

## Greedy Solutions

Solution 1:

Time Complexity :  $O(n)$

Space Complexity:  $O(1)$

```
import java.util.*;

class Solution {

    static int BalancedPartition(String str, int n){

        if (n == 0)
            return 0;

        int r = 0, l = 0;
        int ans = 0;
        for(int i = 0; i < n; i++) {
            if (str.charAt(i) == 'R'){
                r++;
            }
            else if (str.charAt(i) == 'L'){
                l++;
            }

            if (r == l){
                ans++;
            }
        }

        return ans;
    }

    public static void main(String[] args){
        String str = "LLRRRLRRRL";
        int n = str.length();

        System.out.print(BalancedPartition(str, n) + "\n");
    }
}
```

## Solution 2 :

Time Complexity :  $O(1)$

Space Complexity:  $O(1)$

```
class Solution {

    public static int kthOdd(int[] range, int K) {

        if (K <= 0)
            return 0;

        int L = range[0];
        int R = range[1];

        if ((R - L + 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[] p = { -10, 10 };
        int k = 8;
        System.out.println(kthOdd(p, k));
    }
}
```

## Solution 3 :

Time Complexity :  $O(n)$

Space Complexity:  $O(n)$

import java.util.Arrays;

public class Main {

public static char[] lexo\_small(int n, int k){

char arr[] = new char[n];

Arrays.fill(arr, 'a');

for (int i = n - 1; i >= 0; i--) {

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, k = 42;

char arr[] = lexo\_small(n, k);

System.out.print(new String(arr));

}

}

#### Solution 4 :

Time Complexity :  $O(n)$

Space Complexity:  $O(1)$

```
class Solution {
    static int maxProfit(int prices[], int n)
    {
        int buy = prices[0], max_profit = 0;
        for (int i = 1; i < n; i++) {
            if (buy > prices[i])
                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, 6, 4 };
        int n = prices.length;
        int max_profit = maxProfit(prices, n);
        System.out.println(max_profit);
    }
}
```

#### Solution 5 :

Time Complexity :  $O((N-1)C(K-1))$

(Here 'c' here depicts combinations i.e.  $\frac{(n-1)!}{((n-k)!*(k-1)!}$ ) Where N is the number of elements of the array and K is the number of divisions that we are having)

Space Complexity:  $O(n)$

```
class Solution {
    public static int ans = 10000000;
    public static void solve(int a[], int n, int k,
        int index, int sum, int maxsum){
```

```

    if (k == 1) {
        maxsum = Math.max(maxsum, sum);
        sum = 0;
        for (int i = index; i < n; i++) {
            sum += a[i];
        }
        maxsum = Math.max(maxsum, sum);
        ans = Math.min(ans, maxsum);
        return;
    }
    sum = 0;
    for (int i = index; i < n; i++) {
        sum += a[i];
        maxsum = Math.max(maxsum, sum);
        solve(a, 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 Array
    solve(arr, n, k, 0, 0, 0);
    System.out.println(ans + "\n");
}
}

```