Q

Approach 1 :-

n = 13

Ap

Approach 1 :- ✓

n = 13

2 corner case ✓

```
for ( i = 2 ⟶ n ) {
    if ( n % i == 0 )  return false
}
return true
```

TC = O(N) ←

SC = O(1) ←

2 ⟶ 12 | 13

12 ⟶ 3

n = 12 199 ✓

if n is large like

n = 19999 3 { Time ↑

Approach 2 :- Efficient

2 ⟶ n ✗

2 ⟶ √n ✓

```
for ( 2 ⟶ √n ) {
    if ( n % i == 0 )  return false
```

if [n % i == 0]

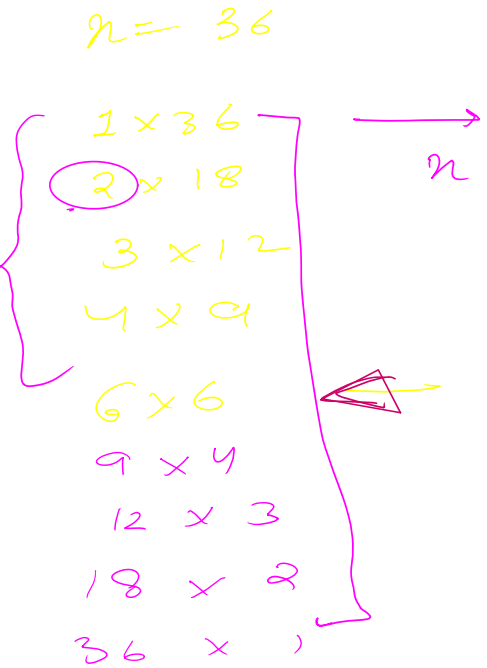
return true

TC → $O(\sqrt{n})$

SC → O(1)

```
Divisors of 36:

1 × 36
2 × 18
3 × 12
4 × 9
6 × 6   <--- Square root meeting point (√36 = 6)
9 × 4
12 × 3
18 × 2
36 × 1
```



Sieve of Eratosthenes :-

[non-prime]

Case 1: Brute force

$O(n \times n) = O(n^2)$

for(i = 1 → n)

Prime(i) ✓

✗

10^6 10^{20} 10^{28}

Case 2: Square Root

$O(n \times \sqrt{n})$

Case 1 < case 2 ✗

Case 3: Generate primes in range

$O(n \times \sqrt{n})$ → $\sqrt{n}$ → $O(1)$

n = 13 primes.

Step 1: array → n+1

Step 2 :- 14 size

4
5
6
7
8
9
10
11
12
13

0 1 2 3 4 5 6 7 8 9 10 11 12 13

2 5 11
3 7 13

n = 13

2	5	11
3	7	13

$n = 13$

Code :-

arr[0] = arr[1] = 0 ✓

for (i = 2 ⟶ n) {

if (arr[i] == 1) {

// mark factors/multiples as 0

}

```
1) arr[ ] = new int[n+1]                          ✓
2) for(i=2 → n) {                                 ✓
       arr[i] = 1;
   }
3) for (i=2 → n) {
       if (arr[i]==1){
           for (j= 2*i ; j<= n; j+=i) {
               arr[i]= 0
           }
       }
4) count = 0
   for (j=2 → n){
       if (arr[i] == 1){
           count++;
       }
   }
   return count;
```

i = = 2

j = 2 × 2

j = 4

j += i

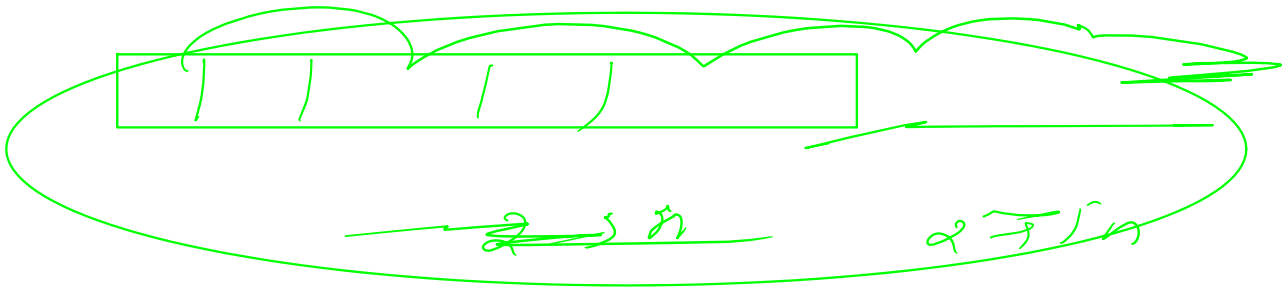
j = 2 + i = 4 (2 → 4)

j = 2i ✓

j = i × i

2 → n ×

$2 \rightarrow \sqrt{n}$ for $(2 - \sqrt{n})$



for j = 2 ←

"1 2 3 4 4 9 2"

largest odd no. = ✓

Even odd

↓ ↓

0, 2, 4, 6, 8 1, 3, 5, 7, 9

"1 2 3 4 5 6"

1 2 2 2 2 9 9 9 2 ✗

i 9 → digit

9 2 3 5 9 6

i 9 → digit
↓
number
↓
largest

Sub(start, end)

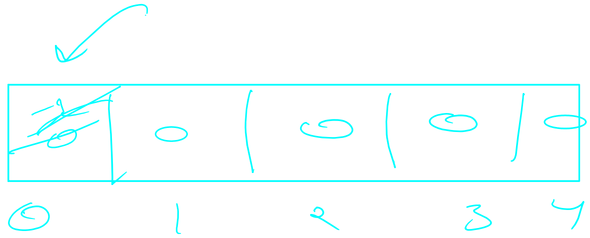
Check Anagram:-

str1 = "BOOB"
str2 = "BOBO" } anagram
↓
sequential arrangement

Sort → BOOB == ✓ ✗
BOOB

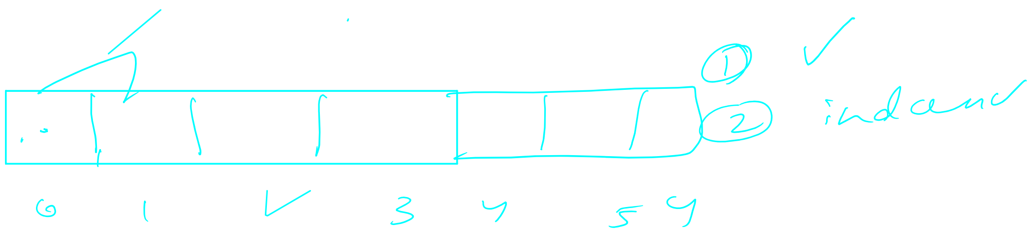
for [char —] sort ✓

Counting Sort



for (char y : str.toCharA

for(o



Approach : B

for(i=0→n){

for(j=i+1→n)

arr[i] + arr[j] ==

$O(n^2)$

$O(1)$

Strings

array

arr [ ] = ["Azaan", "Subhail", "Digvi

arr (n-1).length

Maximum Consecutive 1's :-

Example 1:

Input: nums = [1,1,0,1,1,1] o/p = 3
Output: 3
Explanation: The first two digits or the last three digits are consecutive 1s. The maximum number of consecutive 1s is 3.

Example 2:

Input: nums = [1,0,1,1,0,1]
Output: 2

Consecutive

Step 1 max_count = $-\infty$

Step 2: if 1 → count++ | max_count =

else → count = 0

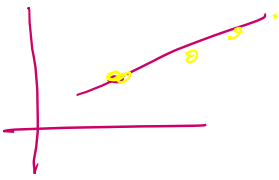
Montonic Arrays :-

Input: nums = [1,2,2,3]
Output: true

Example 2:

Input: nums = [6,5,4,4]
Output: true

Case 1:



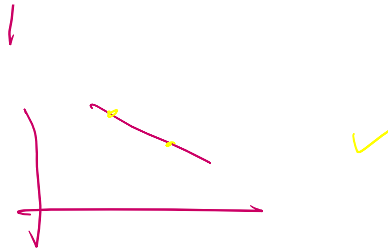
Example 2:

Input: nums = [6,5,4,4]
Output: true

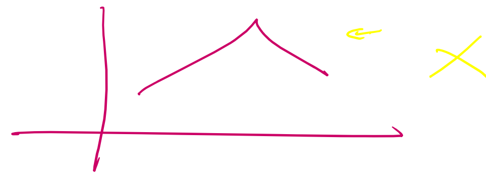
Example 3:

Input: nums = [1,3,2]
Output: false

Case



Case 3:



~~incase~~

inc = true
dec = true

1) for(i=1 → n)   arr[i] > arr[i-1] → decreas = false

2) for (j=1 → n)   arr[i] → incr → fals

if (incr == tr) ↑ ✓
else (dec == tr) ↓ ✓

return false   Case 3 —

x % n