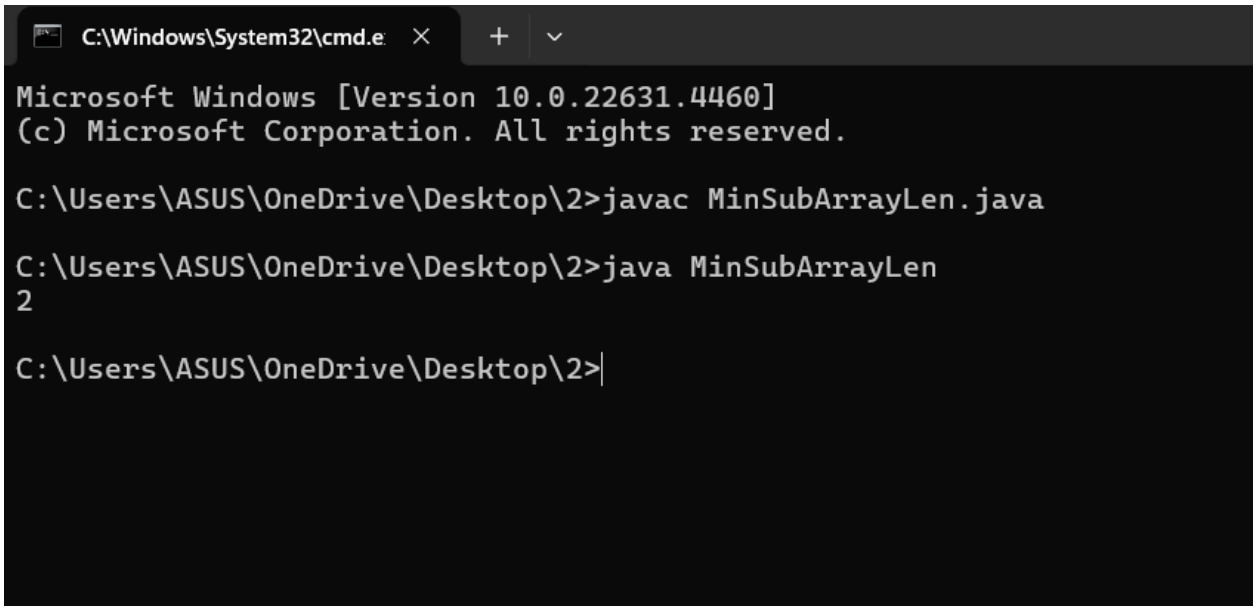
1)Minimum size Subarray Sum

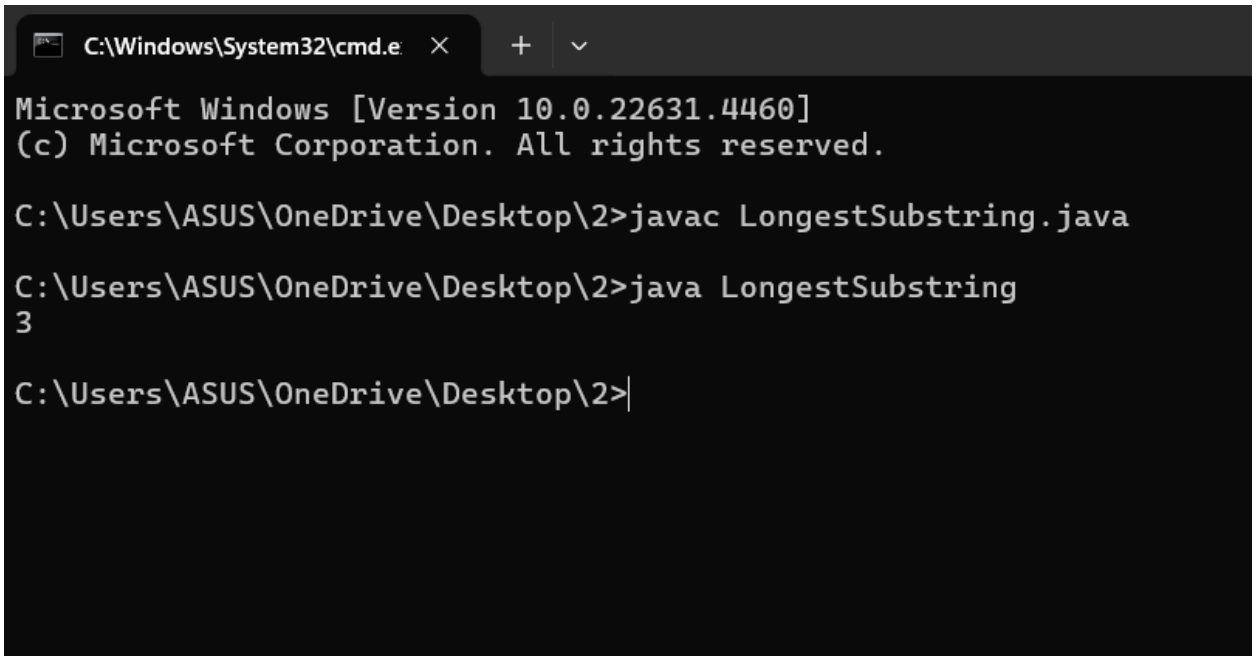
```
public class MinSubArrayLen {
    public static int minSubArrayLen(int target, int[] nums) {
        int left = 0, sum = 0, minLength = Integer.MAX_VALUE;
        for (int right = 0; right < nums.length; right++) {
            sum += nums[right];
            while (sum >= target) {
                minLength = Math.min(minLength, right - left + 1);
                sum -= nums[left++];
            }
        }
        return minLength == Integer.MAX_VALUE ? 0 : minLength;
    }

    public static void main(String[] args) {
        int target = 7;
        int[] nums = {2, 3, 1, 2, 4, 3};
        System.out.println(minSubArrayLen(target, nums));
    }
}
```



Time:O(n)

2)Longest Substring Without repeating characters

```
import java.util.HashSet;

public class LongestSubstring {
    public static int lengthOfLongestSubstring(String s) {
        HashSet<Character> set = new HashSet<>();
        int left = 0, maxLength = 0;
        for (int right = 0; right < s.length(); right++) {
            while (set.contains(s.charAt(right))) {
                set.remove(s.charAt(left++));
            }
            set.add(s.charAt(right));
            maxLength = Math.max(maxLength, right - left + 1);
        }
        return maxLength;
    }

    public static void main(String[] args) {
        String s = "abcabcbb";
        System.out.println(lengthOfLongestSubstring(s));
    }
}
```

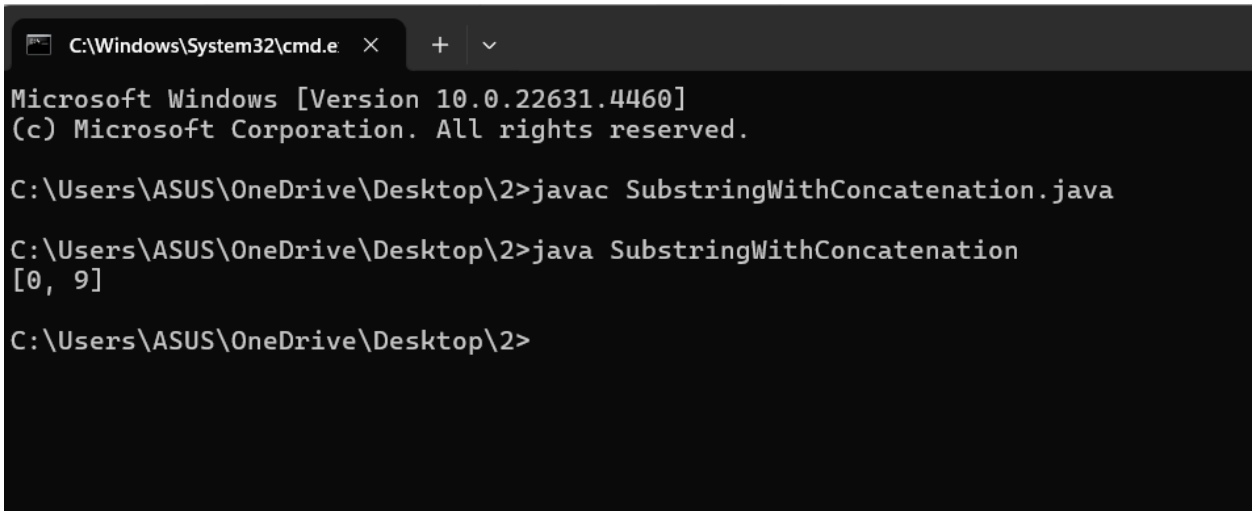


Time:O(n)

3)Substring with concatenation of all words

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class SubstringWithConcatenation {
    public static List<Integer> findSubstring(String s, String[] words) {
        List<Integer> result = new ArrayList<>();
        if (s == null || words == null || words.length == 0) return result;

        int wordLength = words[0].length();
        int substringLength = wordLength * words.length;
        HashMap<String, Integer> wordCount = new HashMap<>();

        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }
```

```
        for (int i = 0; i <= s.length() - substringLength; i++) {
            HashMap<String, Integer> seenWords = new HashMap<>();
            int j = 0;
            while (j < words.length) {
                String word = s.substring(i + j * wordLength, i + (j + 1) *
wordLength);
                if (!wordCount.containsKey(word)) break;
                seenWords.put(word, seenWords.getOrDefault(word, 0) + 1);
                if (seenWords.get(word) > wordCount.get(word)) break;
                j++;
            }
            if (j == words.length) result.add(i);
        }

        return result;
    }

    public static void main(String[] args) {
        String s = "barfoothefoobarman";
        String[] words = {"foo", "bar"};
        System.out.println(findSubstring(s, words));
    }
}
```

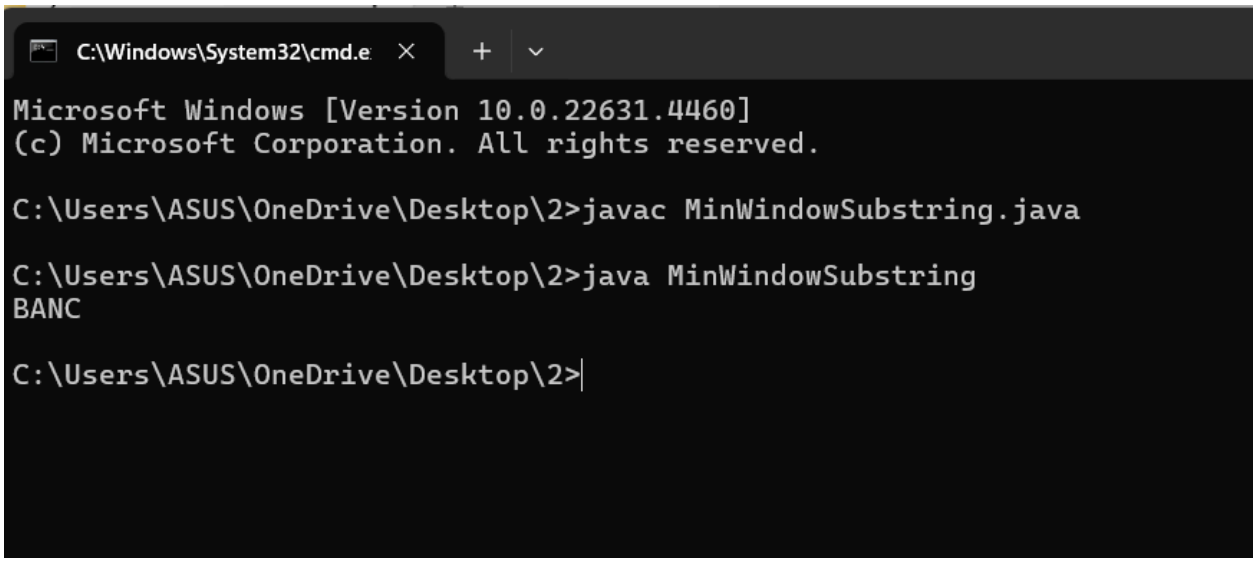


Time:O(n*m*k)

4)Minimum Window substring

```
import java.util.HashMap;

public class MinWindowSubstring {
    public static String minWindow(String s, String t) {
        if (s.length() < t.length()) return "";

        HashMap<Character, Integer> tFreq = new HashMap<>();
        for (char c : t.toCharArray()) {
            tFreq.put(c, tFreq.getOrDefault(c, 0) + 1);
        }

        int left = 0, right = 0, minLen = Integer.MAX_VALUE, start = 0, count =
t.length();
        while (right < s.length()) {
            if (tFreq.containsKey(s.charAt(right))) {
                if (tFreq.get(s.charAt(right)) > 0) count--;
                tFreq.put(s.charAt(right), tFreq.get(s.charAt(right)) - 1);
            }
            right++;

            while (count == 0) {
                if (right - left < minLen) {
                    minLen = right - left;
                    start = left;
                }
                if (tFreq.containsKey(s.charAt(left))) {
                    tFreq.put(s.charAt(left), tFreq.get(s.charAt(left)) + 1);
                    if (tFreq.get(s.charAt(left)) > 0) count++;
                }
                left++;
            }
```

```
        }

        return minLen == Integer.MAX_VALUE ? "" : s.substring(start, start +
minLen);
    }

    public static void main(String[] args) {
        String s = "ADOBECODEBANC";
        String t = "ABC";
        System.out.println(minWindow(s, t));
    }
}
```



Time:O(n)