Editorial: Minimum Cost

Problem: We have to find smallest subarray having sum of its element greater than or equal to a given value.

Brute force approach: Check all the subarrays and update the answer accordingly.

Time Complexity: O(N*N) where N is the size of array.

Space Complexity: O(1)

Code:

```
#include<bits/stdc++.h>
#define int long long
using namespace std;

int32_t main(){
    int n, k, ans, sum;
    cin>>n>>k;
    vector<int> coins(n);

    for(int i=0; i<n; i++){
        cin>>coins[i];
    }

    ans = n;
    for(int i = 0; i < n; i++){
        sum=0;
        for(int j = i; j < n; j++){
         sum += coins[j];
         if(sum >= k){
             ans = min(ans, j - i +1);
         }
        }
    }
    cout<<ans;
    return 0;
}
```

Optimal approach: Since the given array contains only non negative integers, the subarray sum can only increase or remain the same by including more elements. Therefore, you don't have to include more elements once the current subarray already has a sum large enough. This gives the linear time complexity solution by maintaining a minimum window.

Time Complexity: O(N) where N is the size of array.

Space Complexity: O(1)

Code:

```cpp
#include<bits/stdc++.h>
#define int long long
using namespace std;
int helper(int s, vector<int>& nums) {
    int head = 0, ans = INT_MAX, start = 0, end = 0;
    int n = nums.size(), sum = 0;
    while(end < n){
        sum += nums[end++];
        while(sum >= s){
            if(end - start < ans){
                    head = start;
                    ans = end - start;
            }
            sum -= nums[start];
            start++;
        }
    }
    return ans == INT_MAX ? 0 : ans;
}
int32_t main(){
   int n,k;
   cin>>n>>k;
   vector<int> coins(n);
   for(int i=0; i<n; i++)
     cin>>coins[i];
   cout<<helper(k, coins);
   return 0;
}
```