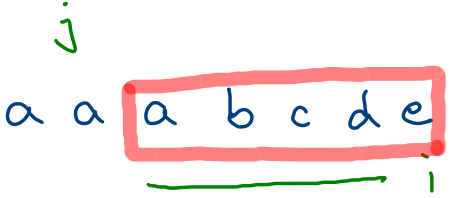
Longest Substring Without Repeating Characters

```
while(i < s.length()-1) {
    //aquire
    while(i < s.length() - 1) {
        //ans updation step
        int len = i - j;

        if(len > oans) {
            oans = len;
        }

        i++;
        char ch = s.charAt(i);

        int nf = map.getOrDefault(ch,0) + 1;
        map.put(ch,nf);

        if(nf >= 2) {
            //invalid
            break;
        }
    }

    //release -> to be valid again
    while(j < i) {

        j++;
        char ch = s.charAt(j);

        if(map.get(ch) == 1) {
            map.remove(ch);
        }
        else {
            int nf = map.get(ch) - 1; //we are valid again, let's go and aquire again
            map.put(ch,nf);
            break;
        }
    }
}

return oans;
```

not correct

(i) aquire till invalid

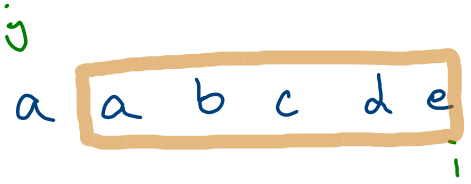
(ii) release to be valid again.



a -> 1
b -> 1
c -> 1
d -> 1
e -> 1

ans = ~~0~~ y ~~2~~ ~~3~~ 4

```
while(i < s.length()-1) {
    //aquire
    while(i < s.length() - 1) {
        i++;
        char ch = s.charAt(i);

        int nf = map.getOrDefault(ch,0) + 1;
        map.put(ch,nf);

        if(nf >= 2) {
            //invalid
            break;
        }
        else {
            //ans updation step
            int len = i - j;

            if(len > oans) {
                oans = len;
            }
        }
    }


    //release -> to be valid again
    while(j < i) {

        j++;
        char ch = s.charAt(j);

        if(map.get(ch) == 1) {
            map.remove(ch);
        }
        else {
            int nf = map.get(ch) - 1; //we are valid again, let's go and aquire again
            map.put(ch,nf);
            break;
        }
    }
}
```



a b c d

a-1 e-1
b-1
c-1
d-1

oans = ~~1 2 3 4 5~~

Count Of Substrings Having All Unique Characters

eg. a b c d

a b c d

a ab abc abcd

b bc bcd

c cd

d

eg. a b b c

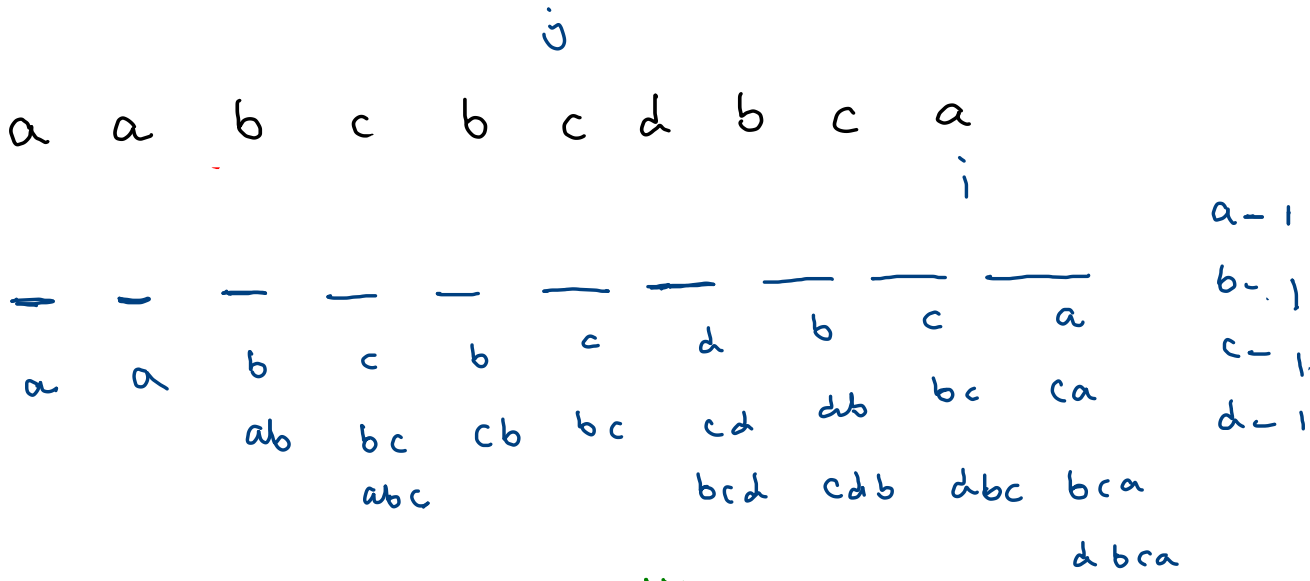
a ✓ ab ✓ abb ✗ abbc ✗

b ✓ bb ✗ bbc ✗

b ✓ bc ✓

c ✓

aabcbcdbca

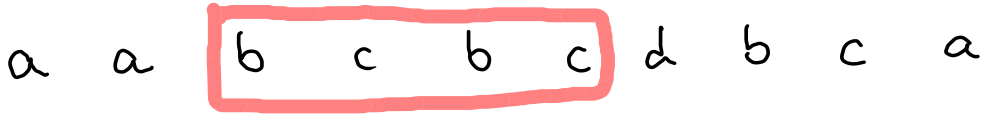


(i) aquiring till invalid [during aquiring always valid]

(ii) release to be valid again

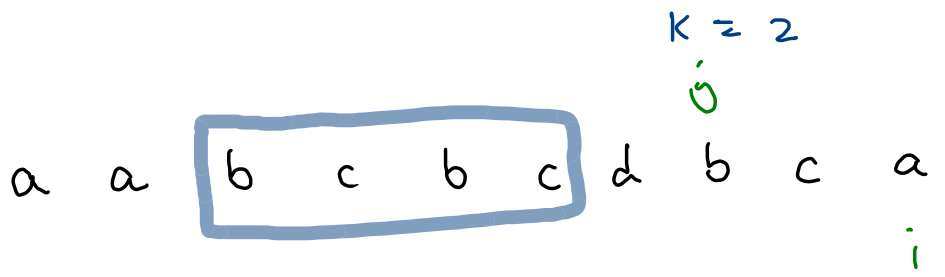
count = 1 + 1 + 2 + 3 + 2 + 2 + 3 + 3 + 4

Longest Substring With Exactly K Unique Characters



aabcbcdbca
2

aquire and release



c = 1

a = 1

0 ans = 4

(i) aquiring to become valid, when invalid ($k+1^{th}$ unique char) break; (to be a longer valid)

(ii) release to become valid.

k = 2

```
//aquire to be valid
while(i < s.length()-1) {
    i++;
    char ch = s.charAt(i);
    int nf = map.getOrDefault(ch,0) + 1;
    map.put(ch,nf);

    if(map.size() == k) {
        int len = i - j;
        ans = Math.max(ans,len);
    }
    else if(map.size() > k) {
        break;
    }
}
```

```
//release to be valid
while(j < i) {
    j++;
    char ch = s.charAt(j);

    if(map.get(ch) == 1) {
        map.remove(ch);
    }
    else {
        int nf = map.get(ch) - 1;
        map.put(ch,nf);
    }


    if(map.size() == k) {
        break;
    }
}
```

j

a a b c b c d b c a a c a b

i

b - 1

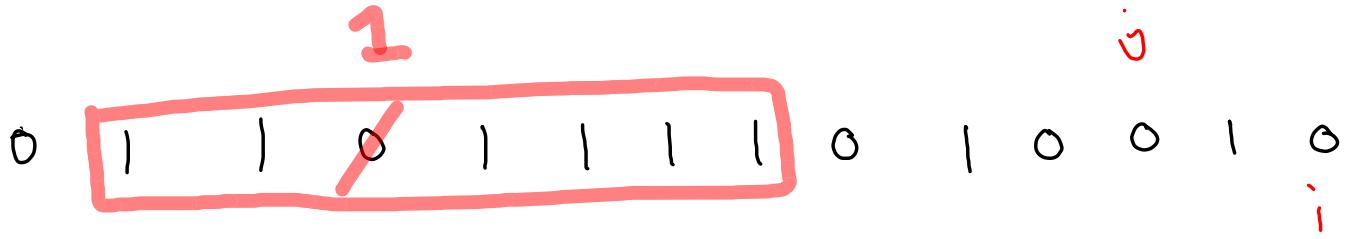
a - 2

ans = ~~1~~ ~~3~~ ~~4~~ 5

Maximum Consecutive Ones - 1

O(n)

flip a 0 → 1



c2 = 1

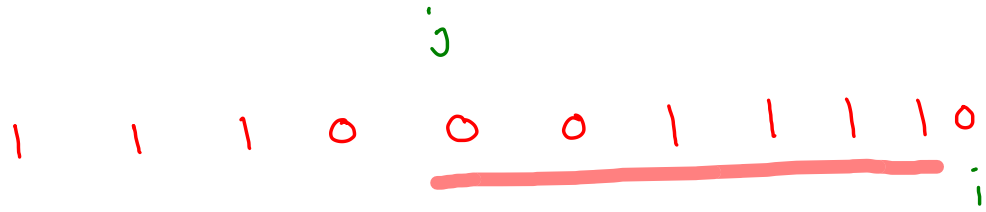
ans = ~~1 2 3 4 5 6~~ 7

flip k '0's → '1's

[1,1,1,0,0,0,1,1,1,1,0]

k = 2

```
while(i <  nums.length-1) {
    //aquire
    while(i < nums.length-1) {
        i++;

        if(nums[i] == 0) {
            cz++;
        }

        if(cz <= k) {
            int len = i - j;
            ans = Math.max(ans,len);
        }
        else {
            //invalid
            break;
        }

    }

    //release
    while(j < i && cz > k) {
        j++;

        if(nums[j] == 0) {
            cz--;
        }
    }
}

return ans;
```

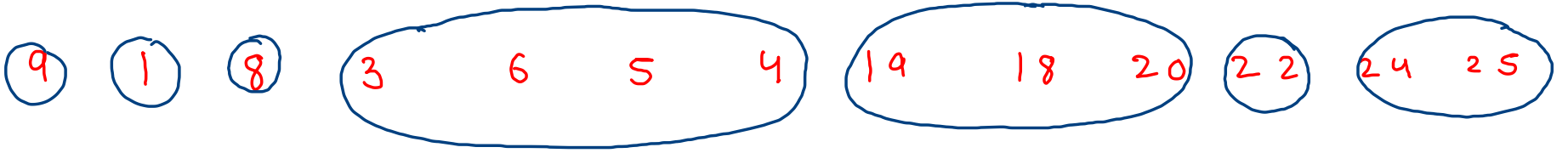


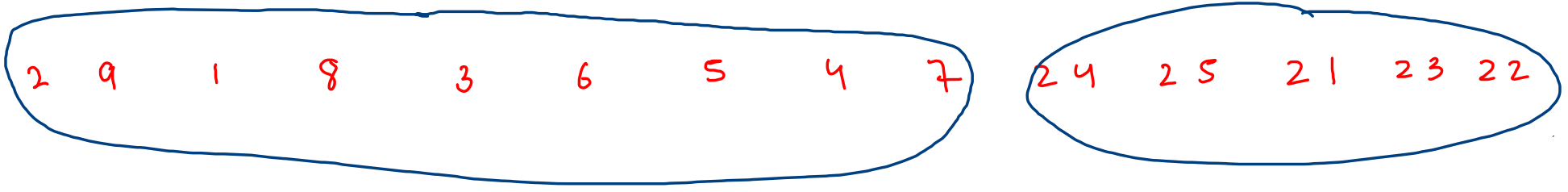
cz = 0 1 2 3 2 3

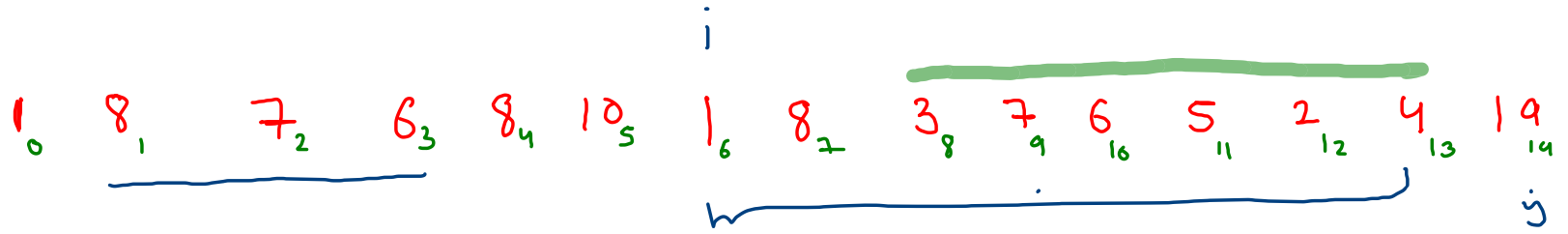
ans = 0 1 2 3 4 5 6

Largest Subarray With Contiguous Elements

$O(n^2)$



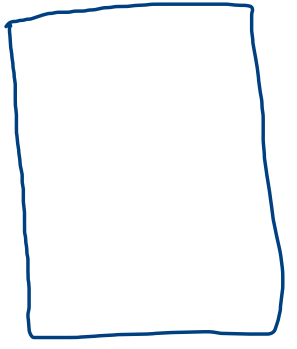





```java
for(int i=0; i < arr.length;i++) {
    int min = Integer.MAX_VALUE;
    int max = Integer.MIN_VALUE;

    HashSet<Integer>hs = new HashSet<>();

    for(int j=i; j < arr.length;j++) {

        if(arr[j] < min) {
            min = arr[j];
        }
        if(arr[j] > max) {
            max = arr[j];
        }

        if(hs.contains(arr[j]) == true) {
            break;
        }
        hs.add(arr[j]);

        int ec = j - i + 1;
        int dc = max - min + 1;

        if(ec == dc) {
            ans = Math.max(ans,ec);
        }
    }
}
return ans;
```

min = 1

max = 19



ans = ~~x~~ ~~x~~ ~~3~~ 8