs {
    public static void main(String[] args) {           // Time Complexity - O(nlogn)
        int A[] = {4, 1, 8, 7};
        int B[] = {2, 3, 6, 5};
        Arrays.sort(A);
        Arrays.sort(B);
        int minDiff = 0;
        for (int i = 0; i < A.length; i++) {
            minDiff += Math.abs(A[i] - B[i]);
        }
        System.out.println("Minimum Absolute Difference of Pairs: " + minDiff);
    }
}
```

13. Split Array into K Subarray

```java
public class SplitArrayIntoKSubArray {
    public static int ans = 10000000;
    public static void solve(int arr[], int n, int k, int index, int sum, int maxSum){
        if (k == 1) {
            maxSum = Math.max(maxSum, sum);             //Time Complexity - O((N-1)c(K-1))
            sum=0;                                      //Space Complexity - O(n)
            for(int i = index; i < n; i++){
                sum += arr[i];
            }
            maxSum = Math.max(maxSum, sum);
            ans = Math.min(ans, maxSum);
            return;
        }
        sum = 0;
        for(int i=index; i<n; i++){
            sum += arr[i];
```

```
            maxSum = Math.max(maxSum, sum);
            solve(arr, n, k-1, i+1, sum, maxSum);
        }
    }
    public static void main(String[] args) {
        int arr[] = {1, 2, 3, 4};
        int k = 3;  // k divisions
        int n = 4;  // size of an array
        solve(arr, n, k, 0, 0, 0);
        System.out.println(ans + "\n");
    }
}
```