```
public class Greedy {

    // Assign Cookies
    class Solution {
        public int findContentChildren(int[] g, int[] s) {
            Arrays.sort(g);
            Arrays.sort(s);
            int i = 0, j = 0, child = 0;
            while (i < g.length && j < s.length) {
                if (g[i] <= s[j]) {
                    child++;
                    i++;
                    j++;
                } else {
                    j++;
                }
            }
            return child;
        }
    }

    // Fractional Knapsack Problem
    class Solution {
        class itemComparator implements Comparator<Item> {
            @Override
            public int compare(Item a, Item b) {
                double r1 = (double) a.value / (double) a.weight;
                double r2 = (double) b.value / (double) b.weight;
                if (r1 < r2) {
                    return 1;
                } else
                    return -1;
            }
        }

        double fractionalKnapsack(int W, Item arr[], int n) {
            // Your code here
            Arrays.sort(arr, new itemComparator());
            int curr = 0;
            double profit = 0.0;
            for (int i = 0; i < n; i++) {
                if (curr + arr[i].weight <= W) {
                    curr += arr[i].weight;
                    profit += arr[i].value;
                } else {
                    int remain = W - curr;
                    profit += ((double) arr[i].value / (double) arr[i].weight) * (double) remain;
                    break;
                }
```

```
        }
        return profit;
    }
}

// Greedy algorithm to find minimum number of coins
class Solution {
    static List<Integer> minPartition(int N) {
        // code here
        int arr[] = new int[] { 2000, 500, 200, 100, 50, 20, 10, 5, 2, 1 };
        int n = arr.length;
        int target = N;
        int i = 0;
        List<Integer> ls = new ArrayList<>();
        while (i < n) {
            if (target >= arr[i]) {
                target -= arr[i];
                ls.add(arr[i]);
            } else {
                i++;
            }
        }
        return ls;

    }
}

// Lemonade Change
class Solution {
    public boolean lemonadeChange(int[] bills) {
        int cnt[] = new int[] { 0, 0, 0 };
        int n = bills.length;
        for (int i = 0; i < n; i++) {
            if (bills[i] == 5) {
                cnt[0]++;
            } else if (bills[i] == 10) {
                if (cnt[0] > 0) {
                    cnt[0]--;
                    cnt[1]++;
                } else {
                    return false;
                }
            } else {
                if (cnt[0] > 0 && cnt[1] > 0 || cnt[0] > 2) {
                    if (cnt[0] > 0 && cnt[1] > 0) {
                        cnt[0]--;
                        cnt[1]--;
                        cnt[2]++;
                    } else {
```

```
                    cnt[0] = cnt[0] - 3;
                }
            } else {
                return false;
            }
        }
    }
    return true;
  }
}
// Valid Paranthesis Checker
// N meetings in one room

class Solution {
    // Function to find the maximum number of meetings that can
    // be performed in a meeting room.
    public static int maxMeetings(int start[], int end[], int n) {
        // add your code here
        ArrayList<Meeting> meet = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            meet.add(new Meeting(start[i], end[i], i));
        }
        ArrayList<Integer> ans = new ArrayList<>();
        MeetingComparator mc = new MeetingComparator();
        Collections.sort(meet, mc);
        int limit = meet.get(0).end;
        ans.add(meet.get(0).pos);
        int count = 1;

        for (int i = 1; i < n; i++) {
            if (meet.get(i).start > limit) {
                count++;
                ans.add(meet.get(i).pos);
                limit = meet.get(i).end;
            }
        }
        return count;
    }

    static class Meeting {
        int start, end, pos;

        Meeting(int s, int e, int p) {
            this.start = s;
            this.end = e;
            this.pos = p;
        }
    }
```

```
    static class MeetingComparator implements Comparator<Meeting> {
        @Override
        public int compare(Meeting m1, Meeting m2) {
            if (m1.end < m2.end) {
                return -1;
            } else if (m1.end > m2.end) {
                return 1;
            } else if (m1.pos < m2.pos) {
                return -1;
            }
            return 1;
        }
    }
}

// Jump Game
class Solution {
    public boolean canJump(int[] arr) {
        int n = arr.length;
        boolean vis[] = new boolean[n];

        Queue<Integer> q = new LinkedList<>();
        int start = 0;
        q.add(start);
        vis[start] = true;

        while (!q.isEmpty()) {
            Integer ind = q.remove();
            if (ind == n - 1) {
                return true;
            }
            int s = arr[ind];
            for (int steps = 1; steps <= s; steps++) {
                if (ind + steps >= n - 1) {
                    return true;
                }

                if (vis[ind + steps] == false) {
                    q.add(ind + steps);
                    vis[ind + steps] = true;
                }
            }
        }
        return false;
    }
}

// Jump Game 2
class Solution {
```

```
    public int jump(int[] arr) {
        if (arr.length == 1) {
            return 0;
        }
        int n = arr.length;
        boolean vis[] = new boolean[n];
        Queue<Integer> q = new LinkedList<>();
        q.add(0);
        vis[0] = true;
        int st = 1;
        while (!q.isEmpty()) {
            int s = q.size();
            for (int j = 0; j < s; j++) {
                Integer ind = q.remove();
                if (ind == n - 1) {
                    return st;
                }
                int steps = arr[ind];
                for (int ir = 0; ir <= steps; ir++) {
                    if (ind + ir >= n - 1) {
                        return st;
                    }
                    int nl = ind + ir;
                    if (vis[nl] == false) {
                        vis[nl] = true;
                        q.add(nl);
                    }
                }
            }
            st++;
        }
        return -1;
    }
}

// Minimum number of platforms required for trains
class Solution {
    // Function to find the minimum number of platforms required at the
    // railway station such that no train waits.
    static int findPlatform(int arr[], int dep[], int n) {
        // add your code here
        Arrays.sort(arr);
        Arrays.sort(dep);
        int res = 1, plat = 1, i = 1, j = 0;
        while (i < n && j < n) {
            if (arr[i] > dep[j]) {
                plat--;
                j++;
            } else {
```

```
                plat++;
                i++;
            }
            if (res < plat) {
                res = plat;
            }
        }
        return res;
    }
}

// Job sequencing Problem
// Candy
// Program for Shortest Job First (or …
// Program for Least Recently Used (LR…
// Insert Interval
// Merge Intervals
class Solution {
    public int[][] merge(int[][] intervals) {
        List<int[]> res = new ArrayList<>();
        if (intervals.length == 0 || intervals == null) {
            return res.toArray(new int[0][]);
        }
        Arrays.sort(intervals, (a, b) -> a[0] - b[0]);
        int start = intervals[0][0];
        int end = intervals[0][1];

        for (int[] i : intervals) {
            if (i[0] <= end) {
                end = Math.max(end, i[1]);
            } else {
                res.add(new int[] { start, end });
                start = i[0];
                end = i[1];
            }

        }
        res.add(new int[] { start, end });
        return res.toArray(new int[0][]);
    }
}
// Non-overlapping Intervals
}
```