Day 11 - Time & Space Complexity

Friday, 29 October 2021 2:37 PM

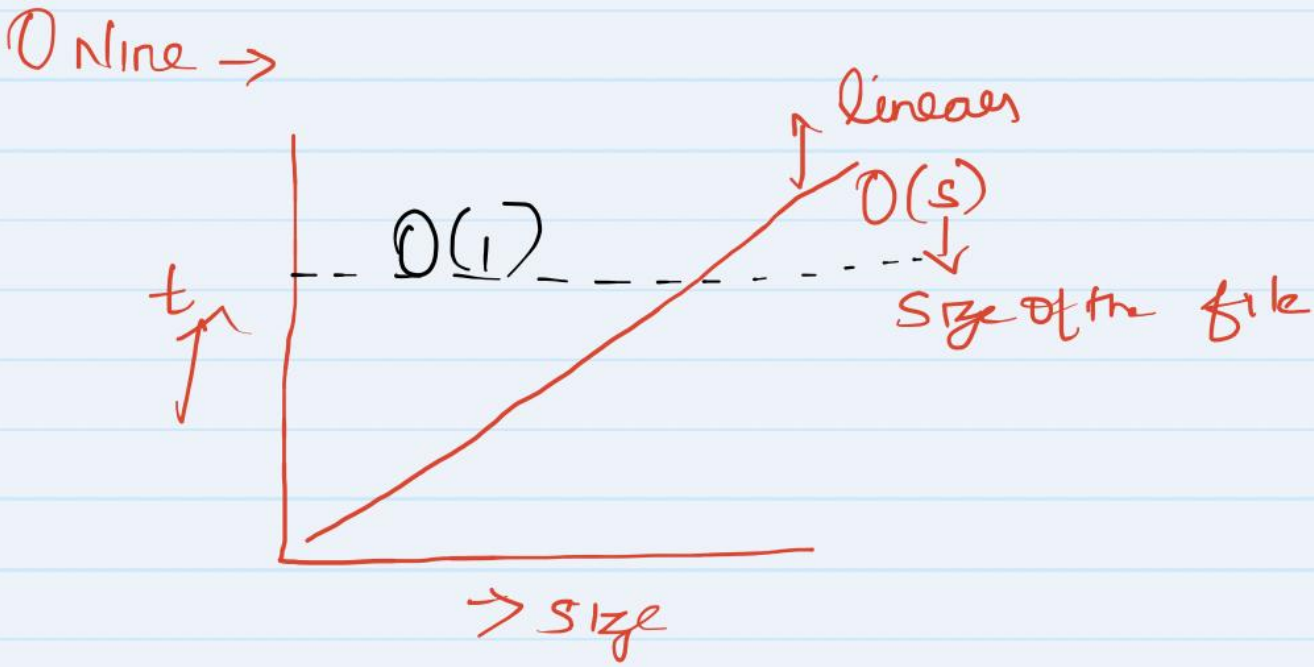
→ Online → time depends size of the file → 5MB → 5TB → days

→ Airplane → doesn't depend on file size (5MB (or) 5TB)

100 hours

↓

O(1) →



Best, Worst Average Case

Worst Case: Maximum number of operations

[10, 20, 30, 40] 4 compare

✓ target - 50 (not Present)

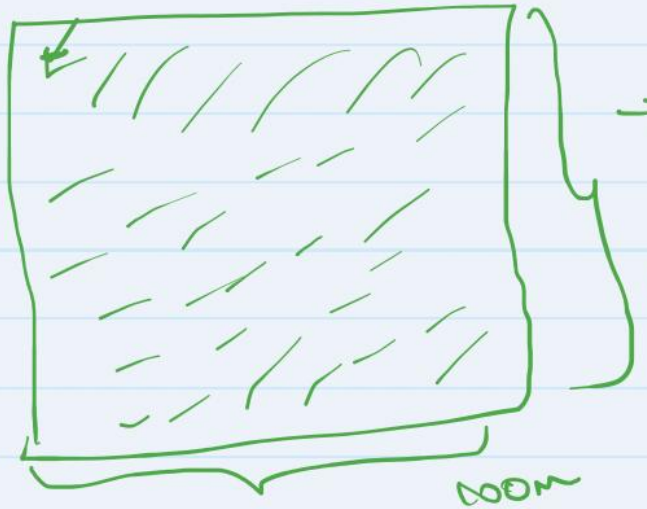
✓ O(n)

✓ Best case: Minimum no. of operations

✓ target - 10 → 1 compare

O(1)

✓ Avg case: taking all Possible inputs



→ O(a) ⇒ <-> O(s²)

↓ ↓

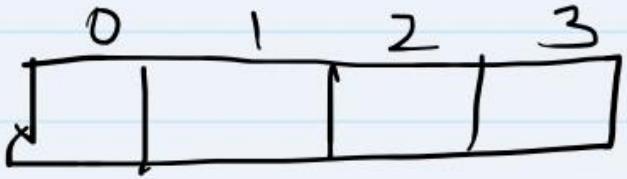
area of the land length of one side

✓ O(n) → what is ?

O(a) <-> O(s²)

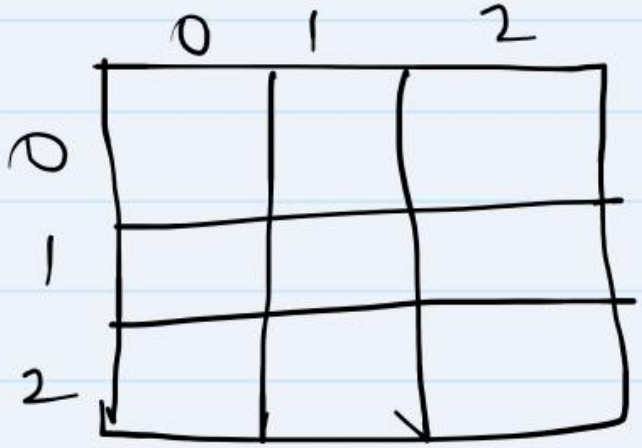
Space Complexity

→ Create an array of size n → 3 (or) 4



→ O(n)

→



n = 3 Square matrix.

→ $O(n^2)$ ✓

Not a square matrix

row = 4 col = 3

→ O(row * col) ✓

```
1  int sum(int n) { /* Ex 1.*/
2    if (n <= 0) {
3      return 0;
4    }
5    return n + sum(n-1);
6  }
```

base cond.

4 + sum(3)

3 + sum(2)

n = 4 → 10

① sum(4) → return

sum(3) 2

sum(2) 1

sum(1) 0

sum(0)

stack

stack space →

sum(4) → 4 + sum(3) (6) 4+6 → 10

sum(3) → 3 + sum(2) 3+3

sum(2) → 2 + sum(1) → 2+1

sum(1) → 1 + sum(0) 1+0

sum(0) → 0

stack

→ O(n)

TIME COMPLEXITY

① Drop the constants

Min and Max 1

```
1  int min = Integer.MAX_VALUE;
2  int max = Integer.MIN_VALUE;
3  for (int x : array) {
4    if (x < min) min = x;
5    if (x > max) max = x;
6  }
```

O(N)

Min and Max 2

```
1  int min = Integer.MAX_VALUE;
2  int max = Integer.MIN_VALUE;
3  for (int x : array) {
4    if (x < min) min = x;
5  }
6  for (int x : array) {
7    if (x > max) max = x;
8  }
```

O(N) + O(N)

O(2N) → ignore the constants

O(2N) → O(N)

O(100N) → O(N)

Size → N

② Different steps get added

Add the Runtimes: O(A + B)

```
1  for (int a : arrA) {
2    print(a);
3  }
4
5  for (int b : arrB) {
6    print(b);
7  }
```

arrA → O(A)

arrB → O(B)

O(A+B)

Multiply the Runtimes: O(A*B)

```
1  for (int a : arrA) {
2    for (int b : arrB) {
3      print(a + "," + b);
4    }
5  }
```

for every a → iterating entire B

O(A * B)

③ Different input → Different variables

```
int intersection(int[] arr1, int [] arr2) {
        int count = 0;

        for(int num1 : arrr1) {
                for(int num2 : arr2) {
                        If(num1 == num2) {
                                count = count + 1;
                        }
                }
        }
        return count;
}
```

arr1 → O(n x n) → O(n²)

A/N, B/M

O(A·B)

④ Drop the non-dominant terms

```
Int max = Integer.MIN_VALUE;
For(int num : nums) {
        Max = Math.max(max, num);
}

For(int num1: nums) {
        For(int num2 : nums) {
                Print(num1 +" " + num2)
        }
}
```

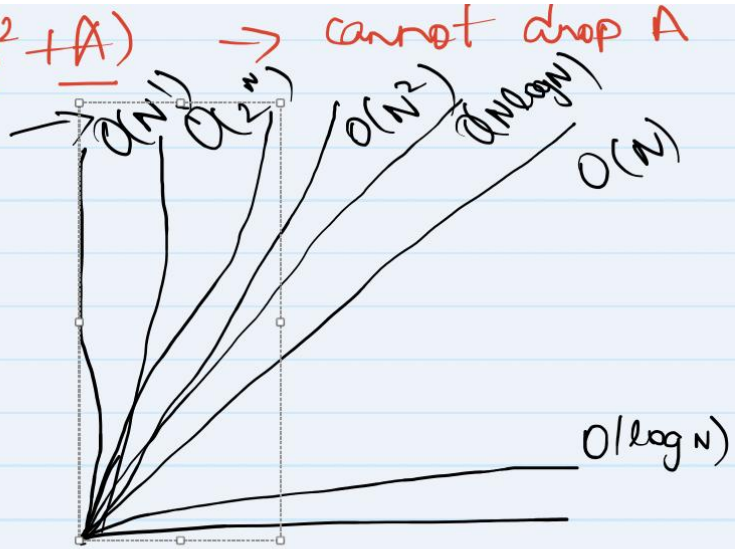
O(N) (nums size) + O(N²) → O(N² + N) → can be dropped

→ O(N²)

eg: O(N + log N) → O(N)

eg O(B² + A) → cannot drop A

eg $O(B^2 + A)$ → cannot drop A



log N

```
                                  0  1  2  3  4  5  6  7  8
search 9 within {1, 5, 8, 9, 11, 13, 15, 19, 21}        N
    compare 9 to 11 -> smaller.
    search 9 within {1, 5, 8, 9, 11}                     N/2
        compare 9 to 8 -> bigger
        search 9 within {9, 11}                          N/4 → (N/2)/2
            compare 9 to 9
            return
```

N → half

N/2

(N/2)/2 ⇒ N/4

(N/4)/2

N = 16

N = 8

N = 4

N = 2

N = 1

→ O(log N)

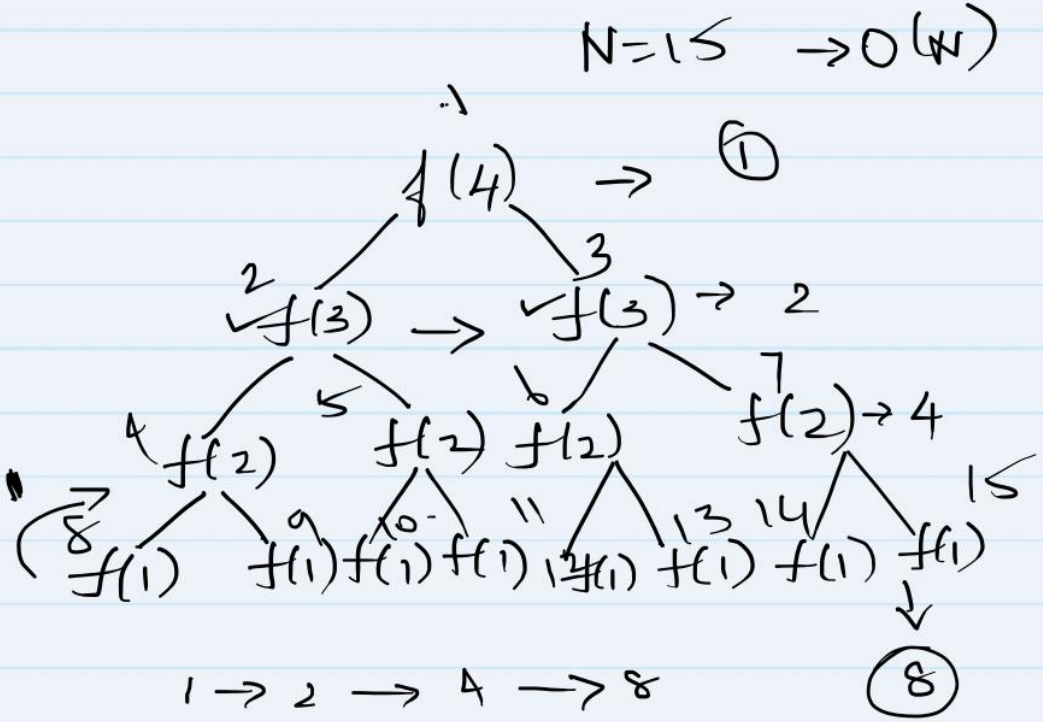
RECURSIVE RUNTIME:

```
int f(int n) {
    If(n <= 1) {
        Return 1;
    }
    Return f(n - 1) + f(n - 1);
}
```

f(3) + f(3)

f(2) + f(2)

n = 4



1 → 2 → 4 → 8

1 → 2 → 4 → 8

$2^0 \to 2^1 \to 2^2 \to 2^3$

$2^0 + 2^1 + 2^2 + 2^3 \ldots + 2^N$

$\Rightarrow 2^{N+1}$

$\Rightarrow (2^N)$

→ O(branches^depth)

(2^N)

Example 1

What is the runtime of the below code?

```
1   void foo(int[] array) {
2      int sum = 0;
3      int product = 1;
4      for (int i = 0; i < array.length; i++) {
5         sum += array[i];
6      }
7      for (int i = 0; i < array.length; i++) {
8         product *= array[i];
9      }
10     System.out.println(sum + ", " + product);
11  }
```

O(N) + O(N)

O(2N) → O(N)

Example 2

What is the runtime of the below code?

```
1   void printPairs(int[] array) {
2      for (int i = 0; i < array.length; i++) {
3         for (int j = 0; j < array.length; j++) {
4            System.out.println(array[i] + "," + array[j]);
5         }
6      }
7   }
```

$O(N^2)$

Example 3

This is very similar code to the above example, but now the inner for loop starts at `i + 1`.

```
1  void printUnorderedPairs(int[] array) {
2    for (int i = 0; i < array.length; i++) {
3      for (int j = i + 1; j < array.length; j++) {
4        System.out.println(array[i] + "," + array[j]);
5      }
6    }
7  }
```

N = 4

(0,1) (0,2) (0,3) NXN

X (1,2) (1,3)

X X (2,3)

$N^2 \rightarrow N^2/2 \rightarrow N^2$

```
1  void printUnorderedPairs(int[] arrayA, int[] arrayB) {
2    for (int i = 0; i < arrayA.length; i++) {
3      for (int j = 0; j < arrayB.length; j++) {
4        if (arrayA[i] < arrayB[j]) {
5          System.out.println(arrayA[i] + "," + arrayB[j]);
6        }
7      }
8    }
9  }
```

O(1)

O(A×B)

Example 5

What about this strange bit of code?

```
1  void printUnorderedPairs(int[] arrayA, int[] arrayB) {
2    for (int i = 0; i < arrayA.length; i++) {
3      for (int j = 0; j < arrayB.length; j++) {
4        for (int k = 0; k < 100000; k++) {
5          System.out.println(arrayA[i] + "," + arrayB[j]);
6        }
7      }
8    }
9  }
```

→ Constant

O(1)

O(A×B)

["abc" "bca" → sort Array of string
"etg" "gte"
"acd" → cda
"blue"] → belu

→ O(N log N)

a log a

string length → s

array length → a

→ [abc, belu, cda, etg]

① sort each string → O(s log s) → maximum length

② sort every string in array → O(a · s log s) step 1

③ sort entire array → O(a log a) → compare every string

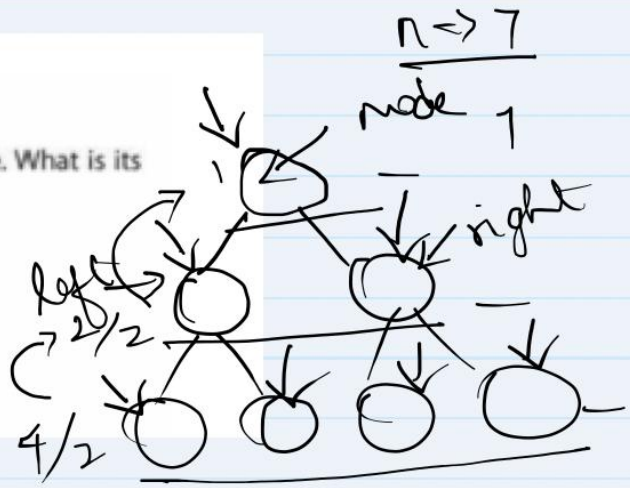
→ O(s · x a log a) → step 2

O(a · s (log s + log a))

Example 9

The following simple code sums the values of all the nodes in a balanced binary search tree. What is its runtime?

```
1  int sum(Node node) {
2    if (node == null) {
3      return 0;
4    }
5    return sum(node.left) + node.value + sum(node.right);
6  }
```



$\log N$ * N

$O(\text{branch}^{\text{depth}})$

$O(2^{\log N}) \Rightarrow P$

W -> number of nodes

$\log_2 2^{\log N} = \log_2 P$

$\log_2 N = \log_2 P$

$N = P$

$N = 2^{\log N}$

$\rightarrow O(N)$