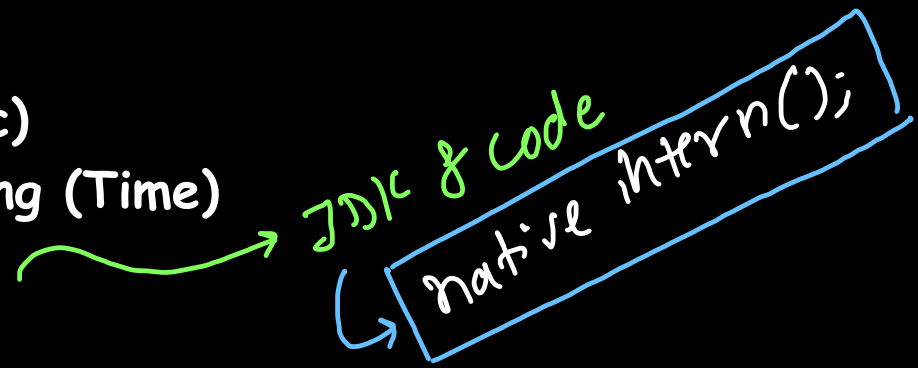
Strings, StringBuilder & StringBuffer

- String Literal vs Object Memory Mapping
- Strings vs Character Array
- Implicit vs Explicit Null Initialization
- Inbuilt Methods in String Class
(charAt, length, Constructors, etc)
- Concatenation Operation in String (Time)
- String Pool or String Interning
- String Immutability
(Reference Variable vs Object Modification)
- String vs StringBuilder
- Operator == vs Method equals()
- StringBuilder vs StringBuffer
- Inbuilt Methods in StringBuilder
(setCharAt, append)
- Wrapper Classes - Immutability



1D array ⇒ character array

↳ 2 bytes (unicode)

Collection of characters ⇝ random index → contiguous allocam

{ 'h', 'e', 'l', 'l', 'o' }

0 1 2 3 4

' ' space

'\n' newline

'\t' tabspace

'\0' null character

Disadvantages ?

① size fixed → size↑ X, size↓ X

② functions or methods (missing) → concatenation (+), compare, equals, hashcode, reverse, append, substring

String Literal vs String object.

class → char[]

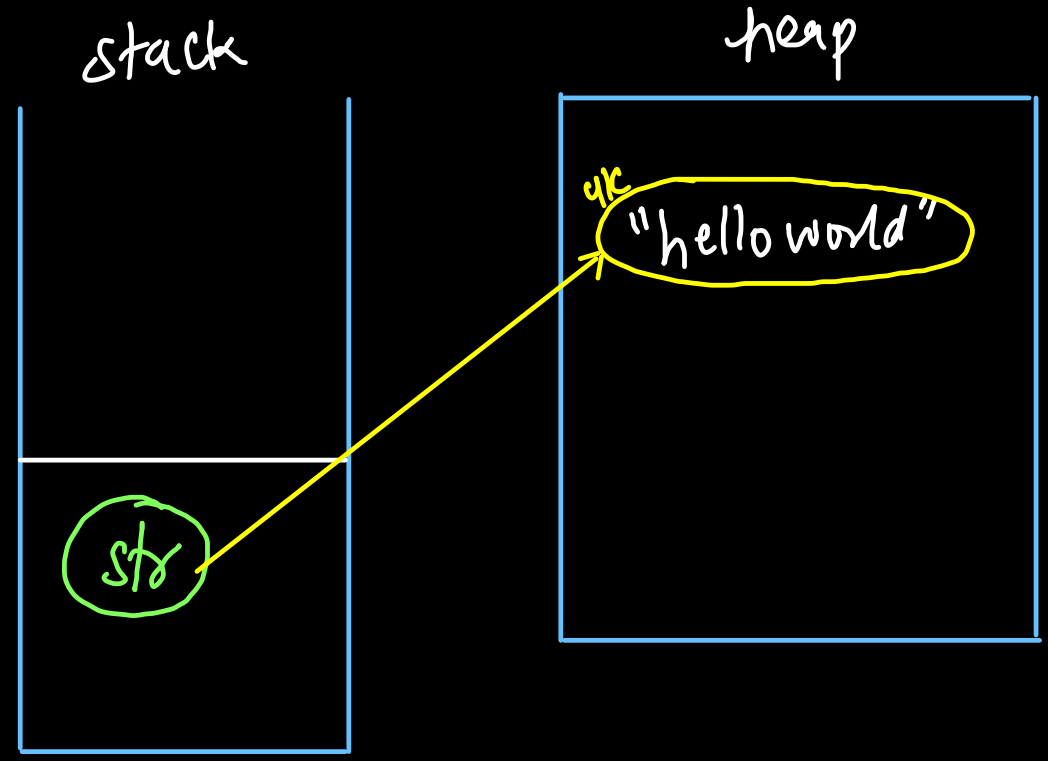
String str = "hello world";

String literal / constant

reference variable (stack)

heap area

Syso(str);
↳ hello world



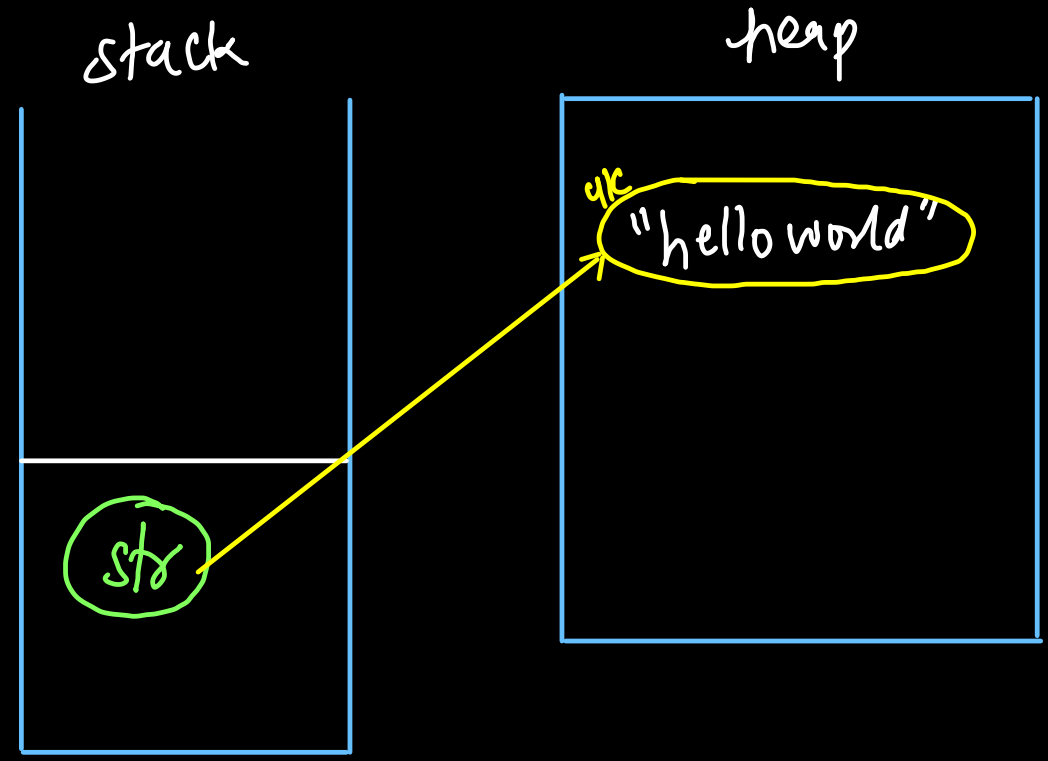
String str = new String("hello world");

reference variable

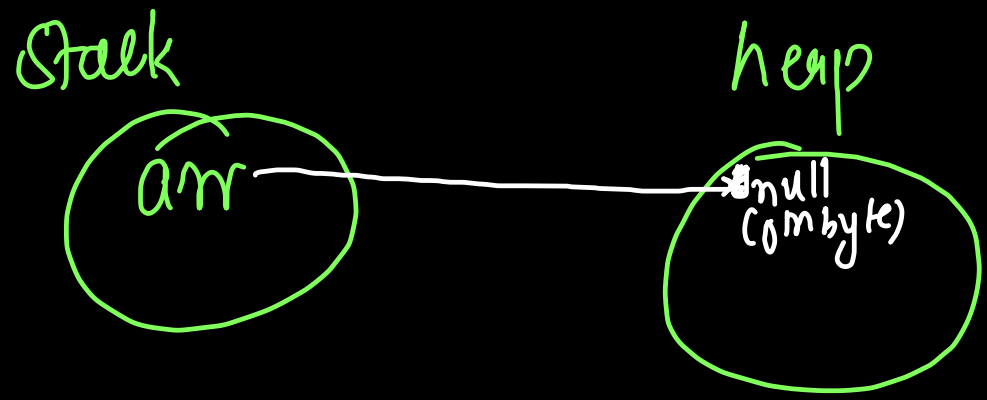
string object

```
Syso(str);
```

↳ hello world



```java
int[] arr; // null (implicitly initialized)
System.out.println(arr); // compilation error

int[] arr = null; // null (explicit initialization)
System.out.println(arr); // null will be printed
System.out.println(arr.length); // Null Pointer Exception
```



```java
String str1;
System.out.println(str1); // compilation error
```

```
Finished in N/A
Line 28: error: variable str1 might not have been in
itialized [in Main.java]
        System.out.println(str1);
                           ^
```

```java
String str2 = null;
System.out.println(str2);
System.out.println(str2.length());
```

```
Finished in N/A
null
java.lang.NullPointerException: Cannot invoke "Strin
g.length()" because "<local1>" is null
  at line 32, Main.main
```


0 1 2 3 4
String str = "hello";
System.out.println(str); // hello -> horizontally
// hello -> vertically
for(int idx = 0; idx < str.length(); idx++){
System.out.println(str.charAt(idx));
}
⇒ hello
⇒
h
e
l
l
o
String funcns
length()
charAt()
length
[]

```java
String str = "";
System.out.println(str);
System.out.println(str.length());

for(char ch = 'a'; ch <= 'z'; ch++){
    str = str + ch; // concatenation
    System.out.println(str);
}
```

O(1)

O(1)

O(1)

n = 26

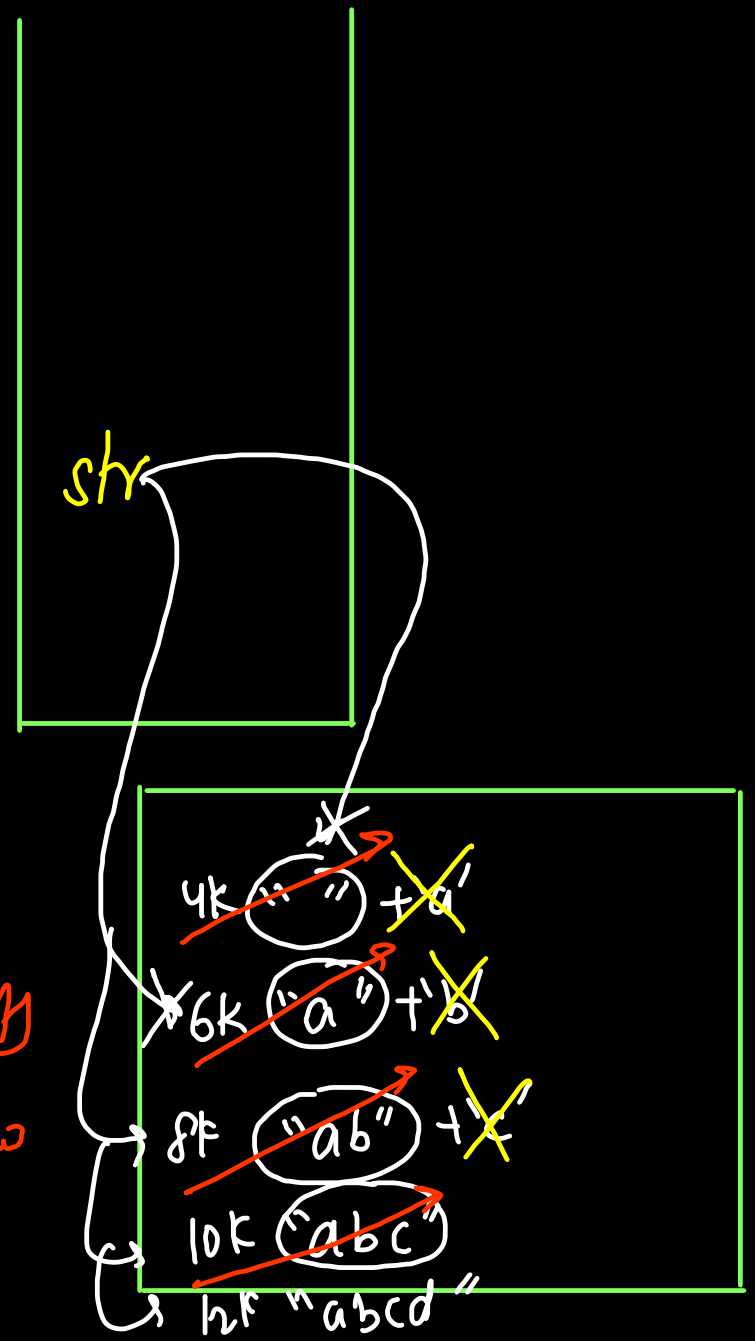
O(n²) ignoring point

O(n)

O(1)

a → ab → abc → abcd → abcde

time → copying both parts & forming new string



String s1dept = "computerscience"

String s2dept = "computerscience"

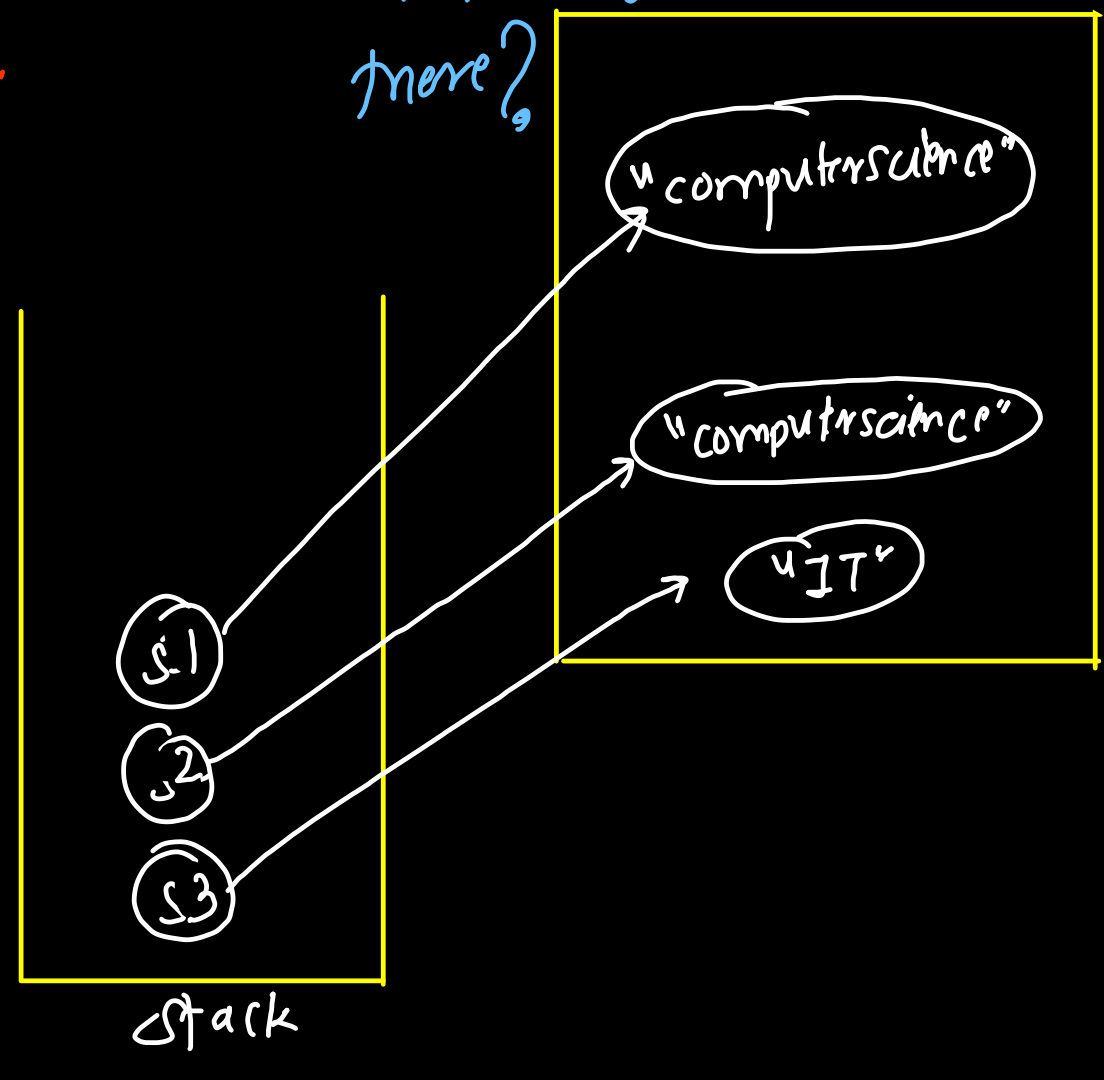
String s3dept = "IT"

need?

memory inefficient

String interning / pool

What if string interning is not there?



String s1 = "CSE";

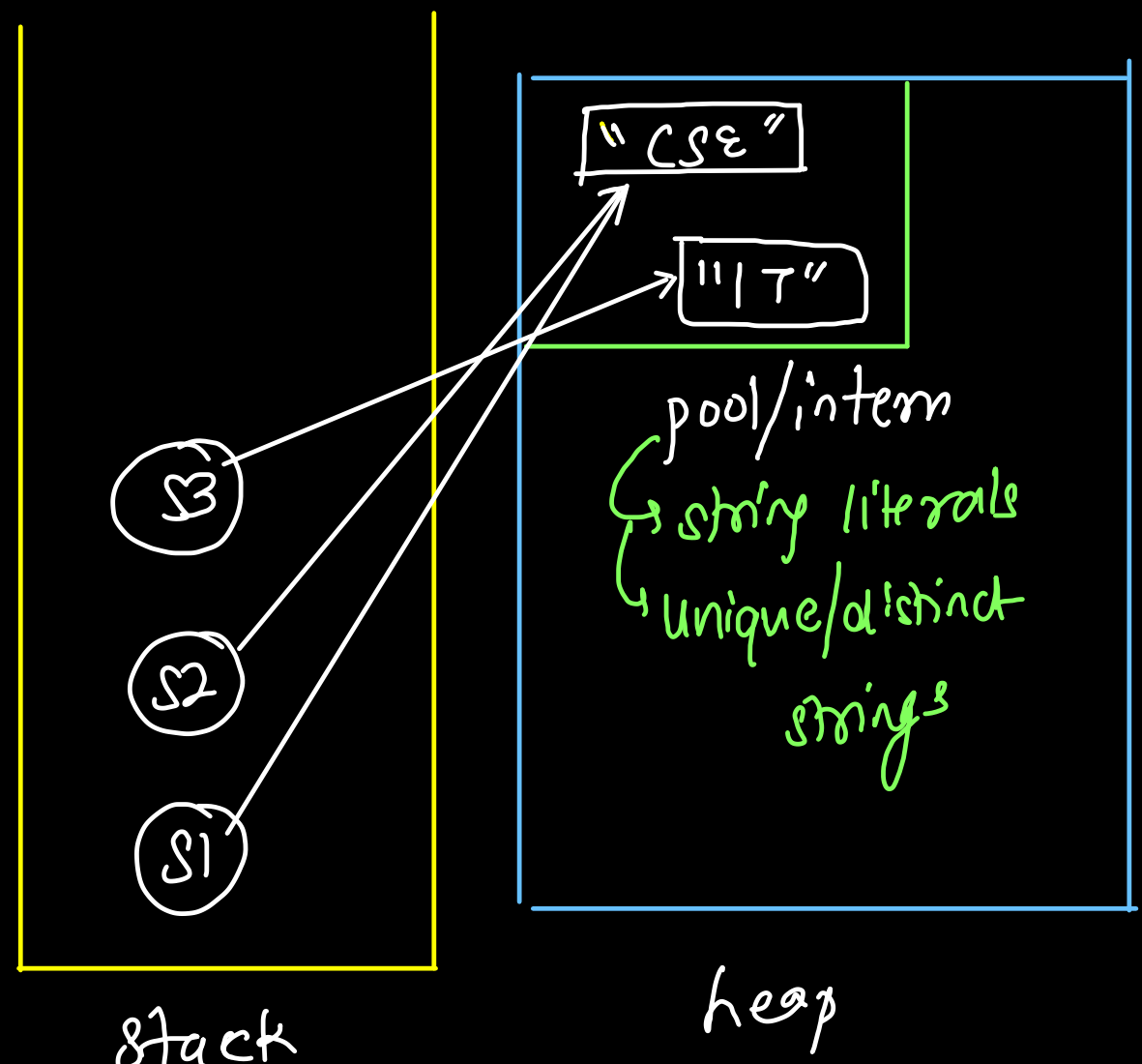
String s2 = "CSE";

String s3 = "IT"

#advantage → reusability (string literals)

⇓

memory efficients



Implication → String Immutability

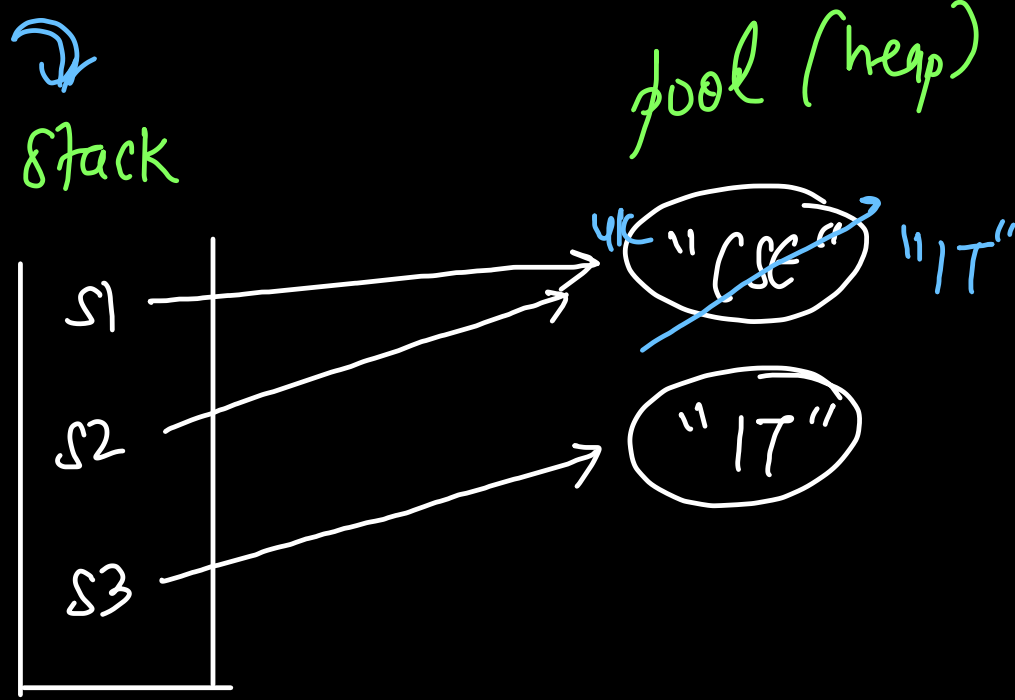
"Strings are **immutable** in Java"

↳ cannot update / change / modify

String immutability is not implemented?

```
String s1 = "CSE";
String s2 = "CSE";
String s3 = "IT";
s1 = "IT";
```

s2 changed but not needed (inconsistent)



String s1 = "CSE";

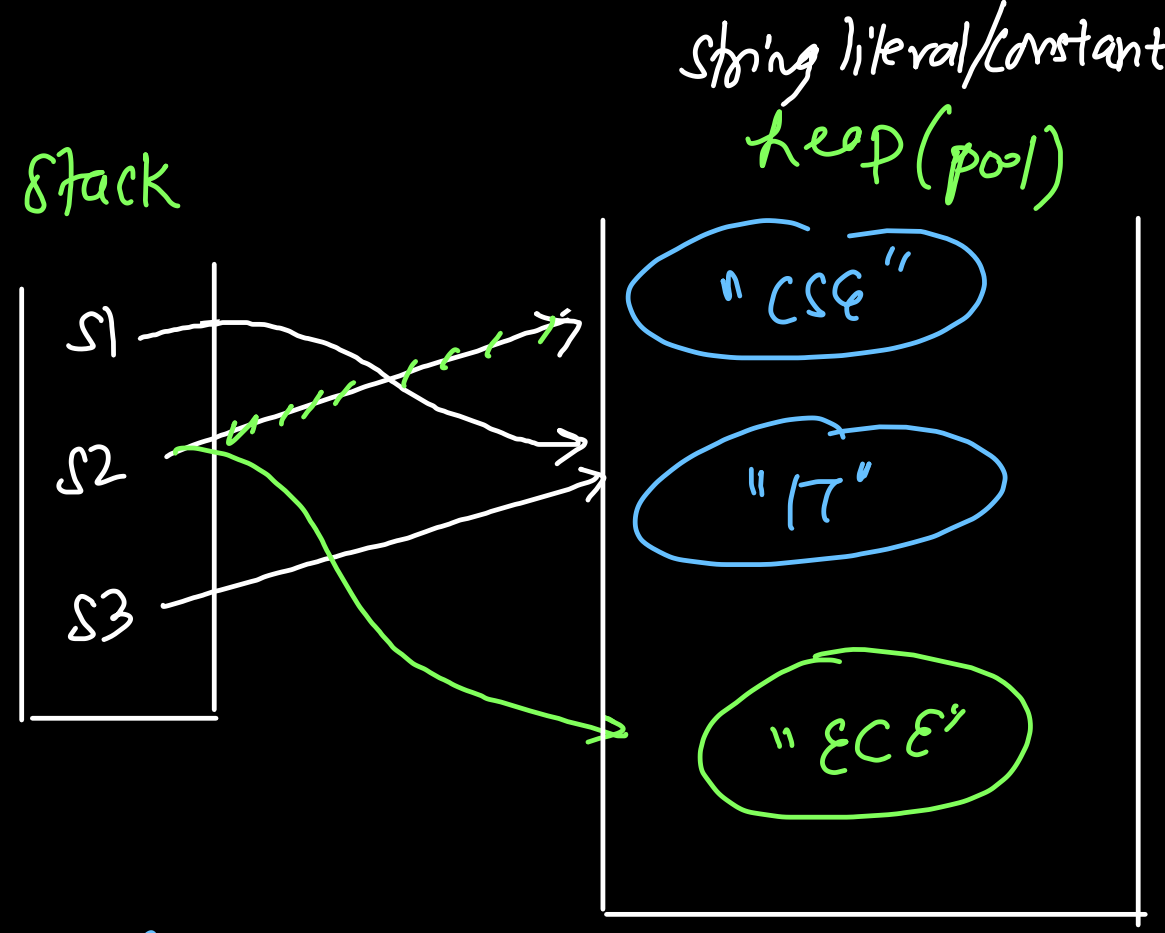
String s2 = "CSE";

String s3 = "IT";

s1 = "IT";

s2 = "ECE";

reference variable can be updated



memory wastage (interning ✗) —stop→ interning pool → inconsistency (immutability ✗)

inconsistency —stop→ immutability

immutability → new strings are created on every modification (time costly)

" Reference variables are mutable but instance/object of heap are immutable "

str = "hello"

variable ✓ reference → str = "world"

str = "hello" → object X

object X: str[0] = 'd' { characters cannot be set }

StringBuilder & StringBuffer

read only characters

⇓

String

read & update characters

⇓

StringBuilder

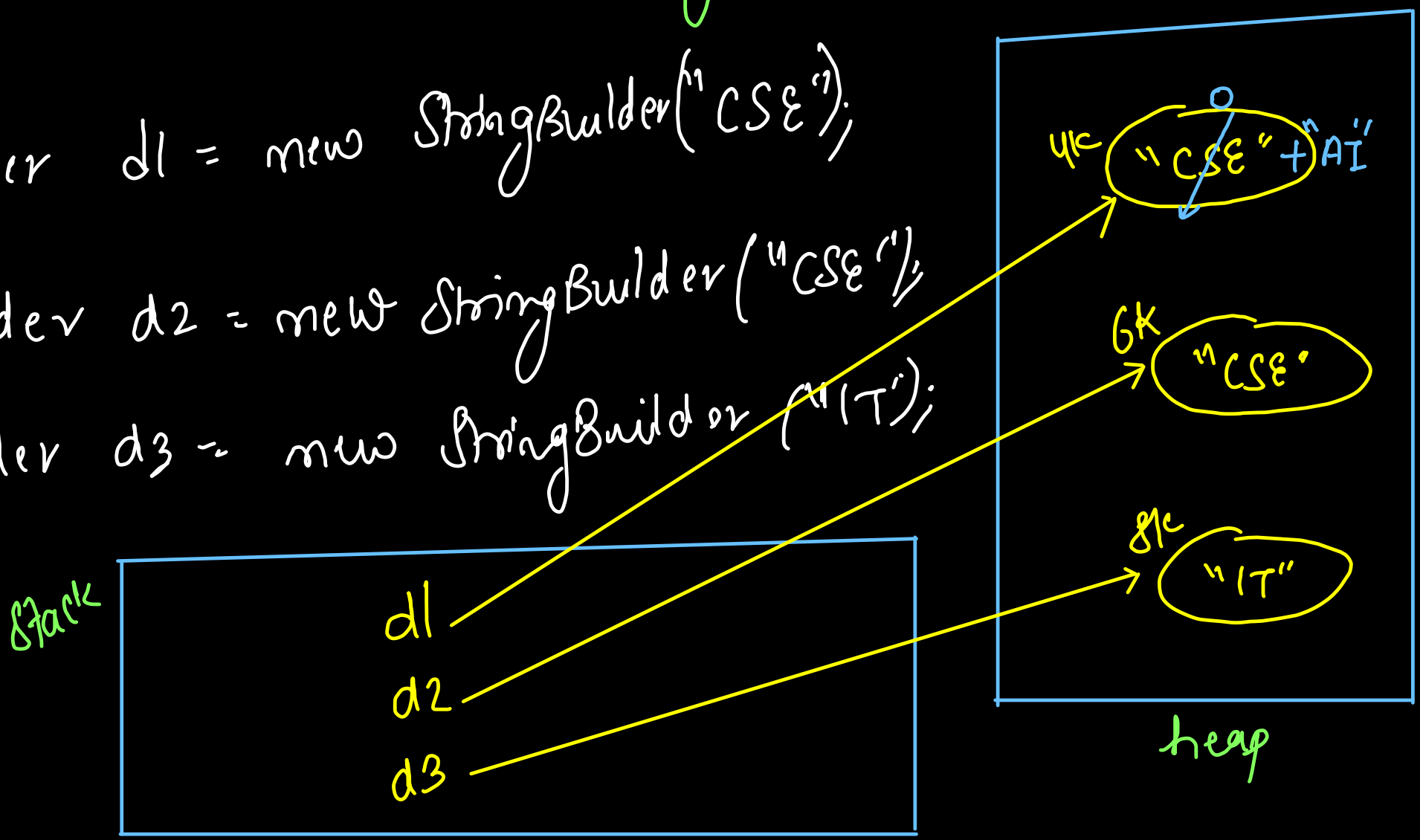
StringBuilder

No interning

No immutability

memory wastage

```
StringBuilder d1 = new StringBuilder("CSE");
StringBuilder d2 = new StringBuilder("CSE");
StringBuilder d3 = new StringBuilder("IT");
```



```java
StringBuilder d1 = new StringBuilder("CSE");
System.out.println(d1.hashCode());

StringBuilder d2 = new StringBuilder("CSE");
System.out.println(d2.hashCode());

StringBuilder d3 = new StringBuilder("IT");
System.out.println(d3.hashCode());

d1.append(" AI");
System.out.println(d1.hashCode());

System.out.println(d1 + " " + d2);
```

```
1828972342
1452126962
931919113
1828972342
CSE AI CSE
```

```java
String s1 = "CSE";
System.out.println(s1.hashCode());

String s2 = "CSE";
System.out.println(s2.hashCode());

String s3 = "IT";
System.out.println(s3.hashCode());

s1 = "COE";
System.out.println(s1.hashCode() + " " + s2.hashCode());
```

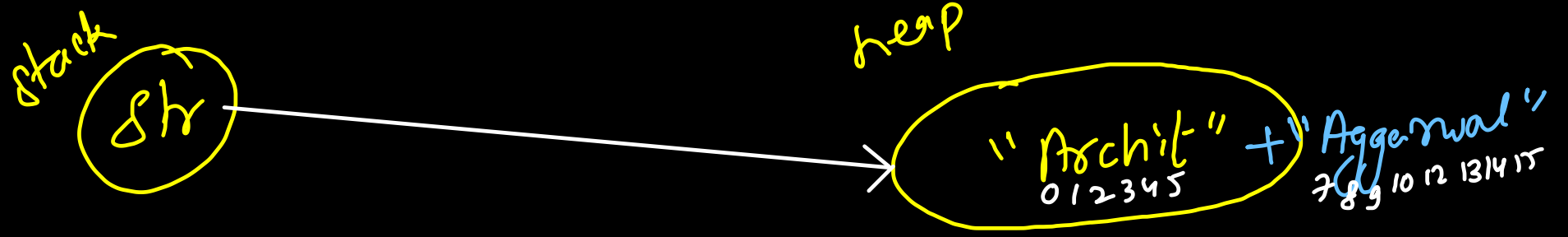
```
67029
67029
2347
66905 67029
```

```
d1.setCharAt(1, 'O'); // update or set: O(1)

d1.deleteCharAt(0); // remove first char: O(n)
d1.deleteCharAt(d1.length() - 1); // remove last char: O(1)
```

```
StringBuilder d1 = new StringBuilder("CSE"); // n1 length
// System.out.println(d1.hashCode());

d1.append(" AI"); // concatenate n2 length

// O(n1 + n2)
// O(n1)
// O(n2)
// O(1)
```

dynamic size (mutable)

StringBuilder str = new StringBuilder("Archit"); n1 = 6



str.append("aggarwal"); ⇒ O(n2) time

n2 = 8

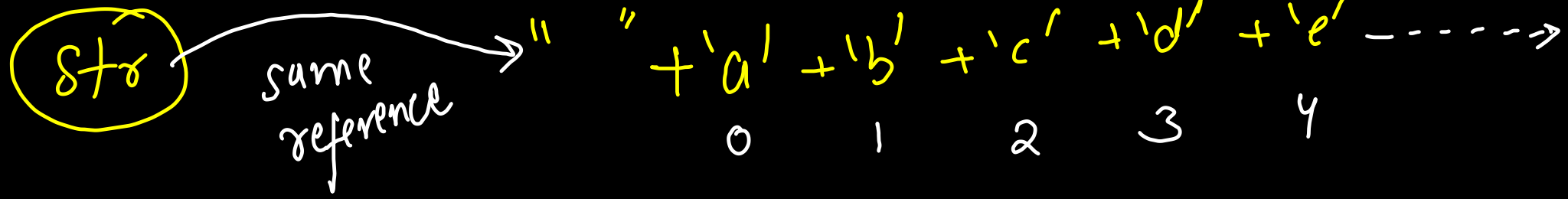
str.append('Z'); ⇒ O(1) constant

StringBuilder str = new StringBuilder();

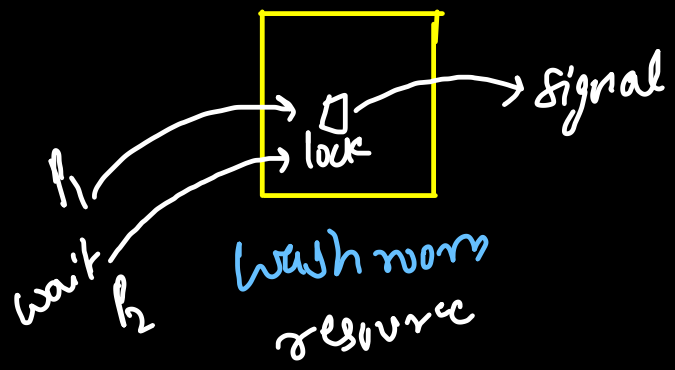
```
// N = 26
for(char ch = 'a'; ch <= 'z'; ch++){
    d1.append(ch);
}
```

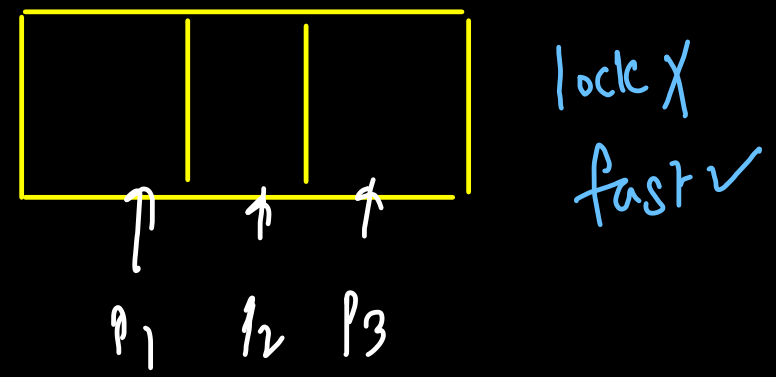
O(n) time (where n=26)

↳ O(1) constant



No.	StringBuffer (multithreading)	StringBuilder
1)	StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
2)	StringBuffer is *less efficient* than StringBuilder.	StringBuilder is *more efficient* than StringBuffer.
3)	StringBuffer was introduced in Java 1.0	StringBuilder was introduced in Java 1.5





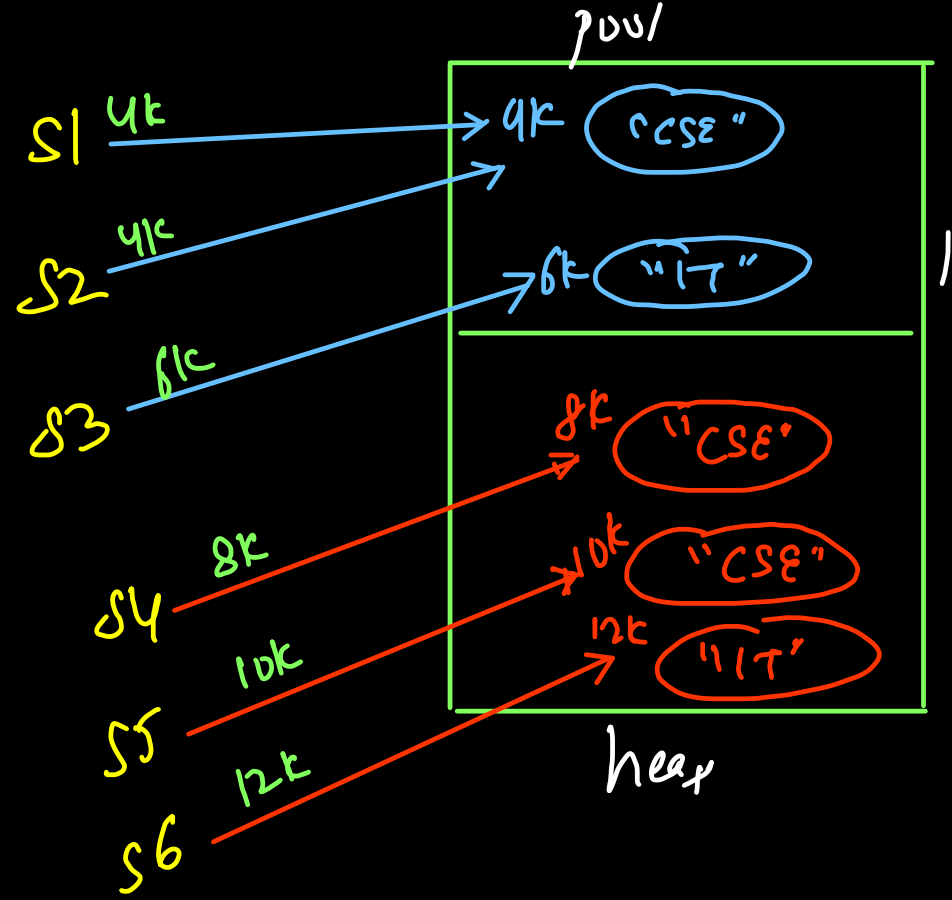
```java
String s1 = "CSE";
String s2 = "CSE";
String s3 = "IT";

// == operator: string literal
System.out.println(s1 == s2); // true
System.out.println(s1 == s3); // false

String s4 = new String("CSE");
String s5 = new String("CSE");
String s6 = new String("IT");

System.out.println(s4 == s5); // false
System.out.println(s1 == s4); // false
System.out.println(s4 == s6); // false
```

interning ✓
immutability ✓

4k == 4k ✓
4k =/= 6k X

immutability ✓
interning X

8k ≠ 10k
4k ≠ 8k
8k ≠ 12k



need new objects on every new string

backward compatibility

== operator: reference

same → true

diff → false

```java
String s1 = "CSE";
String s2 = "CSE";
String s3 = "IT";

// == operator: string literal
System.out.println(s1 == s2); // true
System.out.println(s1 == s3); // false

// equals function: data
System.out.println(s1.equals(s2)); // true
System.out.println(s1.equals(s3)); // false

String s4 = new String("CSE");
String s5 = new String("CSE");
String s6 = new String("IT");

System.out.println(s4 == s5); // false
System.out.println(s1 == s4); // false
System.out.println(s4 == s6); // false

System.out.println(s4.equals(s5)); // true
System.out.println(s1.equals(s4)); // true
System.out.println(s4.equals(s6)); // false
```

```
Finished in 79 ms
true
false
true
false
false
false
false
true
true
false
```

O(1) == operator compares on reference

O(n) worst case equals() compares first on reference, then on object (data)

```
public boolean equals(String s1, String s2){
    if (s1 == s2) return true;                    // O(1)
    if (s1.length() != s2.length()) return false; // O(1)
    for (int i=0; i<s1.length(); i++){
        if (s1.charAt(i) != s2.charAt(i))
            return false;
    }
    return true;
}
```

best case: reference

O(n): data

Wrapper class (reference ⇒ stack, object ⇒ heap)

- int ⟶ Integer
- char ⟶ Character
- double ⟶ Double
- boolean ⟶ Boolean

to provide extra functionalities

Stack (primiti

immutability

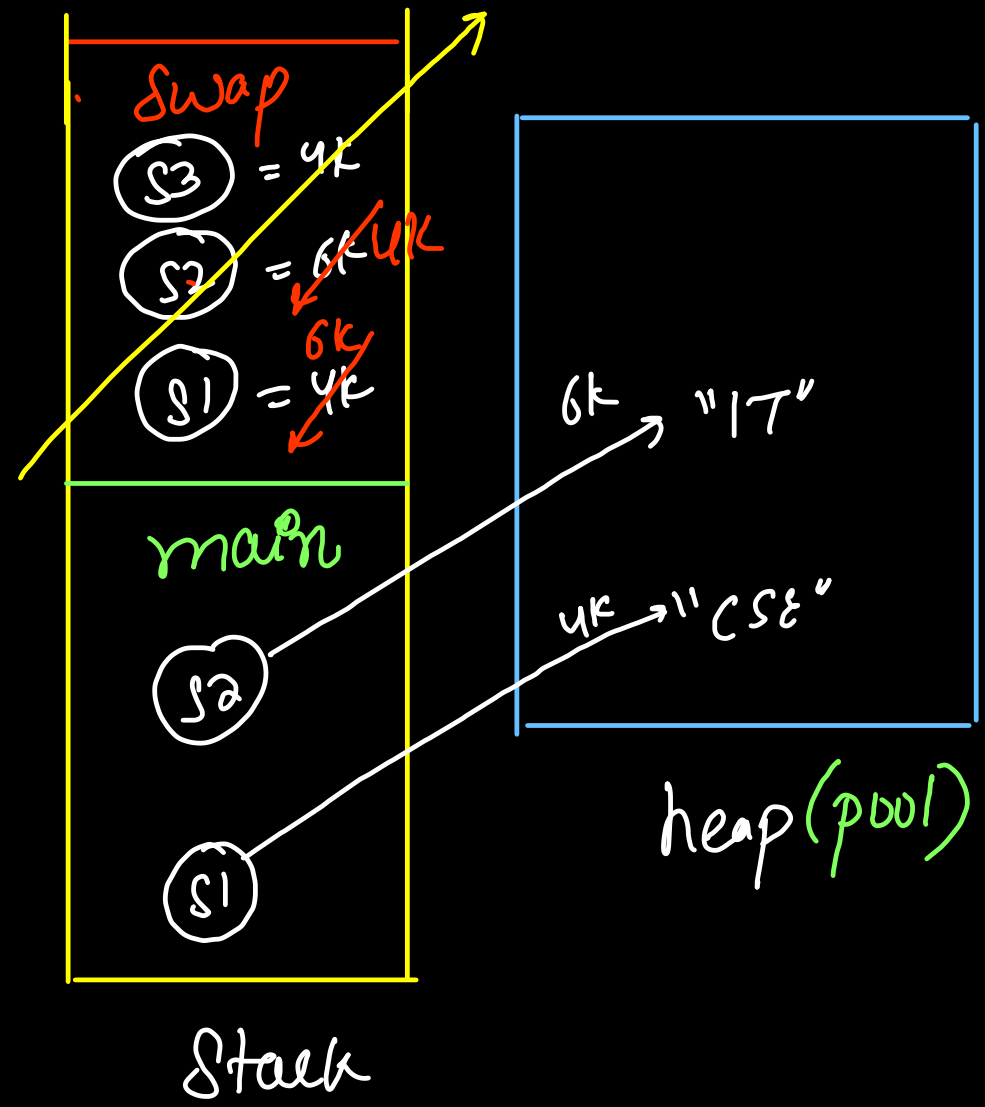
```java
public class Main {
    public static void main(String[] args) {
        String s1 = "CSE";
        String s2 = "IT";
        System.out.println(s1 + " " + s2);
        swap(s1, s2);
        System.out.println(s1 + " " + s2);
    }

    public static void swap(String s1, String s2){
        System.out.println(s1 + " " + s2);
        String s3 = s1;
        s1 = s2;
        s2 = s3;
        System.out.println(s1 + " " + s2);
    }
```

CSE IT

CSE IT

no swap → immutable string

IT CSE

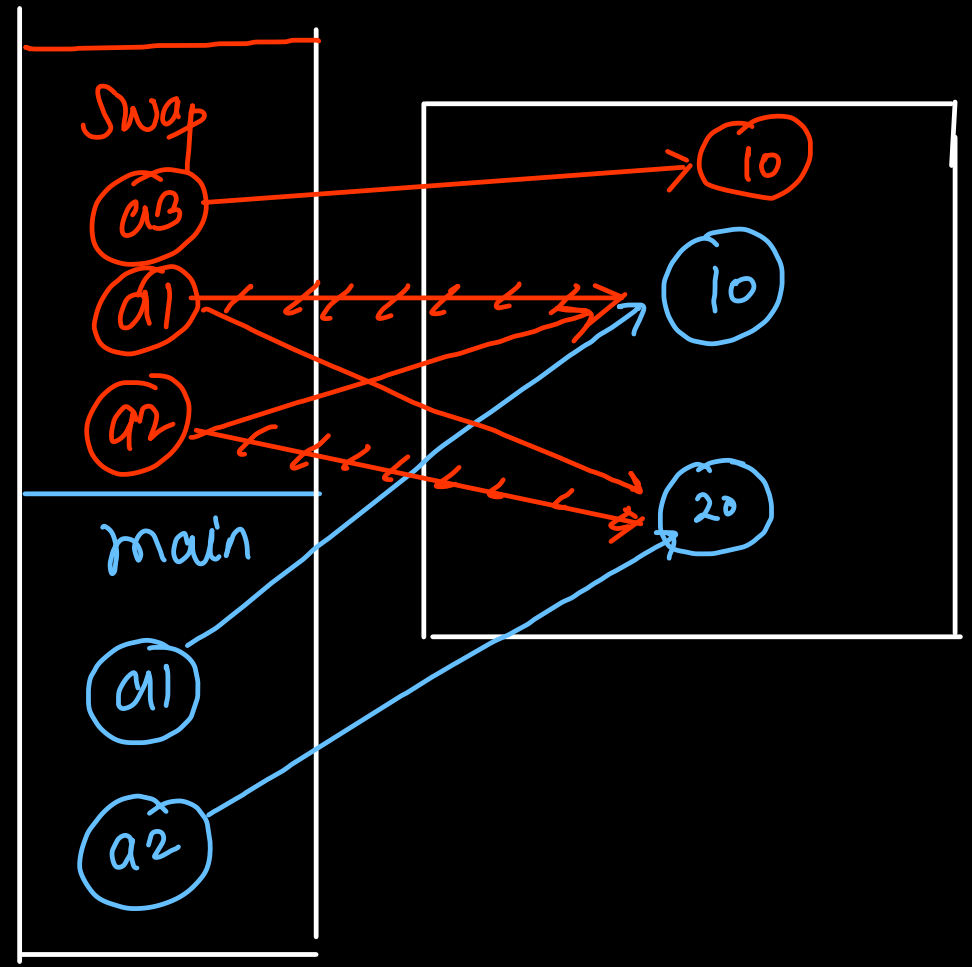


```java
public class Main {
    public static void main(String[] args) {
        Integer a1 = 10;
        Integer a2 = 20;
        System.out.println(a1 + " " + a2);
        swap(a1, a2);
        System.out.println(a1 + " " + a2);
    }

    public static void swap(Integer a1, Integer a2){
        System.out.println(a1 + " " + a2);
        Integer a3 = new Integer(a1);
        a1 = a2;
        a2 = a3;
        System.out.println(a1 + " " + a2);
    }
}
```

10 20

10 20

20 10



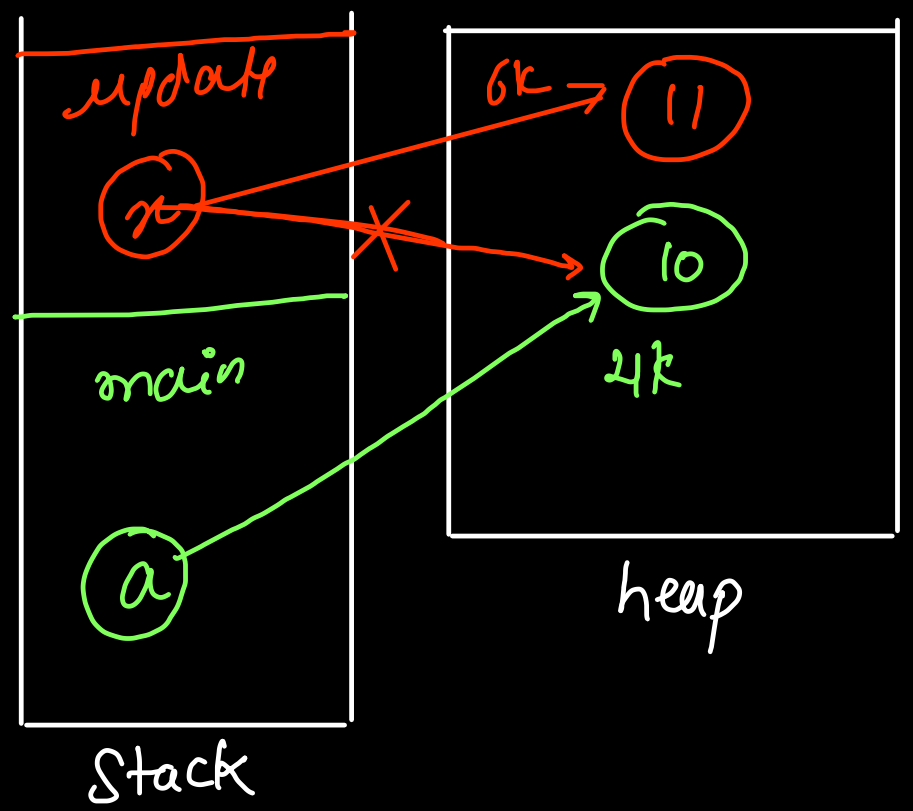
"Wrapper classes are immutable in Java"

```java
public class Main {
    public static void main(String[] args) {

        Integer a = 10;
        Systme.out.println(a);
        update(a);
        System.out.println(a);
    }

    public static void update(Integer x){
        System.out.println(x);
        x++;
        System.out.println(x);
    }
}
```

10

10

11



Lc (51) Reverse Words

eg 1) "the / sky / is / blue" ⟶ "blue . is . sky . the"

eg 2) " ⎵⎵ the ... sky .. is .... blue ⎵⎵ "

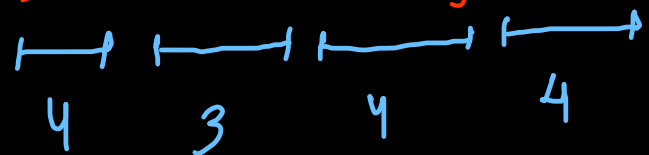
⎵⎵ leading space

⎵⎵ trailing spaces

approach ① Split the string based on spaces.

str.split("\s+"); → array of strings ~ { "the", "sky", "is", "blue" }
0 1 2 3

SB "blue - is - sky - the" ⇒ return as string




```java
public String reverseWords(String str) {
    String[] words = str.trim().split("\s+");
    // trim: leading & trailing spaces: discard
    // split on multiple spaces: single split

    StringBuilder res = new StringBuilder();
    for(int idx = words.length - 1; idx >= 0; idx--){
        if(idx < words.length - 1){
            res.append(" ");
        }
        res.append(words[idx]);
    }

    return res.toString();
}
```

no space before first word

time = O(n)
n = str.length()

space = O(n)
extra space (array)

```java
public String reverseWords(String str) {
    String[] words = str.split("\s+");

    StringBuilder res = new StringBuilder();
    for(int idx = words.length - 1; idx >= 0; idx--){
        res.append(" " + words[idx]);
    }

    return res.toString().trim();
}
```

input split on multiple spaces.

↳ concatenate new words in same output.

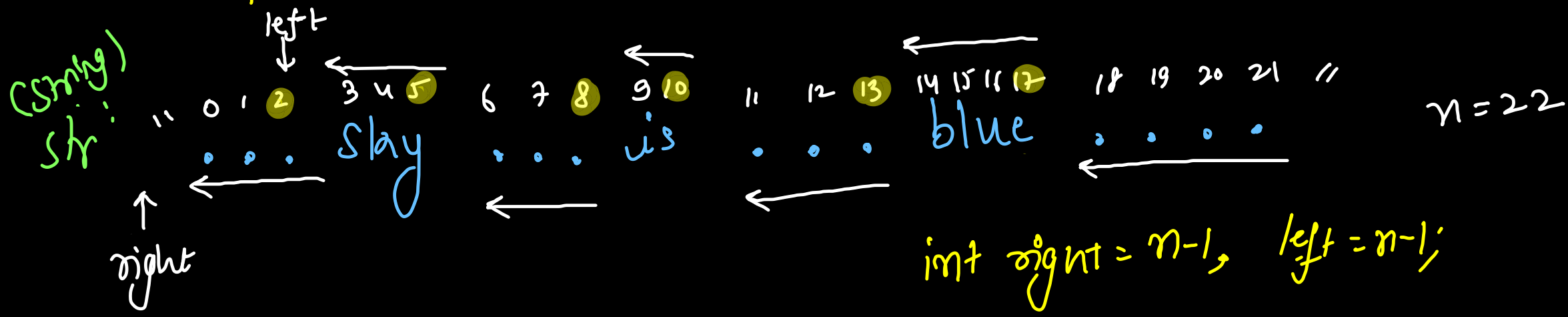
StringBuilder to String convert

leading & trailing spaces are removed

Lc 151

2nd way of 1st approach!

2nd approach : Two Pointer Technique



StringBuilder sb

" blue . is . slay "

```
int right = n-1, left = n-1;

while (right >= 0) {

    while (right >= 0 && str[r] == ' ')
        right--;

    left = right;
    while (left >= 0 && str[l] != ' ')
        left--;

    // append the word in output
}
```

```java
public String reverseWords(String str) {
    int right = str.length() - 1;
    StringBuilder res = new StringBuilder();

    // iterate on all words
    while( right >= 0 ){

        // right will stop at ending index of a word
        while(right >= 0 && str.charAt(right) == ' '){
            right--;
        }

        int left = right;
        // left will stop at starting index of a word - 1
        while(left >= 0 && str.charAt(left) != ' '){
            left--;
        }

        if(res.length() > 0 && left < right) {
            res.append(" ");
        }

        for(int idx = left + 1; idx <= right; idx++){
            res.append(str.charAt(idx));
        }

        right = left;
    }

    return res.toString();
}
```

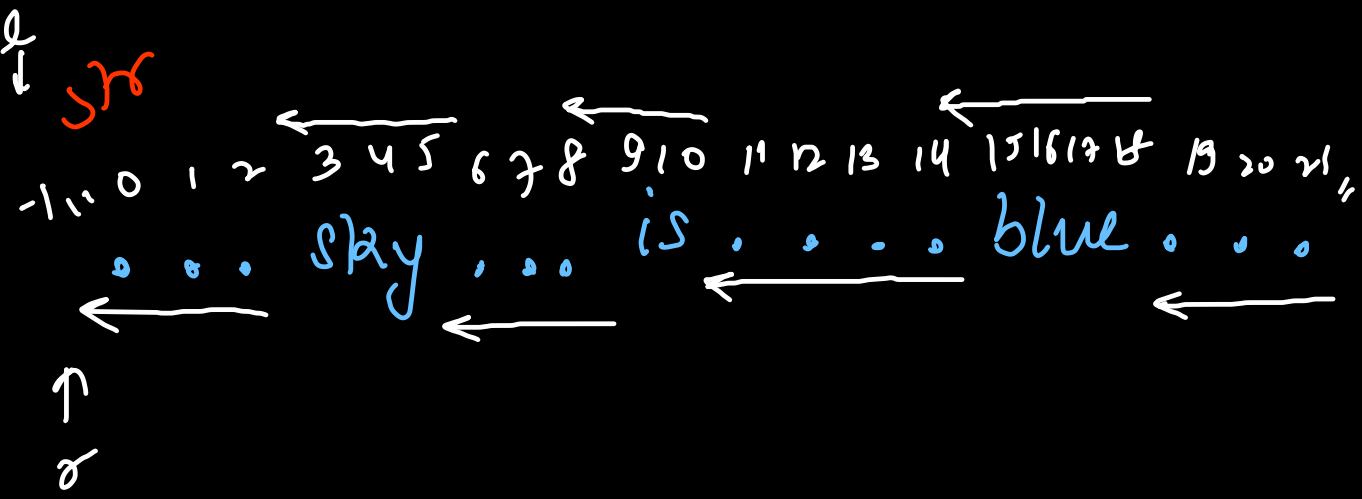
input

output

independent loops

not first word

empty word



res

" blue . is . sky "

2 Pointer

time ⇒ O(n) → linear

space ⇒ O(1)

no extra space

2 (557) Reverse Words - III

- no leading spaces
- no trailing spaces
- no multiple spaces

Input: "the · sky · is · blue"

↓

output: "eht · yks · si · eulb"

Input: "the sky is blue"

Output: "eht yks si eulb"

① approach 1) Split

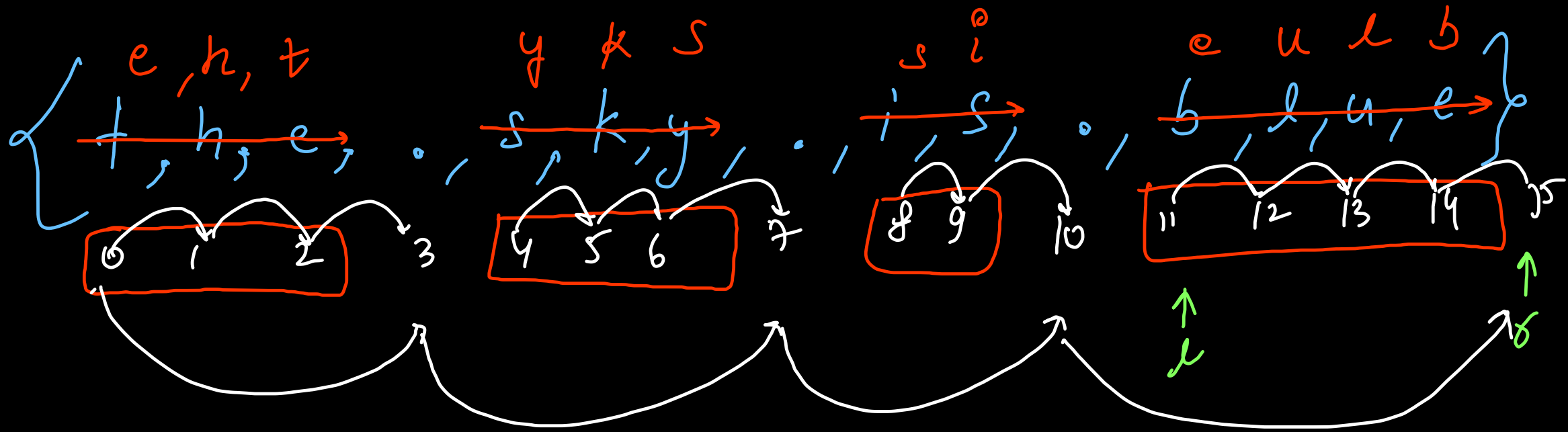
["the", "sky", "is", "blue"]

"eht" "yks" "si" "eulb"

Time ⇒ O(n) ↪ split, append

Space ⇒ O(n) (array)

② approach 2: character array + two pointer



```java
public void reverse(char[] res, int left, int right){
    while(left <= right){
        char temp = res[left];
        res[left] = res[right];
        res[right] = temp;
        left++; right--;
    }
}

public String reverseWords(String str) {
    char[] res = str.toCharArray();
    int left = 0;
    while(left < str.length()){
        int right = left;

        while(right < str.length() && res[right] != ' '){
            right++;
        }

        reverse(res, left, right - 1);
        left = right + 1;
    }

    return String.valueOf(res);
}
```

input

output

all words

all characters of a word

reverse word

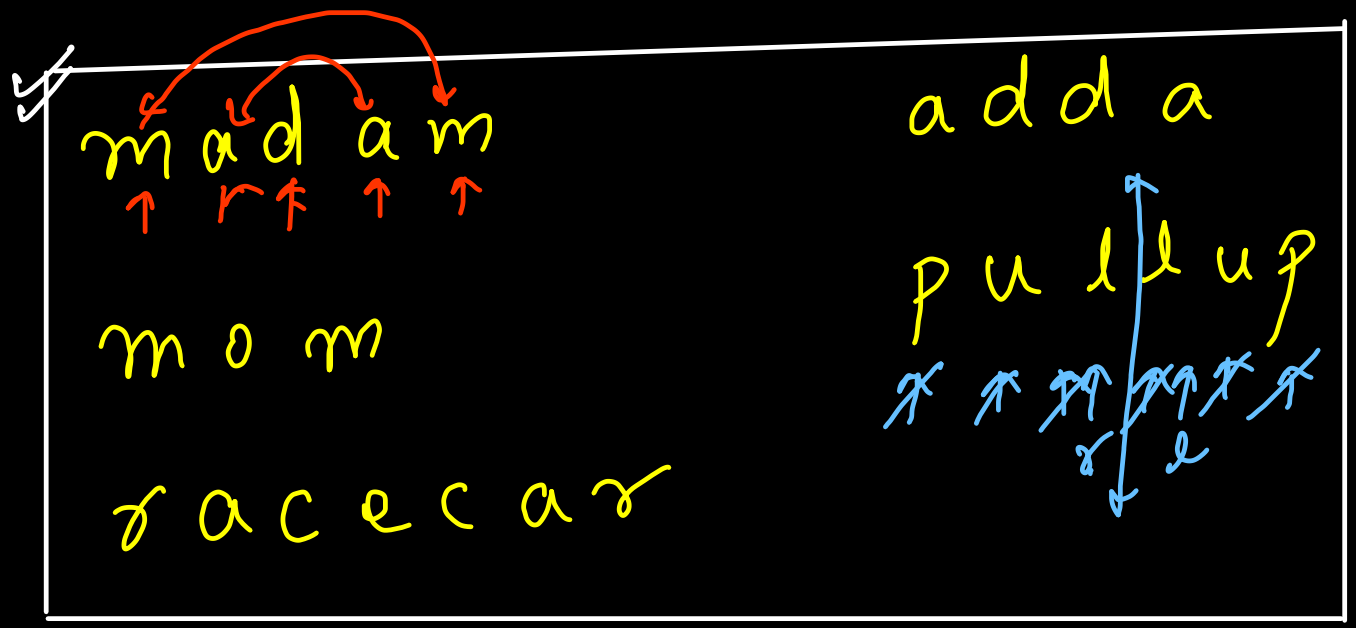
Time => O(n)
linear

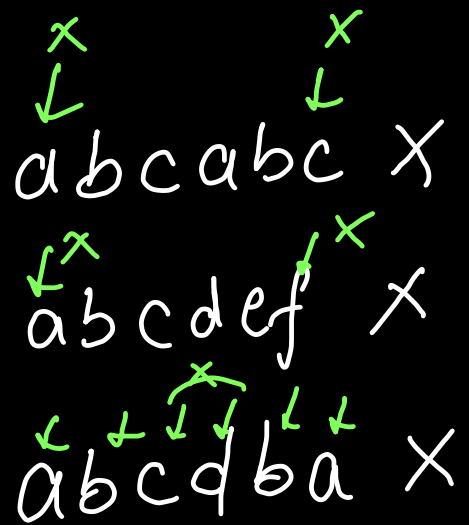
Space => O(1)
no extra space

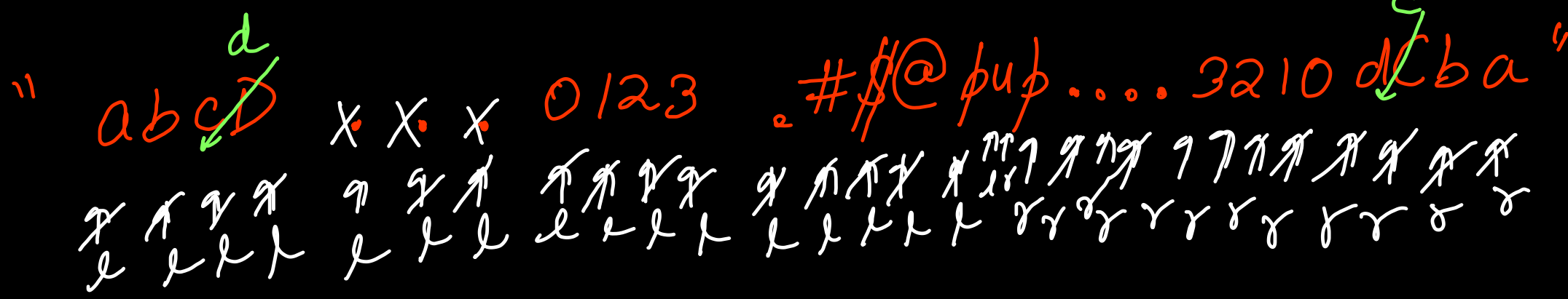
Valid Palindrome

[LC 125]

palindrome : left to right == right to left







- convert uppercase to lowercase ⟹ toLowerCase()
- remove all non-alphanumeric ⟹ two pointer (discard)
  isAlphanumeric()
  (s[i] - 'g' || 'a' - '2')

```java
public boolean isAlphaNumeric(char ch){
    if(ch >= 'a' && ch <= 'z') return true; // lowercase
    if(ch >= 'A' && ch <= 'Z') return true; // uppercase
    if(ch >= '0' && ch <= '9') return true; // digit
    return false;
}

public boolean isPalindrome(String str) {
    str = str.toLowerCase();
    int left = 0, right = str.length() - 1;

    while(left <= right){
        if(isAlphaNumeric(str.charAt(left)) == false){
            left++; continue;
        }

        if(isAlphaNumeric(str.charAt(right)) == false){
            right--; continue;
        }

        if(str.charAt(left) != str.charAt(right)){
            return false;
        }
        left++; right--;
    }

    return true;
}
```

discard non-alpha numeric

Time = O(n) (linear)

Space = O(1) (no extra space)

Big Integers

int ⟶ $2^{31}-1$ ⟶ $2 * 10^9$

Integer.MAX_VALUE

long ⟶ $2^{63}-1$ ⟶ 10^{15}

Long.MAX_VALUE

Bigger Numbers

- array
- strings

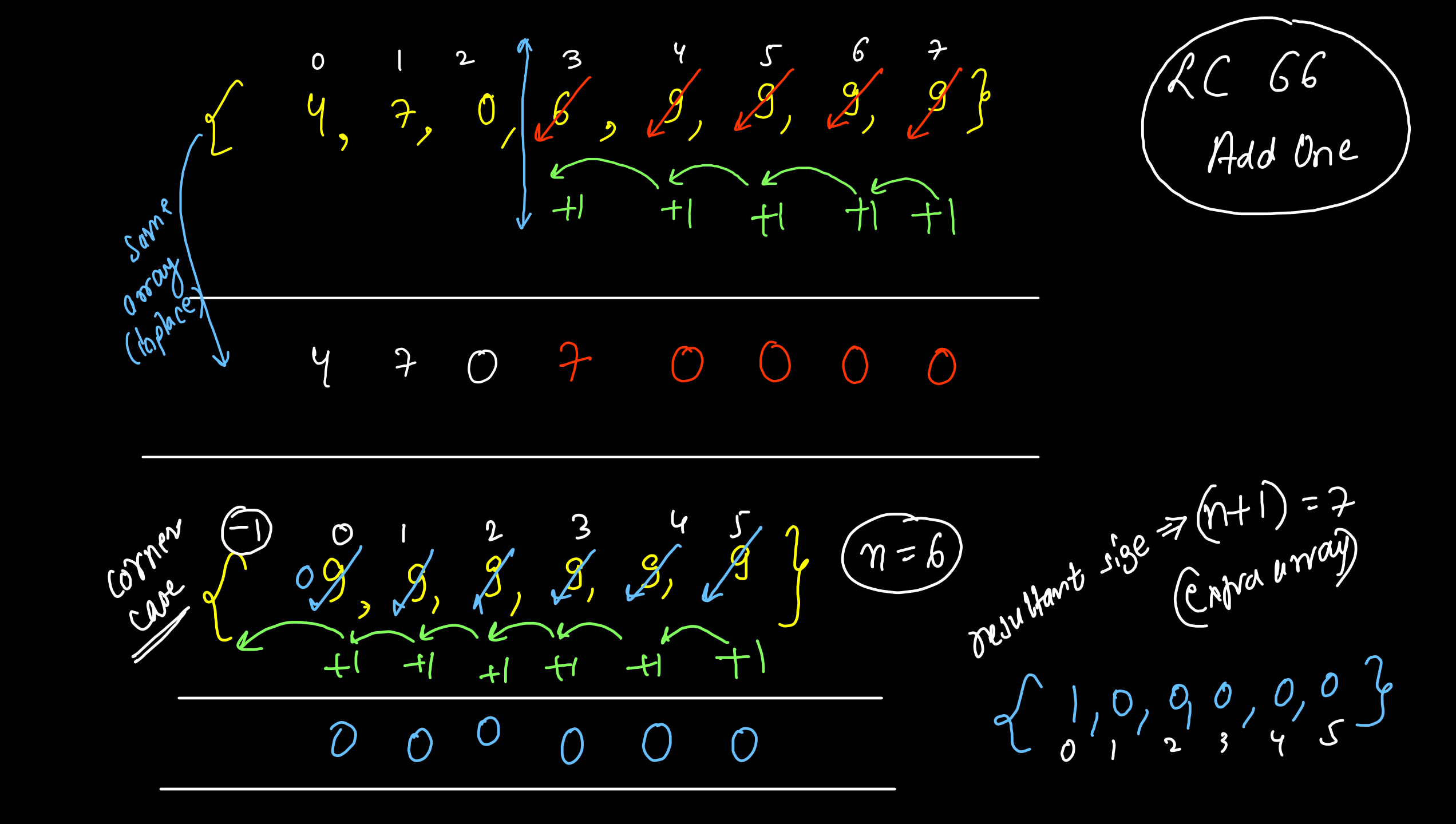
(store)

{2, 1, 7, 0, 0, 0, 0, 0 ---- 0}

"21700 ----- 0"

- BigInteger (Java)

↳ allowed ✗


LC 66
Add One
0 1 2 3 4 5 6 7
4, 7, 0, 6, 9, 9, 9, 9
+1 +1 +1 +1 +1
Same array (Inplace)
4 7 0 7 0 0 0 0
Corner Case
-1
0 1 2 3 4 5
0, 9, 9, 9, 9, 9
n = 6
+1 +1 +1 +1 +1 +1
0 0 0 0 0 0
Resultant size ⇒ (n+1) = 7
(Extra array)
1, 0, 0, 0, 0, 0
0 1 2 3 4 5

```java
public int[] plusOne(int[] digits) {
    for(int idx = digits.length - 1; idx >= 0; idx--){
        if(digits[idx] == 9){
            digits[idx] = 0;
        } else {
            digits[idx]++;
            return digits;
        }
    }

    // corner case: Input has all digits equal to 9
    int[] res = new int[digits.length + 1];
    res[0] = 1;
    return res;
}
```

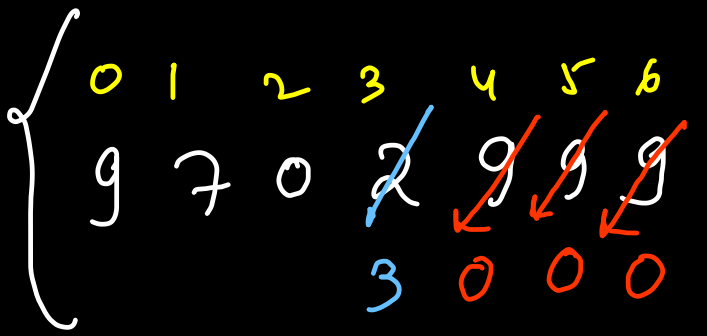
//carry

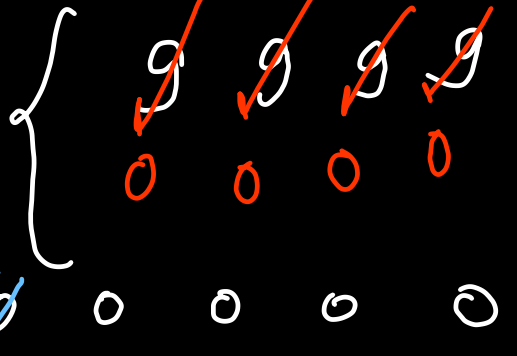
// usual case => O(1) no extra space (in place)

// corner case: O(n) extra space

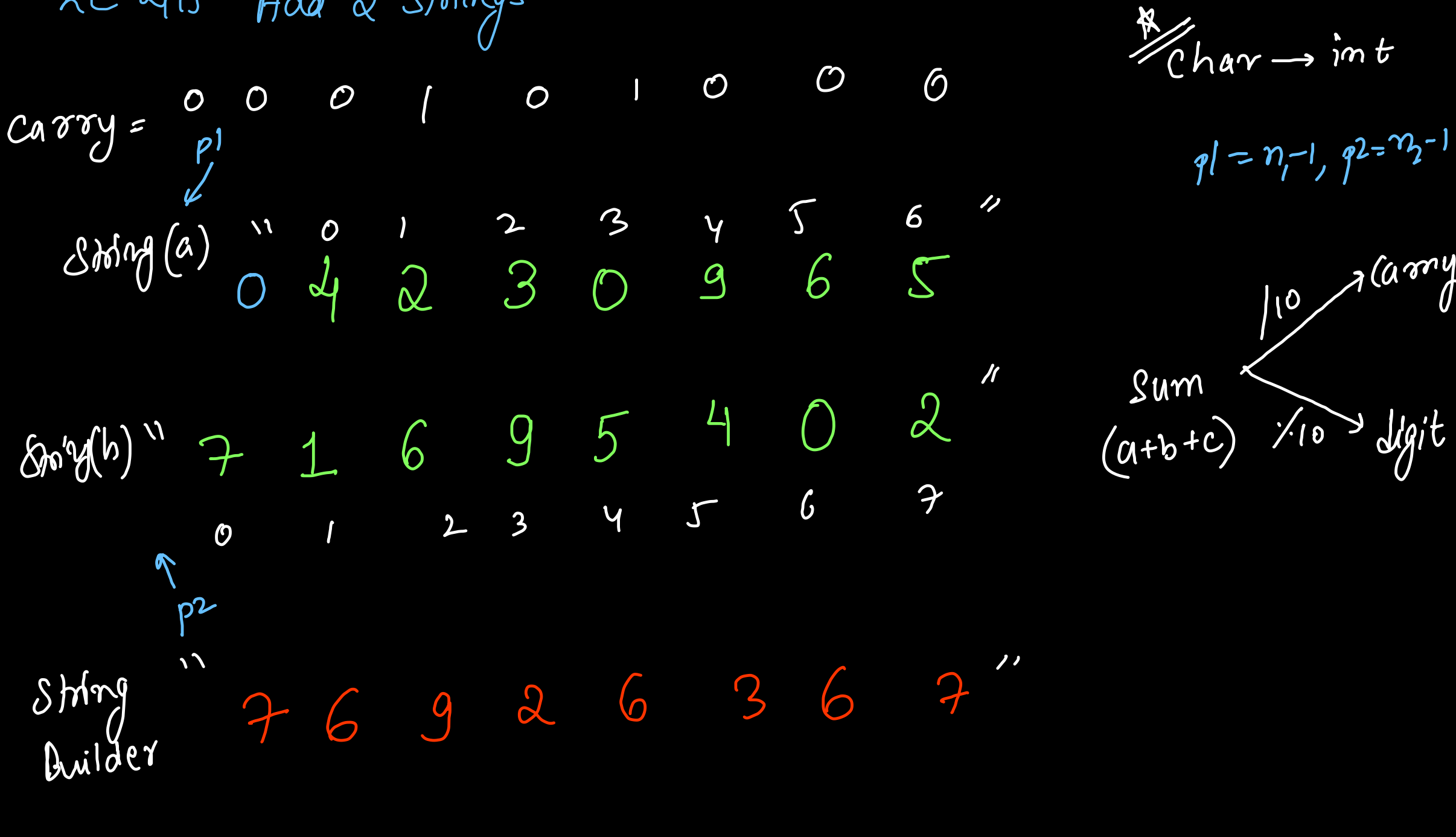
Time = O(n) linear

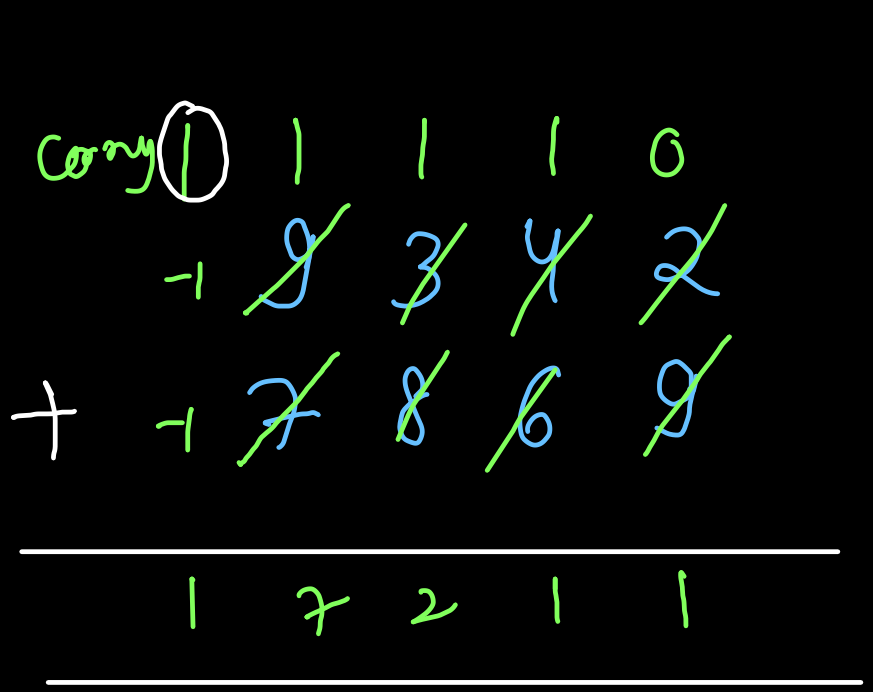
Space = O(n) worst case (corner)

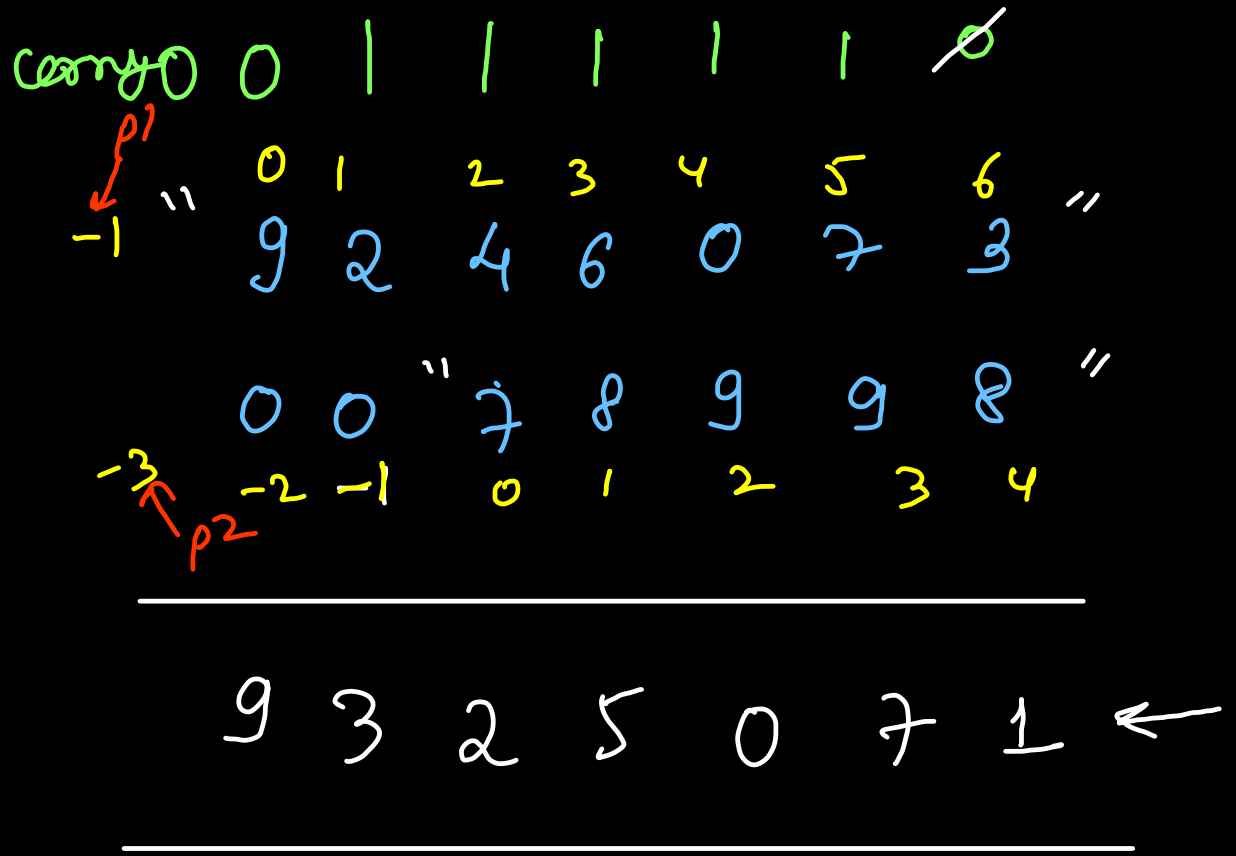




LC 415 Add 2 Strings




Carry (1) 1 1 1 0
-1 9 3 4 2
+ -1 7 8 6 9
1 7 2 1 1


Carry 0 0 1 1 1 1 1 0
p1
0 1 2 3 4 5 6
-1 " 9 2 4 6 0 7 3 "
0 0 " 7 8 9 9 8 "
-3 -2 -1 0 1 2 3 4
p2
9 3 2 5 0 7 1 ←

```java
public String addStrings(String num1, String num2) {
    StringBuilder res = new StringBuilder();
    int p1 = num1.length() - 1, p2 = num2.length() - 1;
    int carry = 0;

    while(p1 >= 0 || p2 >= 0 || carry > 0){
        int d1 = (p1 >= 0) ? num1.charAt(p1) - '0' : 0;
        int d2 = (p2 >= 0) ? num2.charAt(p2) - '0' : 0;
        int sum = d1 + d2 + carry;

        carry = sum / 10;
        res.append(sum % 10);
        p1--; p2--;
    }

    return res.reverse().toString();
}
```

Linear time

$O(\max(n_1, n_2))$

n_1 num1.length()

n_2 num2.length

Space ⇒ O(1)

Constant extra

(considering SB as output)

Subtract Two Numbers

borrow	-1	0	-1	0	0
	9	2	1	7	5
-	5	8	0	9	4
	3	4	0	8	1

```
int diff = (c+d1-d2)
if(diff<0) {
    diff += 10;
    c = -1;
}
else {
    c = 0;
}
```

R(43 Multiply Two Big Integers

2 3 1 0
0 4 7 3

× 5

2 3 6 5

4 6 2
0 4 7 3

× 90

4 2 5 7 0

4 1
4 7 3

× 600

2 8 3 8 0 0

+ + +

473 × 695

= (473 × 5) * 1
+
(473 × 9) * 10
+
(473 × 6) * 100

```java
public String addStrings(String num1, String num2) {
    StringBuilder res = new StringBuilder();
    int p1 = num1.length() - 1, p2 = num2.length() - 1;
    int carry = 0;

    while(p1 >= 0 || p2 >= 0 || carry > 0){
        int d1 = (p1 >= 0) ? num1.charAt(p1) - '0' : 0;
        int d2 = (p2 >= 0) ? num2.charAt(p2) - '0' : 0;
        int sum = d1 + d2 + carry;

        carry = sum / 10;
        res.append(sum % 10);
        p1--; p2--;
    }

    return res.reverse().toString();
}
```

Time = $O(n^2)$ quadratic

Space = $O(n)$ extra space

```java
public String multiply(String num1, String num2) {
    String res = "0";
    int count = 0;
    for(int idx = num2.length() - 1; idx >= 0; idx--){
        int d2 = num2.charAt(idx) - '0';
        String temp = multiply(num1, d2);
        for(int c = 0; c < count; c++){
            temp += '0';
        }

        res = addStrings(res, temp);
        count++;
    }

    if(res.charAt(0) == '0') return "0";
    return res;
}
```

shifting { 4 zeros

Corner case "0000" ⇓ "0"

```java
public String multiply(String num1, int d2){
    StringBuilder res = new StringBuilder();

    int p1 = num1.length() - 1;
    int carry = 0;

    while(p1 >= 0 || carry > 0){
        int d1 = (p1 >= 0) ? num1.charAt(p1) - '0' : 0;
        int prod = d1 * d2 + carry;

        res.append(prod % 10);
        carry = prod / 10;
        p1--;
    }

    return res.reverse().toString();
}
```

473 × 5 = 2365

473 × 9 = 4257

473 × 6 = 2838

473 × 695

res "0" + "2365"

= "2365" + "42570" + "283800"

Frequency, Substring & Subsequence

rearrangement → permutations

Leetcode 242) Valid anagram

"anagram" ⟺ "nagaram"

true

"car" ⟺ "rat"

false

① s1.length() == s2.length()

② Characters (frequency) same

brute force ⇒ Generate all permutations

Time ⇒ $O(n!)$ ⇒ exponential (TLE)

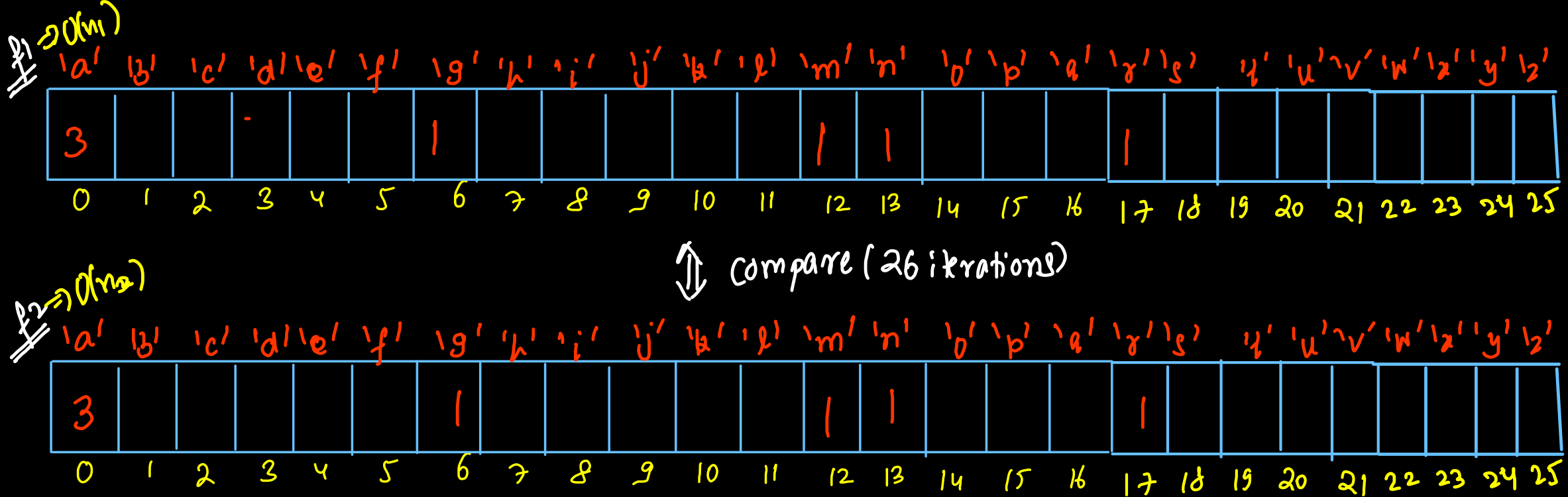
"anagram" ⟺ "nagaram"

0-127 ascii

↳ 128 size

26 distinct letters ('a'-'z') ⇒ 26 size

freq['a'] ⇒ freq[97]



```java
public int[] getFrequency(String str){
    int[] freq = new int[26];
    for(char ch : str.toCharArray()){
        freq[ch - 'a']++;
    }
    return freq;
}

public boolean isAnagram(String s1, String s2) {
    if(s1.length() != s2.length()){
        return false;
    }

    int[] f1 = getFrequency(s1);
    int[] f2 = getFrequency(s2);

    for(int idx = 0; idx < 26; idx++){
        if(f1[idx] != f2[idx])
            return false;
    }

    return true;
}
```

O(n) n = str.length()

O(n1)

O(n2)

O(26) = O(1)

Time = O(n1 + n2)

Space = O(26 + 26)
extra space
≈ O(1) inplace

LC 1832) Pangram

```
Input: sentence = "thequickbrownfoxjumpsoverthelazydog"
```

→ true

Every character in lowercase English alphabets

① frequency array build : O(n)

② freq['a'..'z'] ≥ 1 ∀ 'a'..'z'

LC 205) Isomorphic Strings

"egg" —true→ "add"

e → a

g → d

1:1 ✓

"foo" —false→ "bar"

f → b

o → a, r

1: many X

"paper" —true→ "title"

p → t

a → i

e → l

r → e

1:1 ✓

"bar" —false→ "foo"

b → f

a → o

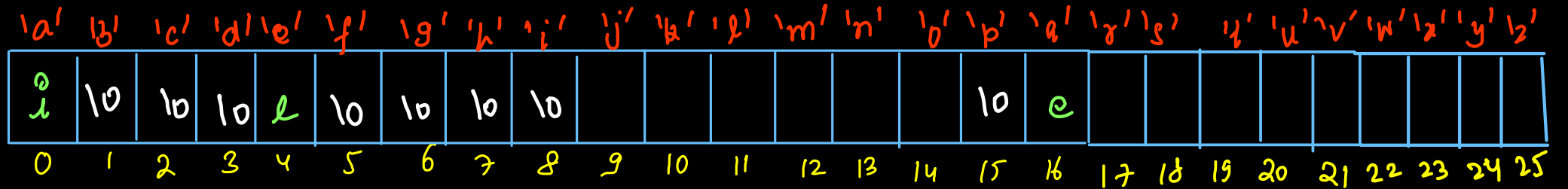
r → o

many:1 X

No two characters should have same mapping

Character may map to itself.

array (map) ⟹ character array



"paper" ⟷ "title"

p→t, a→i, e→l, r→

t→p, i→a, l→e, e→r

"foo" → "bar"

f→b
o→a

'o' ≠ 'r' ⇒ false

1 : many

"bar" → "foo"

b→f
a→o
r→o

many : 1

false

f→b
o→a
r

```java
public boolean mapOneToOne(String s1, String s2){
    char[] map = new char[256]; //ascii code

    for(int idx = 0; idx < s1.length(); idx++){
        char ch1 = s1.charAt(idx);
        char ch2 = s2.charAt(idx);

                                default value
        if(map[ch1] != '\u0000' && map[ch1] != ch2)
            return false;
        map[ch1] = ch2;
    }

    return true;
}

public boolean isIsomorphic(String s1, String s2) {
    return mapOneToOne(s1, s2) && mapOneToOne(s2, s1);
}
```

ch1

paper

p → t
a → i
e → l
r → e

ch2

title

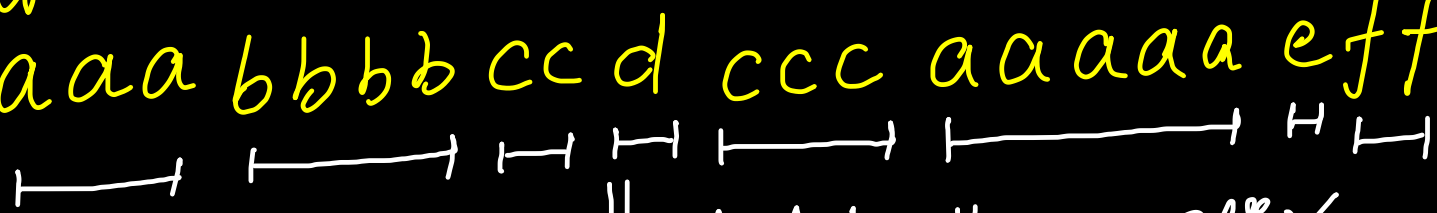
title

t → p
i → a
l → e
e → r

paper

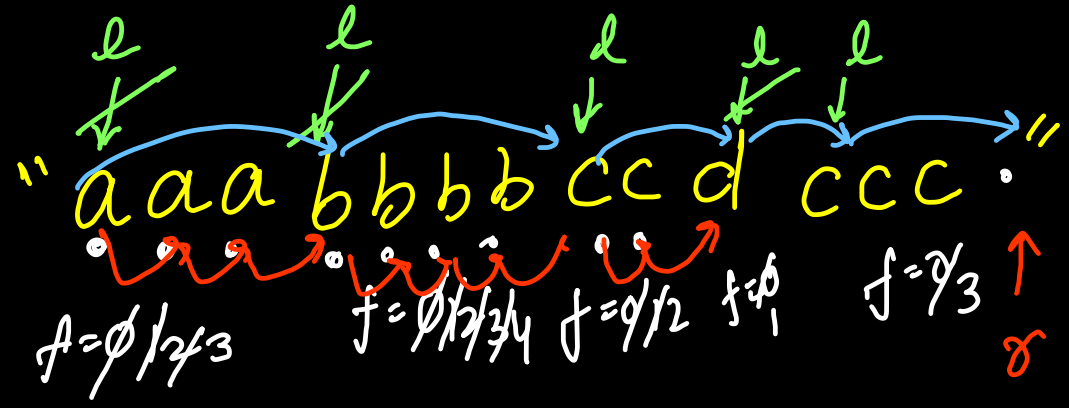
String Compression
(Run length Encoding)

input "aaabbbbccdcccaaaaaeff"



⇓ info loss ✗ compress ✓

output "a3b4c2dc3a5ef2"



consecutive
⇓
freq
⇓
cumulative

"a3b4c2dc3"

Interviews

Time ⇒ O(n) time
linear

Space ⇒ O(1) no extra
(SB as output)

```
String encode(String str)
{
    StringBuilder res = new StringBuilder();

    int left = 0, right = 0;
    while(left < str.length()){
        char ch = str.charAt(left);
        int freq = 0;

        while(right < str.length() && str.charAt(right) == ch){
            freq++; right++;
        }

        res.append(ch);
        res.append(freq);
        left = right;
    }

    return res.toString();
}
```

left ⇒ O(n) + O(n)

nested loops

brute force (nested loops) —reason→ optimized (a pointer)

Time & Space complexity ⇒ dry run → proper reasoning ✓

team work / communicatn ⟹ interest / focus (Continuous)

good companies ⟹ Introductn → short, crisp → 3-5 mins

Subarray / Substring

part of an array
- # continuous
- # same order

→ empty subarray
⋮
→ whole array

"chit" (0 1 2 3) n = 4

total substrings
$= \frac{n*(n+1)}{2}$

c h i t
ch hi it
chi hit
chit

$1+2+3+\cdots n$
$= \frac{n\times(n+1)}{2}$

```java
static void printSubstrings(String str) {
    for(int start = 0; start < str.length(); start++){
        for(int end = start; end < str.length(); end++){
            for(int idx = start; idx <= end; idx++){
                System.out.print(str.charAt(idx));
            }
            System.out.println();
        }
    }
}
```

N → N → N } $O(N^3)$ cubic

Str = Code

0 1 2 3

start = 0

end = 0 [0,0] ⇒ "c"

1 [0,1] ⇒ "co"

2 [0,2] ⇒ "cod"

3 [0,3] ⇒ "code"

start = 1

end = 1 [1,1] ⇒ "o"

2 [1,2] ⇒ "od"

3 [1,3] ⇒ "ode"

start = 2

end = 2 [2,2] ⇒ "d"

3 [2,3] ⇒ "de"

start = 3

end = 3 [3,3] ⇒ "e"

str.substring(left) :→ [left, n)

time: O(substring length)

"code" (0 1 2 3)

suffix substring

- substring(0) ⇒ "code"
- substring(1) ⇒ "ode"
- substring(2) ⇒ "de"
- substring(3) ⇒ "e"

str.substring(left, right)

:→ [left, right)

included excluded

substring length ⇒ right - left

- substring(0,4) ⇒ "code" 4-0 = 4
- substring(1,3) ⇒ "od" 3-1 = 2
- substring(2,2) ⇒ "" 2-2 = 0
- substring(1,4) ⇒ "ode" 4-1 = 3

```java
static void printSubstrings(String str) {
    for(int start = 0; start < str.length(); start++){
        for(int end = start; end < str.length(); end++){
            for(int idx = start; idx <= end; idx++){
                System.out.print(str.charAt(idx));
            }
            System.out.println();
        }
    }
}
```

N → N → N

$O(N^3)$ cubic (polynomial)

```java
static void printSubstrings(String str) {
    for(int start = 0; start < str.length(); start++){
        for(int end = start; end < str.length(); end++){
            System.out.println(str.substring(start, end + 1));
        }
    }
}
```

N → N → N

$O(n^3)$ cubic

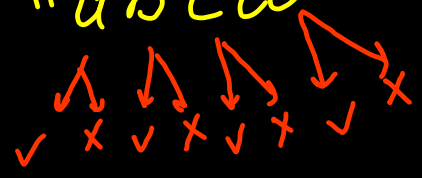
Subsequence

- part of a string
  - empty string
  - ⋮
  - complete string
- need not be contiguous
- order should be maintained

$N_{max} = 15-18$

n = 4

"abcd"



$2 * 2 * 2 * \cdots$ N times

2^N Subsequences (exponential) *

$2^4 = 16$ subsequences

= ""	= "d"
= "a"	"ad"
= "b"	"bd"
= "ab"	"abd"
= "c"	= "cd"
"ac"	"acd"
= "bc"	= "bcd"
= "abc"	= "abcd"

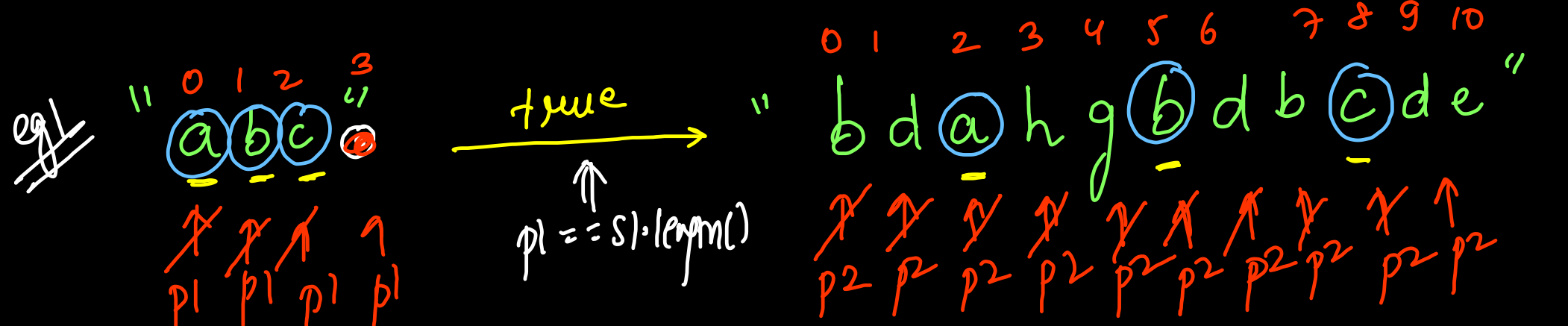
Subsequence : "remaining string after removal of 0 or more characters from the original string"

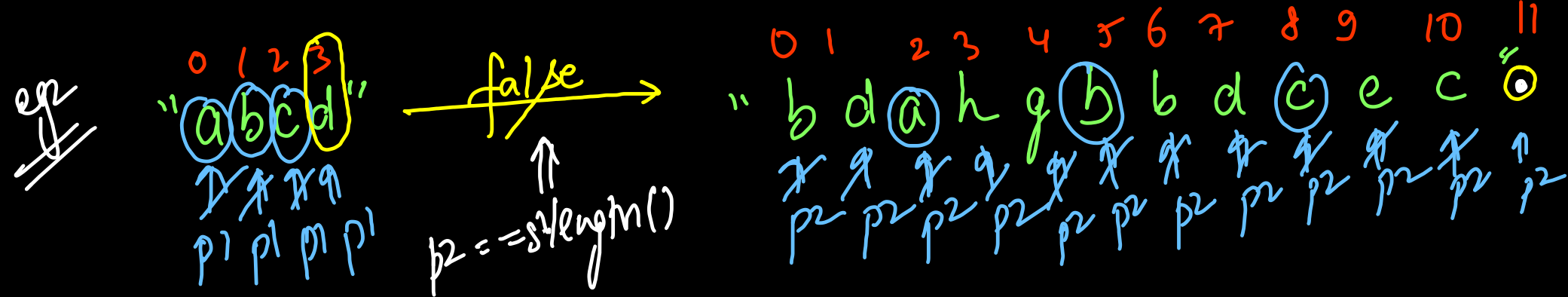
All Substrings are Subsequences ✓

or

All subsequences are substrings ✗

LC 392) Is Subsequence





```java
public boolean isSubsequence(String s1, String s2) {
    int p1 = 0, p2 = 0;

    while(p1 < s1.length() && p2 < s2.length()){
        if(s1.charAt(p1) == s2.charAt(p2)){
            p1++; p2++;
        } else {
            p2++;
        }
    }

    return (p1 == s1.length());
}
```

n_1 n_2

Time $\Rightarrow O(n_1+n_2)$

linear

Space $\Rightarrow O(1)$

constant

(Inplace)

Two Pointer Technique

Find Longest word in Dictionary (GFG)

Example 1:

Input: d = {"ale", "apple", "monkey", "plea"}
S = "abpcplea"

Output: "apple"

Explanation: After deleting "b", "c"
"a" S became "apple" which is present
in d.

eg2 S = "abcdefgh"

dict = {"xyz", "ade", "afg"}

tie breaker
ade afg
lexicographical

eg3 S = "abcdefg"

dict = {"axy", "xya", "cdx"}

answer = ""

" a b c d d c x x y z "
0 1 2 3 4 5 6 7 8 9

First Unique Character
answer = 'a'

First Repeating character
answer = 'c'

" a b c d c b a d "

First Unique Character
answer = -1

First Repeating character
answer = 'a'

" a b c d e f g "

first unique = 'a'

first repeating = -1

Bruteforce: Nested loops :⟶ $O(n^2)$ time
$O(1)$ space

frequency array ⟶ character? ⟶ unique c = 1
⟶ repeating > 1

if only english lowercase ⇒ 26 → 'a'
ascii characters ⇒ 256 → ~~'a'~~

Space = O(1) inplace (Constant)

" a b c d d c x x y z "
0 1 2 3 4 5 6 7 8 9

a → 1, b → 1, c → 2, d → 2, x → 2
y → 1, z → 1

unique, repeating

```java
public int firstUniqChar(String str) {
    int[] freq = new int[26];

    // Fill the Frequency Array
    for(char ch: str.toCharArray()){
        freq[ch - 'a']++;
    }

    // Find the First Unique or Non Repeating or Distinct Character
    for(int idx = 0; idx < str.length(); idx++){
        char ch = str.charAt(idx);
        if(freq[ch - 'a'] == 1) return idx;
    }

    return -1;
}
```

LC 387

2 Loops

Time = O(n) linear

Space = O(1)
Constant / inplace

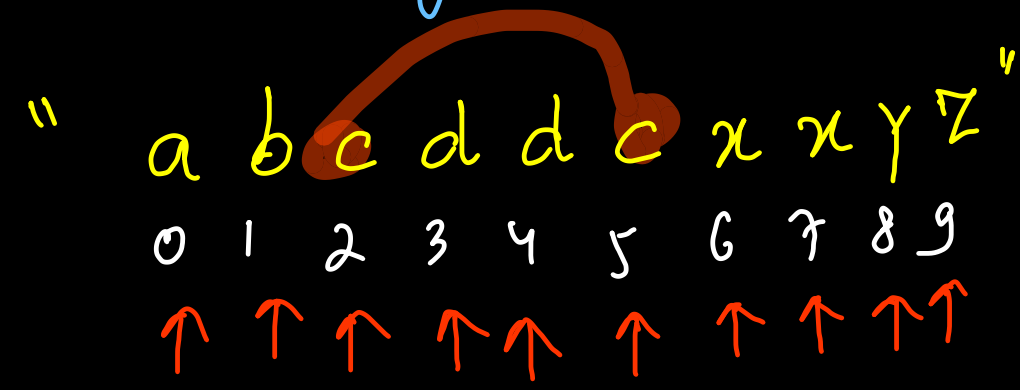
GFG

Can be 1 loop done in

```java
char firstRep(String str)
{
    int[] freq = new int[26];

    // Fill the Frequency Array
    for(char ch: str.toCharArray()){
        freq[ch - 'a']++;
    }

    // Find the First Unique or Non Repeating or Distinct Character
    for(char ch : str.toCharArray()){
        if(freq[ch - 'a'] > 1) return ch;
    }

    return '#';
}
```

Direction: Right to Left

first repeating character in single iteration



z : 1

y : 1

x : 1/2

c : 1/2

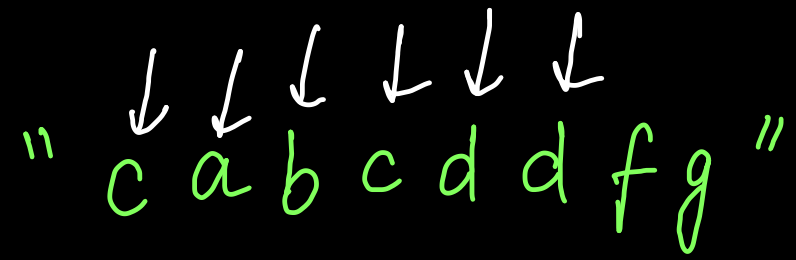
d : 1/2

b : 1

a : 1

answer = ~~'#'~~ ~~'x'~~ ~~'d'~~ 'c'

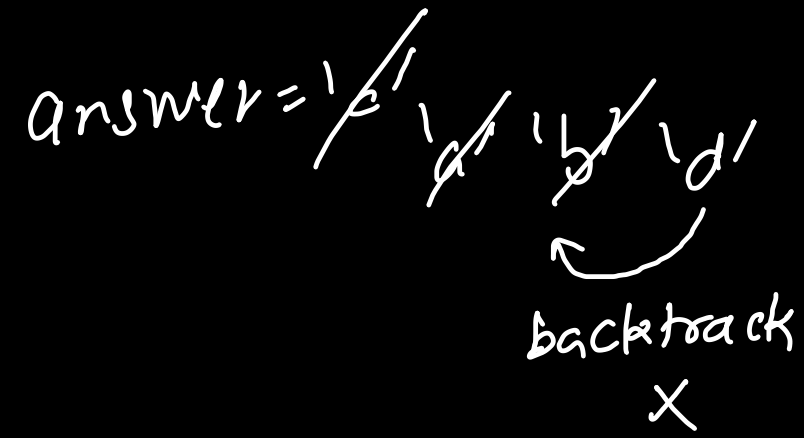
```
char firstRep(String str)
{
    int[] freq = new int[26];
    char repeating = '#';

    for(int idx = str.length() - 1; idx >= 0; idx--){
        char ch = str.charAt(idx);
        freq[ch - 'a']++;
        if(freq[ch - 'a'] > 1) repeating = ch;
    }

    return repeating;
}
```



c : 1/2 d : 1/2

a : 1

b : 1



First Unique Character cannot be solved in single loop