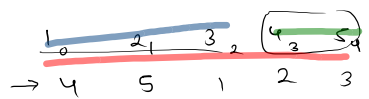
(2) 1. Rotate Right

(2) 2. Sort zero and one.

3. ↳ Sort 0 1 2.

1. Rotate Right.

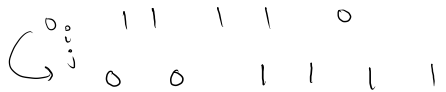
k=2



3 2 1 5 4

4 5 1 2 3

Sort 0 1



ele
arr[i] = 0

swap(i,j)

i++
j++

j-1 → 0

arr[i] = 1

j++

j → i-1 → 1

Sort 0 1 2.

region

↳ 0 ... 0, j-1

↳ 1 j - (i-1)

↳ 2 k+1, n-1

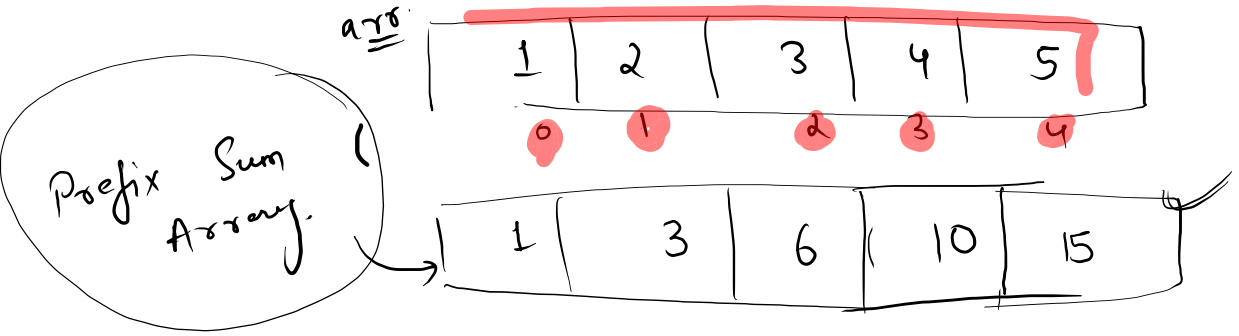
↳ U i - k

0 1 2 3

4

4th = ___) ___ (___) ___ (___ ?

Prefix Array.



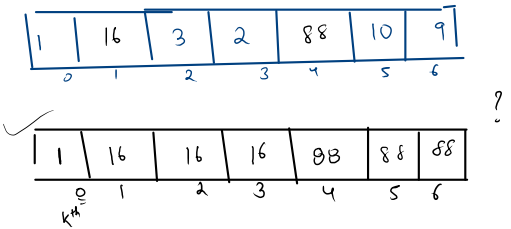
at each idx of PSA store. Sum till this idx.

Greatest Till Me

Make a prefix array of size n such that at the kth index of the prefix array store the greatest element from the left till the kth index of the given array.

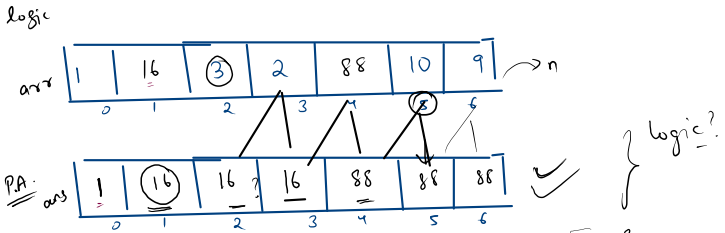
Sample Input 0

```
7
1
88
3
2
16
10
9
```



Sample Output 0

```
1
88
88
88
88
88
88
```



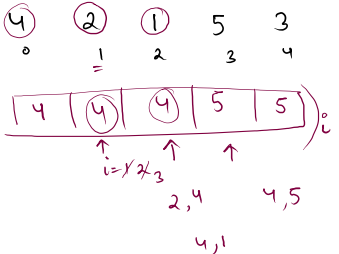
SC ⇝ ? O(n) ✓

TC ⇝ ? O(n)

HW ?
↳ Print Greatest till kth
SC ⇝ O(1)
TC ⇝ O(n)

ans[i-1], arr[i]

ans: 4 4 4 5 5 ?



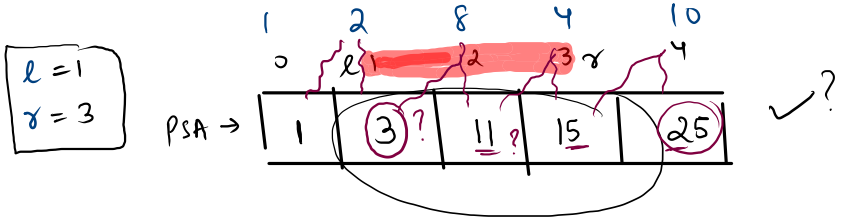

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] arr = new int[n];
        for(int i = 0; i < n; i++){
            arr[i] = scn.nextInt();
        }

        //implementing logic
        int [] ans = new int[n];
        ans[0] = arr[0];
        for(int i = 1; i < n; i++){
            ans[i] = Math.max(ans[i-1] , arr[i]);
        }
        for(int i = 0; i < n; i++){
            System.out.println(ans[i]);
        }

    }
}
```

Print Prefix Sum between L and R

Take an integer input l and r such that l,r<=array.length. Given an array. Make a prefix sum array from this. The print the sum of the elements inside the array starting from the l-index till the r-index(l and r both inclusive)

Sample Input 0

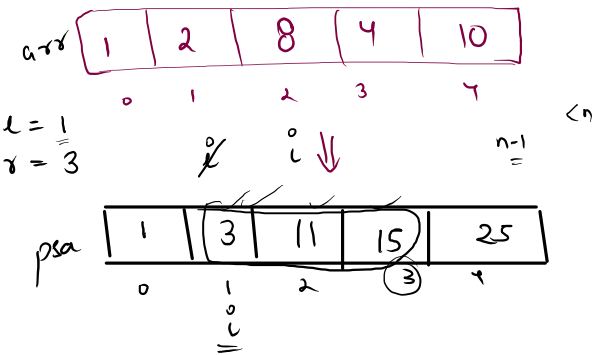
```
5
1
2
8
4
10
1
3
```



Sample Output 0

```
3
11
15
```

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] arr = new int[n];
        for(int i = 0; i < n; i++){
            arr[i] = scn.nextInt();
        }
        int l = scn.nextInt();
        int r = scn.nextInt();

        //logic
        int [] psa = new int[n];
        psa[0] = arr[0];
        for(int i = 1; i < n; i++){
            psa[i] = psa[i-1] + arr[i];
        }
        for(int i = l; i <= r; i++){
            System.out.println(psa[i]);
        }
    }
}
```



Print Freq of Alphabet in String

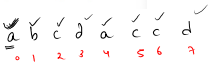
Given a string consisting of only small case alphabets. Find the frequency of each alphabet in O(N) time and you can take extra space. Then print the frequency of each alphabet in the string if that alphabet is contained in the string.

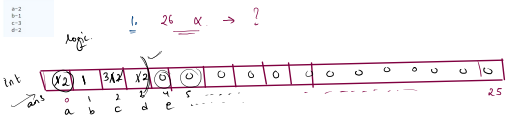
Sample Input 0

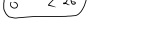
```
abcdaccd
```

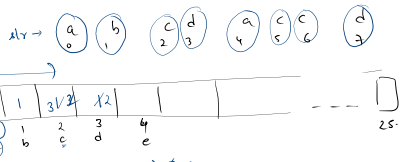
Sample Output 0

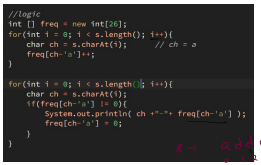
```
a=2
b=1
c=3
d=2
```

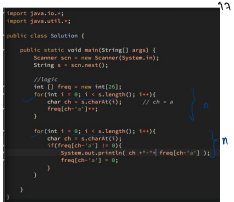












Maximum Freq Character

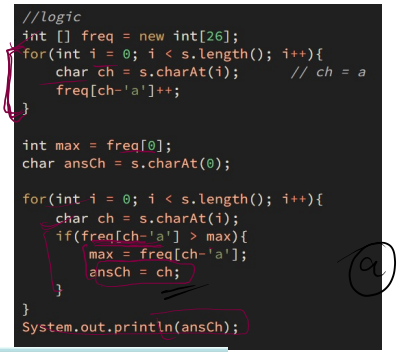
Given a string consisting of only small case alphabets. Find the element with the maximum occurrence. The solution should have O(n) time complexity.

Sample Input 0

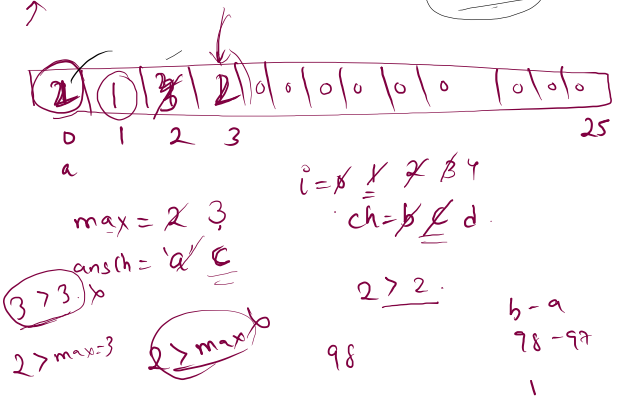
abcdaccd

Sample Output 0

c



```
//logic
int [] freq = new int[26];
for(int i = 0; i < s.length(); i++){
    char ch = s.charAt(i);      // ch = a
    freq[ch-'a']++;
}

int max = freq[0];
char ansCh = s.charAt(0);

for(int i = 0; i < s.length(); i++){
    char ch = s.charAt(i);
    if(freq[ch-'a'] > max){
        max = freq[ch-'a'];
        ansCh = ch;
    }
}
System.out.println(ansCh);
```




```
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();

        //logic
        int [] freq = new int[26];
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);      // ch = a
            freq[ch-'a']++;
        }

        int max = freq[0];
        char ansCh = s.charAt(0);

        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            if(freq[ch-'a'] > max){
                max = freq[ch-'a'];
                ansCh = ch;
            }
        }
        System.out.println(ansCh);

    }
}
```