Exercise #00 — CS 335

Jared Dyreson
California State University, Fullerton

August 26, 2021

Contents

1 Examples

1.1 Arithmetic

Problem:

```
def arithmetic(lst):
    x = lst[0]
    y = lst[1]
    lst[2] = x + y
    return lst[0] + lst[1]
```

Analysis: Each line only contains one 1 step, therefore it linear. Thus having a classification of $O(n)$.

1.2 Nested Function Calls

```
def setup():
    forward = make_list(20) # n + 3 : dependent of defined size and contents of
                                          function called below
    backward = make_list(30) # n + 3 ^ see above
    return forward, backward # 1
    # 2(n + 3) + 1 => 2n + 6 + 1 => 2n + 7

def make_list(n):
    L = [] # 1
    for i in range(n): # n
        L.append(0) # 1
    return L # 1
    # 1 + n + 1  + 1 => n + 3
```

Analysis: Therefore, the current runtime is $O(2n + 7)$. However, since we are classifying algorithms in Big Oh, we ignore leading coefficients and only focus on the leading term. Thus, the true classification of this algorithm is of $O(n)$, as it is directly proportional to the input of the problem set.

1.3 Loops

$$T(n) = \sum_{x \in X} t_x$$

```
for x in range(100):
    print("hello") # please assume printing only takes one instruction/atomic
    print("world")
    print("goodbye")
```

Analysis: Since this loop only contains atomics, we can add up the amount of them and use it as our coefficient. Thus becoming: $O(3n) \implies O(n)$

1.4 Loops in Sequence

```
def func():
    for x in range(100):
        # contains three atomics
        print("hello")
        print("hello")
        print("hello")
    for x in range(1000):
        # contains two atomics
        print("world")
        print("hello")
```

Analysis: Since both of these loops only contain atomics, we can add up the amount of them and use it as our coefficient for each. Both loops are independent of one another and do not contribute to the overall runtime classification. Thus becoming:
$(O(3n) + O(2n)) \implies O(5n) \implies O(n)$

1.5 Nested Loops

```
for x in range(10):
    # three atomics
    for y in range(10):
        # two atomics
```

Analysis: When attempting to analyze the runtime of nested loops, we need to count how many loop calls there are. This will give us the degree our polynomial will be. These instructions compound on one another, as the parent must execute along with all it's children. In this case, our deepest for loop call is of $O(2n)$ and our parent is $O(3n)$. We can determine the runtime by saying that:

$$O(2n) \times O(3n) \implies O(6n^2) \implies O(n^2)$$

2 General Notes

- If a math function gives a large resultant from a small input size, please refrain from using this algorithm