LeetCode 3: Longest Substring Without Repeating Characters

Problem:
Given a string s, find the length of the longest substring without repeating characters.

1) Brute Force Approach (O(n^3)):
- Check all substrings.
- For each substring, check if all characters are unique.
- Track maximum length.

Python:
```
def lengthOfLongestSubstring(s):
    n = len(s)
    max_len = 0
    for i in range(n):
        for j in range(i+1, n+1):
            sub = s[i:j]
            if len(sub) == len(set(sub)):
                max_len = max(max_len, j-i)
    return max_len
```

C++:
```
int lengthOfLongestSubstring(string s) {
    int n = s.length(), max_len = 0;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<=n;j++){
            string sub = s.substr(i,j-i);
            unordered_set<char> set(sub.begin(), sub.end());
            if(set.size() == sub.size()) max_len = max(max_len,j-i);
        }
    }
    return max_len;
}
```

Java:
```
public int lengthOfLongestSubstring(String s){
    int n = s.length(), maxLen = 0;
    for(int i=0;i<n;i++){
        for(int j=i+1;j<=n;j++){
            String sub = s.substring(i,j);
            Set<Character> set = new HashSet<>();
            for(char c: sub.toCharArray()) set.add(c);
```

```
            if(set.size() == sub.length()) maxLen = Math.max(maxLen, sub.length());
        }
    }
    return maxLen;
}
```

2) Sliding Window + HashSet (O(n)):
- Use two pointers (left, right).
- Maintain a HashSet to track current substring characters.
- Expand right; remove from left when duplicate found.
- Track maximum length.

Python:

```
def lengthOfLongestSubstring(s):
    char_set = set()
    left = max_len = 0
    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_len = max(max_len, right-left+1)
    return max_len
```

C++:

```
int lengthOfLongestSubstring(string s){
    unordered_set<char> set;
    int left = 0, max_len = 0;
    for(int right=0; right<s.length(); right++){
        while(set.find(s[right])!=set.end()){
            set.erase(s[left]);
            left++;
        }
        set.insert(s[right]);
        max_len = max(max_len, right-left+1);
    }
    return max_len;
}
```

Java:

```
public int lengthOfLongestSubstring(String s){
    Set<Character> set = new HashSet<>();
    int left = 0, maxLen = 0;
    for(int right=0; right<s.length(); right++){
```

```
        while(set.contains(s.charAt(right))){
            set.remove(s.charAt(left));
            left++;
        }
        set.add(s.charAt(right));
        maxLen = Math.max(maxLen, right-left+1);
    }
    return maxLen;
}
```

Summary:
Brute Force: Time O(n^3), Space O(n)
Sliding Window + HashSet: Time O(n), Space O(min(n, charset))