Question 3

First, we traverse the entire string s starting from index 0, each time mapping the individual characters pointed to by index to a hash table to record the number of occurrences until the current substring contains all k different characters, i.e., there are no 0s in the hash table.

In the second step, the current substring is reduced from the left to the substring that still meets the condition (contains all k different characters). After that, the minimum index and maximum index of the substring are recorded.

The third step is to move the substring to the right in the string s as a whole, that is, the minimum index and maximum index of the substring are +1 at the same time, and in the process update the number of occurrences of the character inside the hash table. Each time we move, we try again to reduce the current substring from the left to the substring that still meets the condition. If a smaller substring is found, the minimum index and maximum index of the record are replaced. The condition for a smaller substring can be obtained by calculating the total number of times the current substring is compared with the recorded substring in the corresponding hash table.

Finally, the length of the shortest substring of s which contains all k different

characters is obtained by subtracting the minimum index from the maximum index.

Given that the maximum index is added by 1 each time and is less than or equal to n, and the minimum index does not exceed the maximum index, the time complexity of the algorithm is $\boldsymbol{O(n)}$.

Supplement Python code

```python
def solution(a,k):
    # Hash table that records the number of
    # occurrences of each k element
    times = [0] * (k + 1)
    # The number of elements that satisfy k
    # should be exactly equal to k
    visited = 0
    # i is the left index, j is the right index
    i = j = 0

    # Start by expanding from 0 to the right until
    # you find the window containing all k
    while j < len(a):
        times[a[j]] += 1
        if times[a[j]] == 1:
            visited += 1
        if visited == k:
            break
        j += 1

    # Then shrink the window from the left to the smallest
    # window that still meets the condition
    while i < j:
        times[a[i]] -= 1
        if times[a[i]] == 0:
            times[a[i]] += 1
            break
        i += 1

    # Finally, move the whole window to the right and
    # replace it if you find a smaller window
    ii = i
    jj = j + 1
    new_times = times.copy()
    while jj < len(a):
        # Delete one on the left
        new_times[a[ii]] -= 1
        if new_times[a[ii]] == 0:
            visited -= 1

        # Add one on the right
        new_times[a[jj]] += 1
```

```
        if new_times[a[jj]] == 1:
            visited += 1

        # Shrink the window again from the left to the smallest
        # window that still meets the condition
        while ii < jj and visited == k:
            ii += 1
            new_times[a[ii]] -= 1
            if new_times[a[ii]] == 0:
                new_times[a[ii]] += 1
                ii -= 1
                break

        # Replace if smaller
        if sum(new_times) < sum(times) and visited == k:
            times = new_times.copy()
            i = ii + 1
            j = jj

        ii += 1
        jj += 1

    print("[", end='')
    for x in range(len(a)):
        if (x >= i and x <= j):
            print(f"\033[31m{a[x]}\033[0m", end='')
        else:
            print(a[x], end='')
        if x != len(a) - 1:
            print(", ", end='')

    print("]")
    print("Length => ", j - i + 1)
    print("")


solution([1,2,3], 3)
solution([1,1,2,3,3], 3)
solution([1,1,1,2,2,2,3,3,2,2,1,3], 3)
solution([1,1,1,2,2,3,2,2,2,1,1,2,2,2,3,2,1], 3)
solution([1,1,1,2,2,3,2,2,1,2,3,2,2,2,2,2,1], 3)
solution([1,1,1,2,2,3,2,2,2,1,1,2,2,2,3,2,2,1], 3)
solution([1,2,3,3,4,1,1,1,1,1,4,3,2,1], 4)
```

```
# Output
# [1, 2, 3]
# Length =>  3

# [1, 1, 2, 3, 3]
# Length =>  3

# [1, 1, 1, 2, 2, 2, 3, 3, 2, 2, 1, 3]
# Length =>  3

# [1, 1, 1, 2, 2, 3, 2, 2, 2, 1, 1, 2, 2, 2, 3, 2, 1]
# Length =>  3

# [1, 1, 1, 2, 2, 3, 2, 2, 1, 2, 3, 2, 2, 2, 2, 2, 1]
# Length =>  3

# [1, 1, 1, 2, 2, 3, 2, 2, 2, 1, 1, 2, 2, 2, 3, 2, 2, 1]
# Length =>  4

# [1, 2, 3, 3, 4, 1, 1, 1, 1, 1, 4, 3, 2, 1]
# Length =>  4
```