Assignment 1: Algorithmic Coding

Carlos Andrés Osorio

May 18, 2023

What is the time and space complexity of the following functions?

1.
```python
import numpy as np


def func1(n, m):
  a = 0
  b = 0
  for i in range(n):
    a += np.random.randn()
  for i in range(m):
    b += np.random.randn()
  return a + b
```

Time complexity: In this algorithm, we are performing two separated loops: one of length n and the other one of length m. In each iteration of the two loops, we are generating a random number between $(0, 1)$ and sum it to the variables a, b. The generation of each of these random numbers and summing it to a and b has time complexity $O(1)$. So, finally, time complexity of the entire algorithm is the sum of the time complexity of both loops: $O(n + m)$.

Space complexity: The auxiliary space complexity is used by storing the variables a and b. The algorithm does not creates any data structure that grows with the input size, so the auxiliary space complexity is $O(1)$.

2.
```python
def func2(n):
  a = 0
  for i in range(n):
    for j in range(n, i, -1):
      a = a + i + j
  return a
```

Time complexity: Each iteration of the outer loop make $(n + 1) - i$ different $O(1)$ operations following the inner loop. It means that the total number of operations are

$$\sum_{i=1}^{n}[(n + 1) - i] = \frac{1}{2}n(n + 1)$$

It means that the algorithm has $O(n^2)$ time complexity.

Space complexity: The algorithm only creates the variable a which is a number and does not grows with the input n, so the auxiliary space complexity is $O(1)$.

3.
```python
def func3(n):
  a = 0
  i = n
  while i > 0:
    a += i
    i //=2
    print(i)
  return a
```

Time complexity: Let k be the number of iterations the program makes. Notice that the program stops for the first k when $\frac{n}{2^{k-1}} < 1$, and it means $k > \log_2(n) + 1$. So, the number of iterations the program makes is at least $\log_2(n) + 1$, so the time complexity is $O(\log(n))$.

Space complexity: We are only storing two variables: a and i. It means the auxiliary space complexity is $O(1)$.

4.
```python
def func4(n):
  a = 0
  j = 2
  for i in range(n // 2, n, 1):
    while j <= n:
      a = a + n // 2
      j *= 2
  return a
```

Time complexity: The outer loop makes $\frac{n}{2}$ iterations. Let's analyze the while statement: if k is the number of iterations that the while loop performs, then we can write j as $j = 2^k$. It means that the program stops with the least k such that $2^k > n$, therefore the number of iterations of the program is at least $\log_2(n)$. Since this number of iterations is carry out for every iteration in the outer loop, the total operations is roughly

$$\frac{n}{2}\log_2(n).$$

Since the others operation are $O(1)$, the time complexity is $O(n\log(n))$.

Space complexity: As before, the algorithm does not creates objects whose length depend of the input, so the auxiliary space complexity is $O(1)$.

5.
```python
from typing import List


Vector = List[int]
def func5(arr: Vector):
    for i in range(1, len(arr)):
      key = arr[i]
      j = i-1
      while j >=0 and key < arr[j] :
        arr[j+1] = arr[j]
        j -= 1
      arr[j+1] = key
    return arr
```

Time complexity: The outer loop iterates over the array from the second element to the last element. Then, it creates a key with the current element of the iteration and performs a comparison. If the key is less than the previous element of the array, they are interchanged. This process continues until

the array is completely sorted in ascending order. The worst-case scenario occurs when the array is initially sorted in descending order, requiring the while loop to iterate over the entire array. In the worst-case scenario, the function performs $(n-1) \times n$ operations, resulting in a time complexity of $O(n^2)$.

Space complexity: Since we are not creating a new array or another kind of structure with different space complexity, the final auxiliary space complexity of this algorithm is $O(1)$.