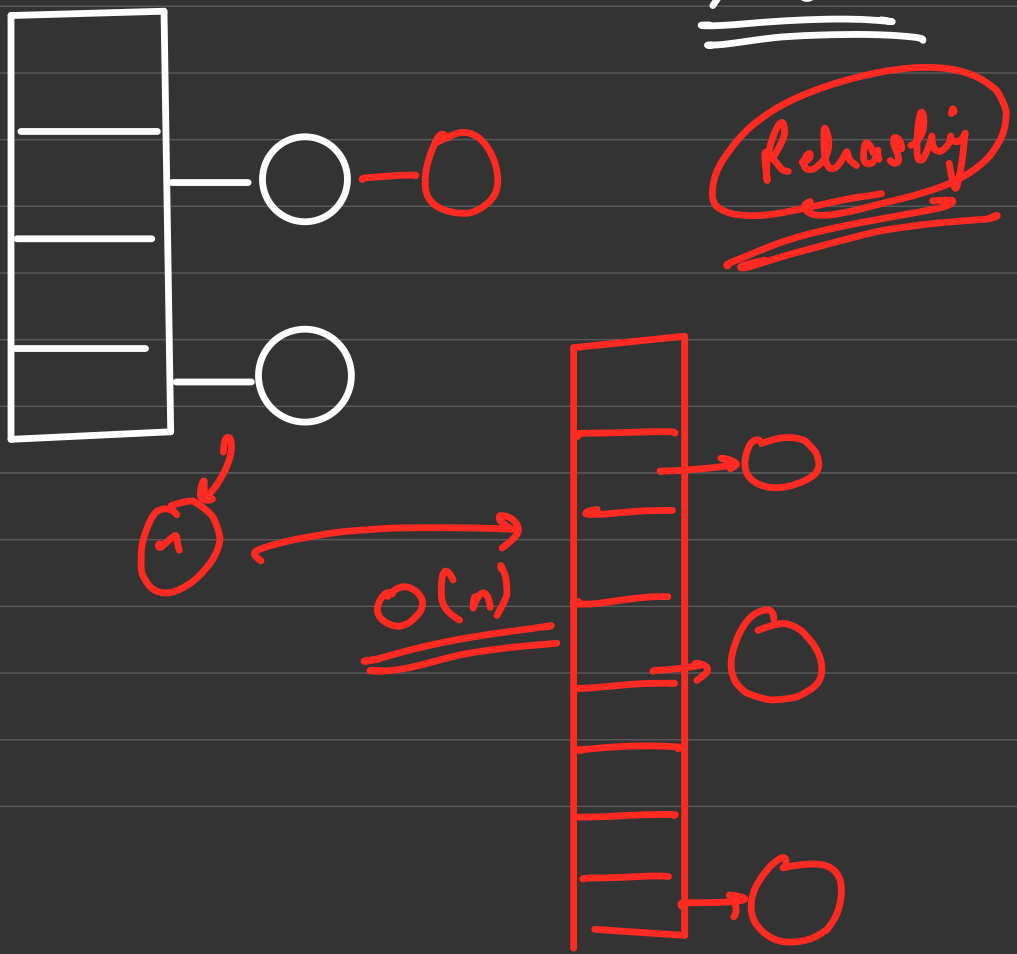

λ=0.5
Rekurencyjny
n
O(n)

Complexity analysis for arrays.

Internally arrays are always of fixed size.

How arrays handles dynamic insertion.

the moment the arrays is full & we need to insert more elements, we create a new array of double the size of prev array & then insert all the prev array elements to new array


Arrays
Capacity
Size
operations
1, 2, 3, 4, 5, 6, 7, 8, 9...
amortized
[ ]
1
1
1
[1, 2]
2
2
2 → 2^0 + 1
[1, 2, 3, 4]
4
3
3 → 2^1 + 1
1
17 → 2^4 + 1
[1, 2, 3, 4, 5, 6, 7, 8]
8
5
5 → 2^2 + 1
1
1
[1, 2, 3, 4, 5, 6, 7, 8, 9]
1
9 → 2^3 + 1

What will the avg complexity ?:

$$\text{avg} = \frac{\text{Total operation}}{\text{Total insertions}}$$

$$= \frac{1+2+3+1+5+1+1+1+9+1+1+1+1+1+1+1+17+\ldots}{n}$$

$$\frac{(1+2+3+1+5+1+1+1+9+1+1\text{-}\text{-}\text{-}\text{-}\text{-}\text{-})}{n}$$

$$\frac{(1+(2^0+1)+(2^1+1)+1+(2^2+1)+1+1+1+(2^3+1)\text{-}\text{-}\text{-}\text{-})}{n}$$

n 1's $\quad$ n $\quad$ $\log n$ terms

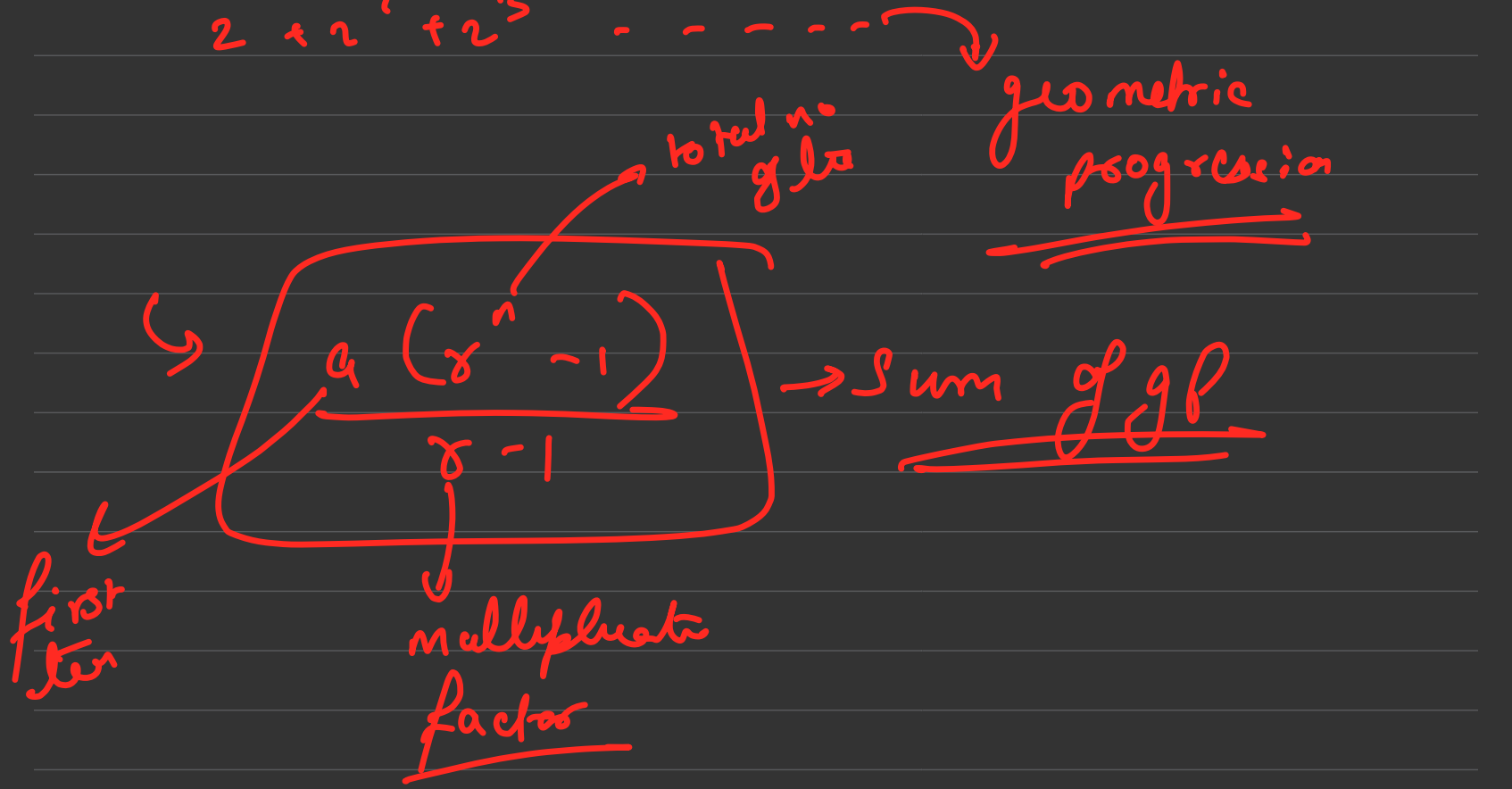
$$(1+1+1+1\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}) + (2^0+2^1+2^2\text{-}\text{-}\text{-}\text{-}\text{-})$$

$$\frac{n + (2^0+2^1+2^2\text{-}\text{-}\text{-}\text{-}\text{-}\text{-}\text{-})}{n}$$ $\log n$ terms

$$\frac{n + (2^0 + 2^1 + 2^2 \cdots\cdots)}{n}$$ → $\log n$ times

$$\frac{\left(n + \frac{1 \times (2^{\log_2 n} - 1)}{2-1}\right)}{n}$$

$$\frac{n + (2^{\log_2 n} - 1)}{n}$$


2 + 2^2 + 2^3 - - - - - -
geometric progression
total no. of terms
$\frac{a(r^n - 1)}{r - 1}$
Sum of GP
first term
multiplicate factor

By logarithmic identity $a^{\log_a n} = n$

↳ $\frac{n + (2^{\log_2 n} - 1)}{n}$

$\Rightarrow \frac{n + (n-1)}{n} \Rightarrow \frac{2n-1}{n} \rightarrow$ constant

avg → $\Theta(1)$

worst → $O(n)$

Best → $\Omega(1)$

$[1]_1 \rightarrow 1$

$[1,2]_2 \rightarrow 2$

$[1,2,3]_3 \rightarrow 3$ $\quad O(n)$

$[1,2,3,4]_4 \rightarrow 4$ $\quad \Theta(n)$

$\vdots$

$\underbrace{(1+2+3+4+5+\cdots\cdots n)}_{n} \rightarrow \Theta(n)$

$\vdots$

[eat, tea, tan, ate, nat, bat]

{

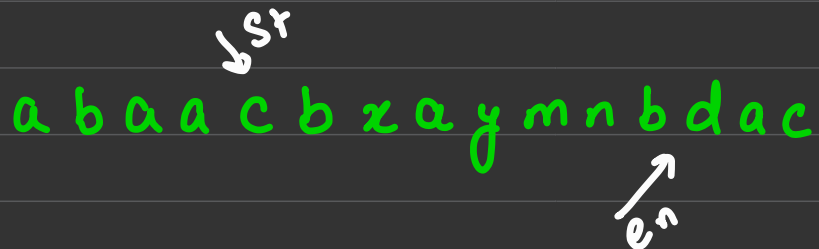
aet : [eat, tea, ate]

ant : [tan, nat]

abt : [bat]

}

We can maintain a window of unique chars



freq map

{
a : 2 1
b : 2 0 2 2
c : 1
x : 1
y : 1
m : 1
n : 1
}

len → en - st + 1

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

$O(N+2N)$

$\approx O(N)$

ans = 7

[-3, -2, 7, 1, 3, 5, 2, 6, 11, 10, 4, -4, -5]

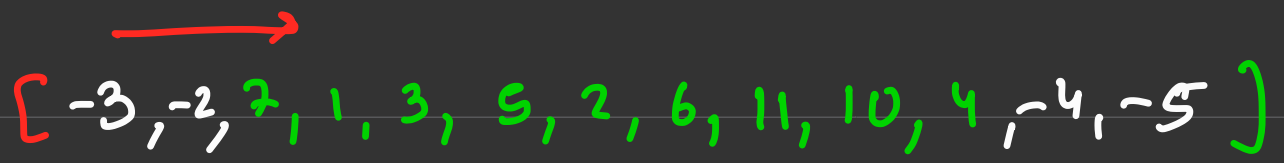
freq map

for uniquely identifying any consecutive sequence

→ if we know start of the sequence and length of the sequence we can generate it.

What if we detect what all elements can be start of some sequence ??

To check if any element x can become start of a longest consecutive sequence or not then we can try to check if x+1 is present in the array or not.




```
var longestConsecutive = function(nums) {
    if(nums.length == 0) return 0;
    let freq = {};
    for(let i = 0; i < nums.length; i++) {
        if(!freq[nums[i]]) freq[nums[i]] = 1;
        else freq[nums[i]]++;
    }
    let ans = 1;
    for(let i = 0; i < nums.length; i++) {
        // check can nums[i] become start of a sequence
        if(freq[nums[i] - 1]) continue;
        // nums[i] is the start point
        // check the max length sequence it can generate
        let len = 0;
        let el = nums[i];
        while(freq[el]) {
            el++;
            len++;
        }
        ans = Math.max(ans, len);
    }
    return ans;
};
```