344. Reverse String

Easy 8239 1147 Add to List Share

Write a function that reverses a string. The input string is given as an array of characters `s`.

You must do this by modifying the input array in-place with `O(1)` extra memory.

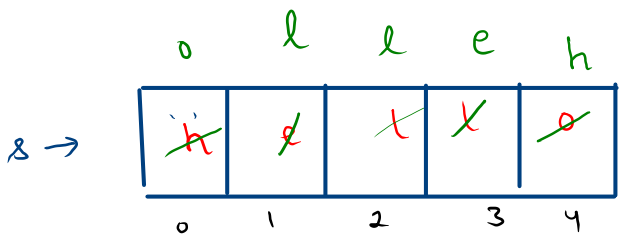
Input: s = ["h","e","l","l","o"]
Output: ["o","l","l","e","h"]

given

```
public void reverseString(char[] s) {

}
```

1. no need to mange i/p
2. no need to print o/p.



while (i < j)

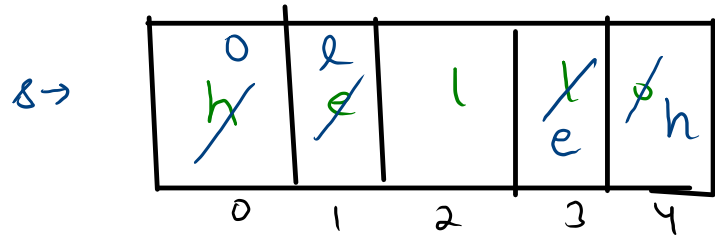
(i ≤ j)

i

j

```java
class Solution {
    public void reverseString(char[] s) {
        int i = 0;
        int j = s.length-1;

        while(i < j){
            //swap and manage i and j
            char tmp = s[i];
            s[i] = s[j];
            s[j] = tmp;
            i++;
            j--;
        }
    }
}
```



0<4 ✓

2<2 ✗

1<3 ✓ tmp=e

tmp = h[j]

s[i] = s[h]

s[j] =

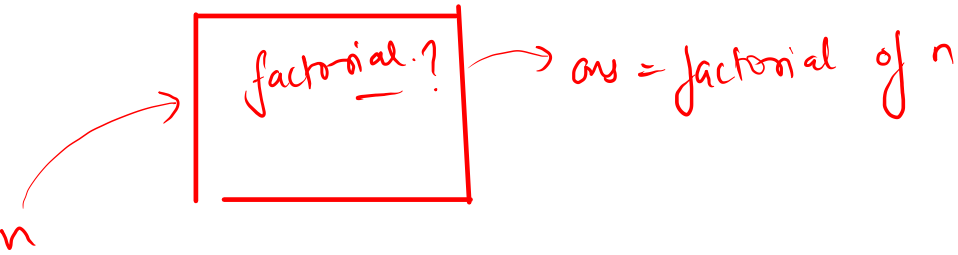
Pg 16 Q2

$${}^{n}C_{r} = \frac{n!}{r!\,(n-r)!}$$

$${}^{5}C_{3} = \frac{5!}{3!\,2!}$$

$$= \frac{5 \times 4 \times 3!}{3! \times 2}$$

$$= 10$$



```java
import java.io.*;
import java.util.*;

public class Solution {
    public static int factorial(int n){
        int fact = 1;
        for(int i = 1; i <= n; i++){
            fact *= i;
        }
        return fact;
    }

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int r = scn.nextInt();

        int nFact = factorial(n);
        int rFact = factorial(r);
        int nmrFact = factorial(n-r);

        int ans = (nFact)/(nmrFact * rFact);
        System.out.println(ans);
    }
}
```

$^5C_3 = 10$?

n = 5
r = 3.

fact = 1 1 2 6 24 120

i = 1 2 3 4 5 6

1 ≤ 5
2 ≤ 5
3 ≤ 5 ✓
4 ≤ 5 ✓
5 ≤ 5
6 ≤ 5 ✗

120 / (6 × 2)

Time Complexity (TC)

TC $\neq$ time to run your program

TC: any maths function to represent relationship between number of user's i/p & operations.

(5) i/p ⟶ operation.

Time Complexity Why?

(A) solution

TC

SC

(B) solution

TC

SC

3 Notations.

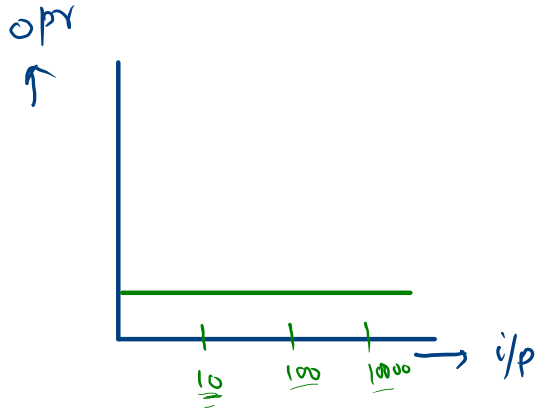
↳ represent

ω	omega	→	best case
θ	theta	→	avg. case
O ***	Oh	→	worst case.

O(1)

- maths opr. (+ - * / %)
- declaring a variable
- i/p
- print

→ constant TC



$f(x) = 2x^2 + x + 5$

degree = 2.

$Tc = n^3 + 2n + 6$

n^3

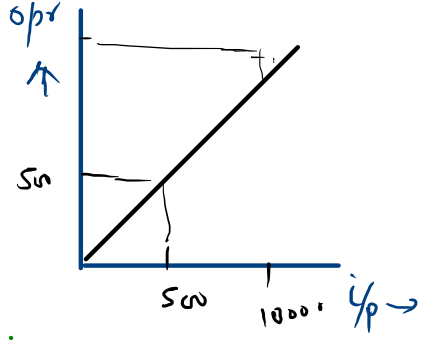
O(n) → linear time complexity.

i/p

n = 500 --- 500 opr

n = 1000 --- 1000 opr

n = 10000 --- 10000 opr.




```java
public static void function(int [] A){
    for(int i = 0; i < A.length; i++){
        System.out.println(A[i]);
    }
}
```

```java
10      public static void function(int n){
11          for(int i = 0; i < n; i++){
12              System.out.println(i);
13          }
14      }
15
```

Que.

```java
public static void function(int n){
    for(int i = 0; i < 10; i++){
        System.out.println(i);
    }
}
```

$O(10) \rightarrow O(1)$

linear

```
10      public static void function(int n){
11          for(int i = 0; i < n; i++){
12              int x = scn.nextInt();
13              System.out.println(x);
14          }
15      }
```

O(1) → int x = scn.nextInt();

O(1) → System.out.println(x);

O(1) } O(1)

$k_1 + k_2 = k$

1 itr ⟶ O(1)

n itr ⟶ O(1) × n = O(n)

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int x = scn.nextInt();
        int n = scn.nextInt();

        for(int i = x; i <= n; i++){
            System.out.println(i);
        }

    }
```

$-2^{31} < x \leq 2^{31} - 1$

$-2^{31} < n \leq 2^{31} - 1$

Worst

$x = -2^{31}$

$n = 2^{31} - 1$

$x = 0$

$n = 100$

2^n

$x = 100$

$n = 900$

$n = 2^{31} - 1$

$x = -2^{31}$

$2n$

$O(2n)$

$O(n)$

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        int n = scn.nextInt();

        for(int i = 1; i <= 10; i++){
            System.out.println(n+"x" + i + "=" + (n*i));
        }
    }
}
```

n = 5

n = 50

n = 50,000

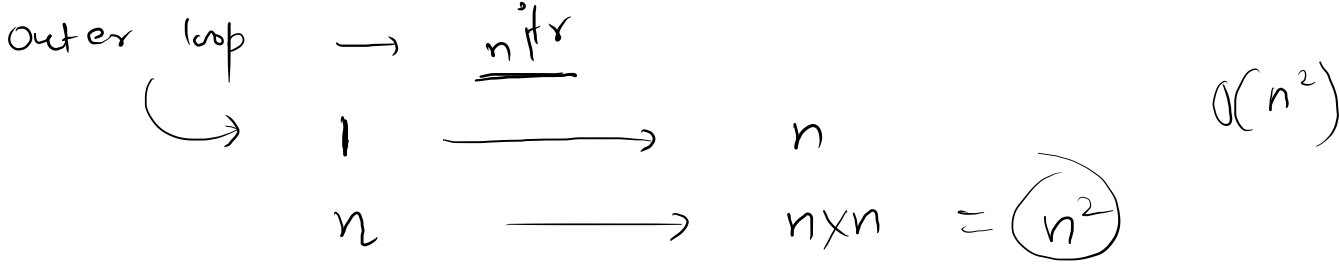
10 times. → constant

O(1).

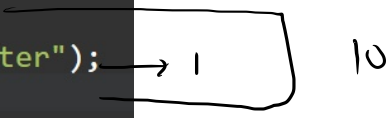
2 loops ?

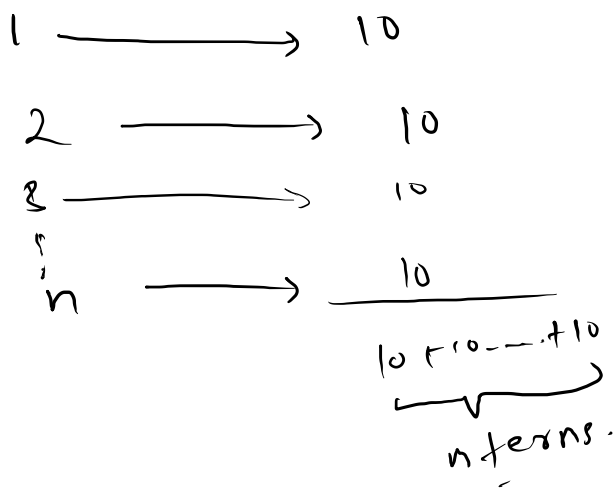
n^2

```
16      public static void function(int n){
17          for(int i = 0; i < n; i++){
18              for(int j = 0; j < n; j++){
19                  System.out.println("Geekster");
20              }
21          }
22      }
23
```



```java
public static void function(int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < 10; j++){
            System.out.println("Geekster");
        }
    }
}
```





$1 \leq n \leq 10000$

```java
public static void function(int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < 10; j++){
            System.out.println("Geekster");
        }
    }
}
```

eg 1. $n = 100 \rightarrow$ opr 1000

$n = 10000 \rightarrow 100000$

```java
16  public static void function(int n){
17      for(int i = 0; i < 10; i++){
18          for(int j = 0; j < 10; j++){
19              System.out.println("Geekster");
20          }
21      }
22  }
```

$n = 100 \rightarrow 100$

$n = 10000 \rightarrow 100$

$O(n^2)$

$n = 100$

Ex

```
public static void function(int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            System.out.println("Geekster");
        }
    }
}
```

Que.

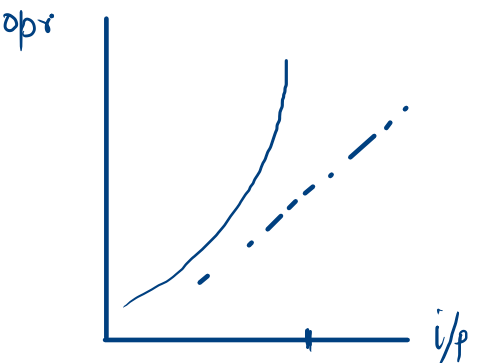
```java
public static void function(int n){
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            System.out.println("Geekster");
        }
    }
    for(int i = 0; i < n; i++){
        int x = scn.nextInt();
        System.out.println(x);
    }
}
```

n^2

n

$n^2 + n$

$O(n^2)$



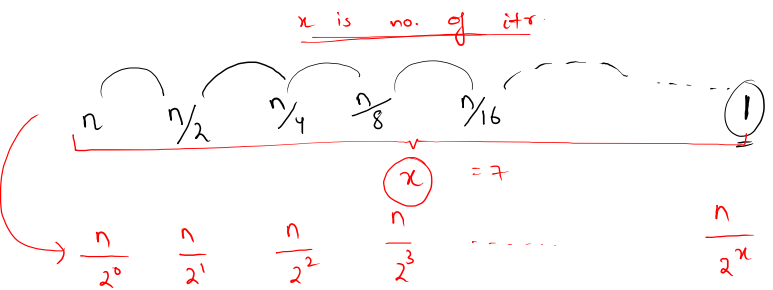
$O(\log n) \rightarrow O(\log_2 n)$

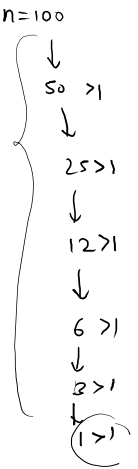
$\log n \rightarrow \log_2 n$

```java
public static void function(int n){
    while(n > 1){
        System.out.println("Geekster");
        n /=2;
    }
}
```

binary search.

$O(1)$	...	$O(\log n)$	...	$O(n)$	...	$O(n^2)$
constant 1		7		100		10000





$$1 = \frac{n}{2^x}$$

$$2^x = n$$

$$\log_2 2^x = \log_2 n$$

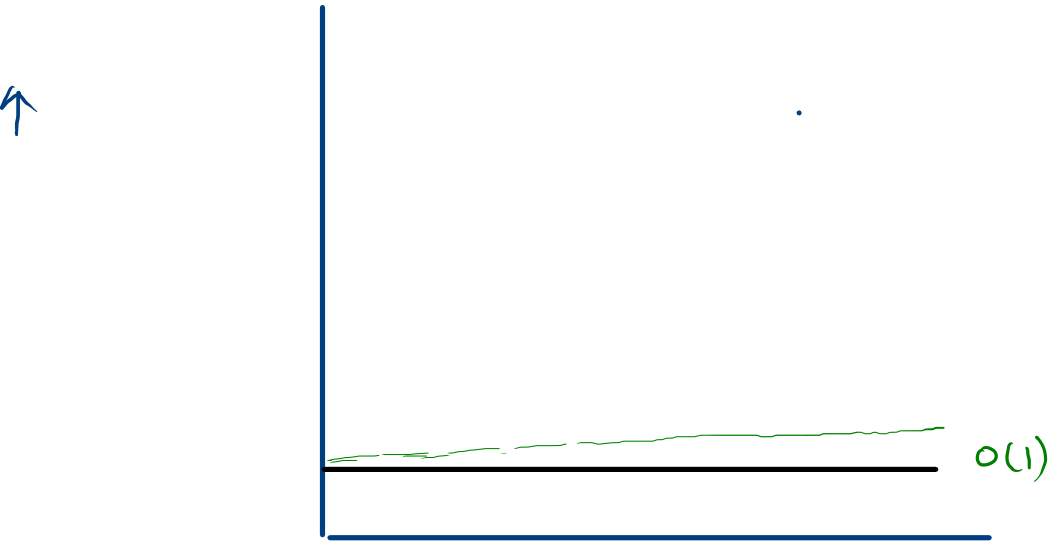
$$x \log_2 2^1 = \log_2 n$$

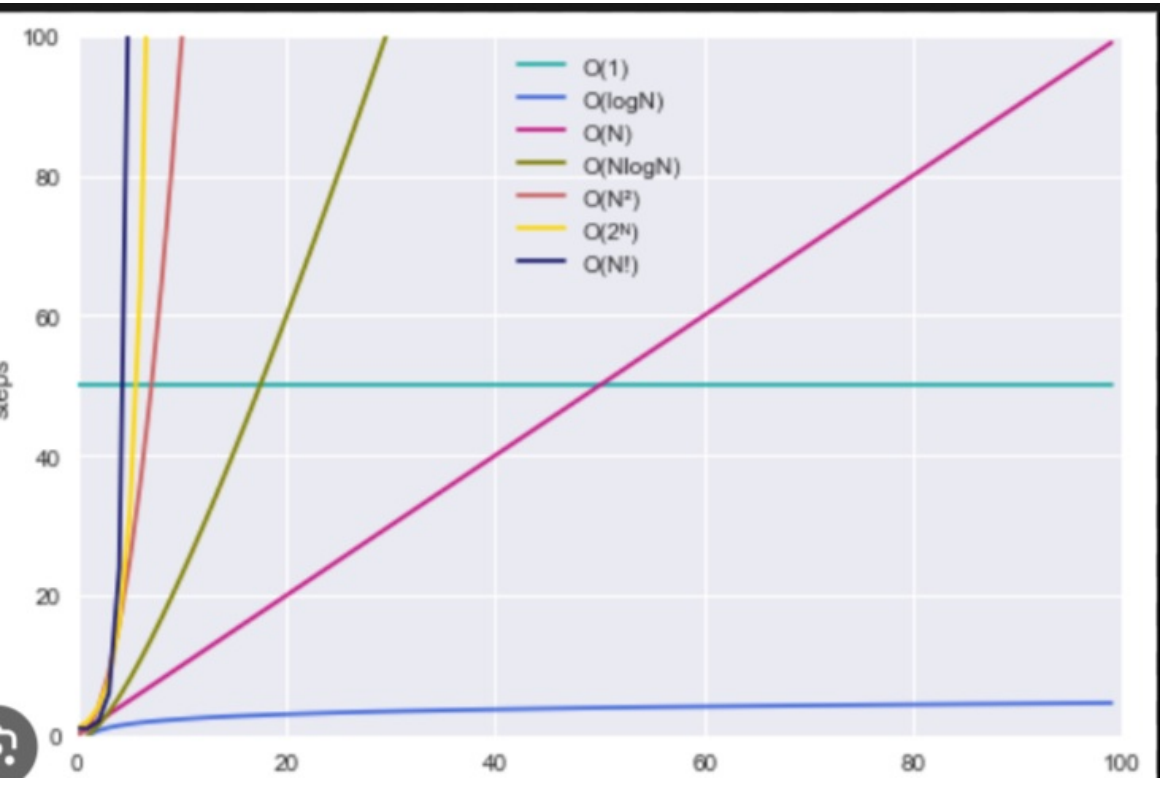
$$x = \log_2 n$$

```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int count = 0;
        while(n > 0){
            n /= 2;
            count++;
        }
        System.out.println(count);
    }
}
```


O(1)



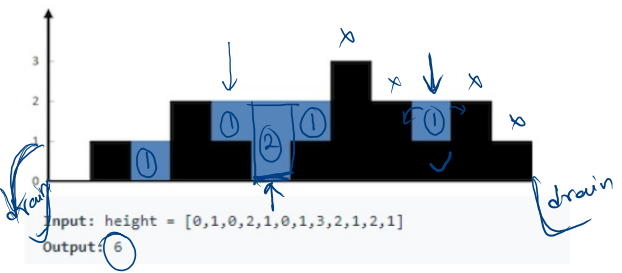
$O(2^n) \rightarrow$ recursion

$O(n \log n) \rightarrow$ sorting

- $O(n)$
- $O(n^2)$
- $O(1)$
- $O(\log n)$

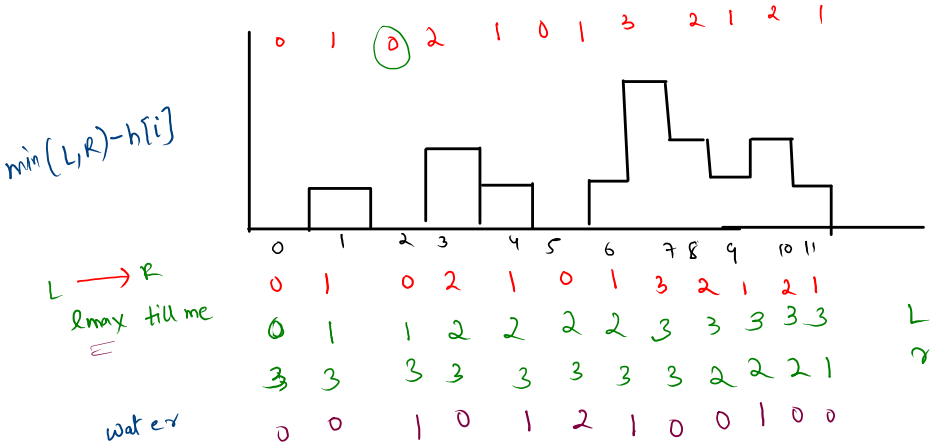
Store Maximum

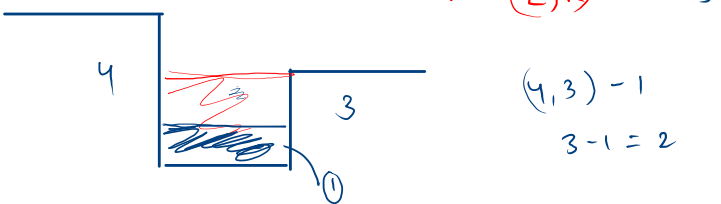
Given **n** non-negative integers representing an elevation map where the width of each bar is **1**, compute how much **maximum water** it can trap after raining.




```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

store water above each container

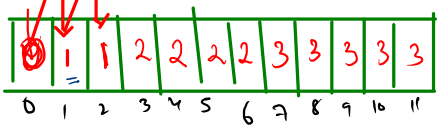




```
12
0 1 0 2 1 0 1 3 2 1 2 1
```

l to r

max till me



prefix.

or max.

$lmax[i] = max(lmax[i-1], A[i])$

max 7, 2

5 6 7 4 3 2

7

7 ?

5 6 7 4 8

7

7,8

8