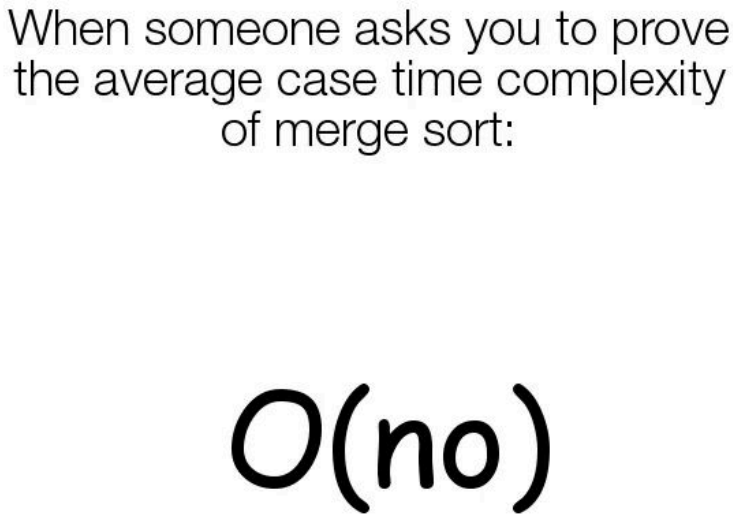
5.1 - Week 5 - Applied - Theory

Objectives of this Applied Session

- To understand Big O notation
- To be able to find the best and worst-case time complexities for simple algorithms
- To learn how to use Exceptions

Complexity - Simplified!



Alright!
We've heard a lot about complexity in the previous few weeks, and possibly in other units too. Now, we will delve deeper into what exactly complexity is and how we can use it to make better programs!

Now, we know that complexity can be a daunting concept and that's why we should break it down into simpler concepts, which will help us understand it.

Your tutor will now go through the following concepts with you:

1. What is the need to calculate complexity

2. What terms do we consider while calculating complexity of an algorithm?

3. How do we consider complexity while scaling an algorithm

4. What are the different types of complexity

5. How to quickly calculate the complexity of an algorithm?

But first! Lets have a simple quiz to get your brains chugging along :)

Complexity Basics - Quiz

Try to answer these questions to the best of your knowledge, don't worry too much about knowing/not knowing the concepts of complexity. These should (hopefully) help you understand how to approach complexity.

Imagine a classroom of 10 students in which you gave your pen to one person. You have to find that pen without knowing to whom you gave it.

Now, answer the following questions

Question 1 *Submitted Aug 23rd 2022 at 8:16:44 am*

You decide to go and ask the first person in the class if he has the pen. Also, you ask this person about the other 9 people in the classroom if they have that pen and so on. In the end, every person in the room asked every other person in the classroom for the pen.
In Big O terms, how would you best describe this complexity?

- $O(1)$
- $O(n)$
- **$O(n^2)$**
- $O(10n)$

Question 2 *Submitted Aug 23rd 2022 at 8:16:56 am*

You decide to go and ask every person in the classroom yourself on a mission to find the pen alone.
In Big O terms, how would you best describe this complexity?

- $O(1)$
- **$O(n)$**

- $O(n^2)$
- $O(10n)$

Question 3 *Submitted Aug 23rd 2022 at 8:17:10 am*

Now you divide the class into two groups, then ask: “Is it on the left side, or the right side of the classroom?” Then you take that group and divide it into two and ask again, and so on. Repeat the process till you are left with one student who has your pen.

In Big O terms, how would you best describe this complexity?

- [x] $O(log(n))$
- $O(n)$
- $O(n^2)$
- $O(1)$

Count of instructions

Okay so hopefully you've got a bit more of an understanding of Complexity and how we can approach it.
Now, let's discuss the complexity of algorithms.

1. An $O(n)$ algorithm

2. An $O(n^2)$ algorithm

We will also discuss how to calculate the number of instructions run in these algorithms and how they differ from each other.

Once we've done that, we can then use the knowledge to answer the following questions:

Question 1 *Submitted Aug 23rd 2022 at 8:28:34 am*

If an algorithm has complexity $O(n)$, with an input of size n, explain what this means with respect to the number of operations in the algorithm. Give an example of such an algorithm.

> Number of operations in the algorithm increases linearly and is directly proportion to the input size of n.

Question 2 *Submitted Aug 23rd 2022 at 8:34:47 am*

An algorithm that involves comparing list elements has an input of size n, cost of comparison k, and complexity $O(n^2 + kn)$. Explain what this means with respect to the number of operations in the algorithm.

> *O*(*n*^2) = Called twice for each input for size n
>
> *O*(*kn*) = Called in the multiple of k for each input for size n
>
> *O*(*n*^2+*kn*) = Number of operations in the algorithm increases by a multiple of n with the addition of twice of the input size n

Question 3 *Submitted Aug 23rd 2022 at 8:36:37 am*

Consider the following function:

```
def example_func(the_list: List[str]) -> int:

    int_list = []
```

```
    for i in range(len(the_list)):
        int_list.append(len(the_list[i]) % len(the_list))

    insertion_sort(int_list)

    index_to_find = int_list[len(int_list)-1]
    count = 0
    item_to_count = the_list[index_to_find]

    for i in range(len(the_list)):
        if the_list[i] == item_to_count:
            count += 1

    return count
```

This function is an example of the complexity discussed above. Explain what is going on in the function above and how can the complexity of this algorithm be confirmed.

No response

Comparing Algorithms

Okay! Hopefully, we are halfway through turning you into a complexity Wizard! We will now compare some runtimes for certain unknown algorithms and you'll find yourself able to identify complexities without even looking at the algorithms!

Consider the following runtimes for 5 algorithms:

input size	maximum running time (ms)				
N	Alpha	Bravo	Charlie	Delta	Echo
10	9999999	10	20	9876543	1415926535
20	99999	40	200	3010	8979323846
30	220	777777	2000	4771	2643383279
40	330	160	20000	6021	5028841971
50	439	245	200000	6990	6939937510
60	549	360	2000000	7782	5820974944
70	660	489	20000000	8451	5923078164
80	767	640	200000000	9031	628620899
90	880	808	2000000000	9542	8628034825
100	989	1000	20000000000	10000	3421170679

Please note that the point of this exercise is not to figure out exact time functions such as 11(N − 10), but for you to become familiar with some of the trademarks of time growth, such as:

- Linear - if as N increases by a constant, time goes up by at most some constant.
- Quadratic - if when N is doubled, time goes up by at most a factor of 4.
- Exponential - (on k) if when N increases by a constant, time increases by a factor of k.
- Sublinear - if when N increases by a constant, time increases by a decreasing amount.
- Constant - if when N increases by a constant, time does not seem to have a growth pattern and is always below a certain constant, independent of N.

And lets answer the following questions:

Question 1 *Submitted Aug 23rd 2022 at 8:57:16 am*

What is the complexity for Alpha?

O(n)

Question 2 *Submitted Aug 23rd 2022 at 8:57:39 am*

What is the complexity of Bravo?

O(n^2)

Question 3 *Submitted Aug 23rd 2022 at 8:58:35 am*

What is the complexity of Charlie?

O(10^n)

Question 4 *Submitted Aug 23rd 2022 at 8:59:10 am*

What is the complexity of Delta?

O(log(n))

Question 5 *Submitted Aug 23rd 2022 at 9:01:24 am*

What is the complexity of Echo?

O(1)

Question 6 *Submitted Aug 23rd 2022 at 9:02:43 am*

This data is suggestive of some likely Big-O expressions for the worst-case complexity of each algorithm, but it does not *prove* anything about them. Give an explanation as to why it does not.

No mathematical proof

Question 7 *Submitted Aug 23rd 2022 at 9:01:48 am*

Which algorithm is most scalable, according to big-O complexity?

- ○ Alpha
- ○ Bravo
- ○ Charlie
- ○ Delta
- ◉ Echo

Question 8 *Submitted Aug 23rd 2022 at 9:02:02 am*

Which algorithm would you choose for an input of size 40?

- [ ] Alpha
- [x] Bravo
- [ ] Charlie
- [ ] Delta
- [ ] Echo

Question 9 *Submitted Aug 23rd 2022 at 9:02:12 am*

Which algorithm would you choose for an input of size 10000? Have an educated guess rather than trying to calculate it.

- [ ] Alpha
- [ ] Bravo
- [ ] Charlie
- [x] Delta
- [ ] Echo

Question 10 *Submitted Aug 23rd 2022 at 9:02:21 am*

Which algorithm would you choose for an input of size 10^(10^10)? Again take an educated guess rather than calculating an answer.

- [ ] Alpha
- [ ] Bravo
- [ ] Charlie

- [ ] Delta
- [x] Echo

Different Views of Complexity

So far, we've explore algorithms that have a singular complexity. What about an algorithm that can be viewed as having multiple complexities?
How do we deal with a situation like that?
Why are variable SO important when defining complexities?
Lets take the following quiz to understand.

Consider the python program below:

```
def sum_of_digits(x:int) -> int:
    """ Computes the sum of the digits of integer x
    :pre: x is >= 0
    :raises ValueError: if x < 0
    :complexity: best and worst are O(log x)  or O(d),
    where x is the value of the number and d is the number of digits.
    """
    if x < 0:
        raise ValueError("number should not be negative")
    digit_sum = 0
    while x > 0:
        digit_sum += x % 10
        x = x // 10
    return digit_sum
```

And answer the questions below:

Question 1 *Submitted Aug 23rd 2022 at 9:23:56 am*

What is the function in the program written above doing?

Sum of the digits of a number

Question 2 *Submitted Aug 23rd 2022 at 9:36:01 am*

Write the line-by-line complexity of the function.

loop according to the number of zeros

log(1000) = loop 3 times

log(10000) = loop 4 times

log(100000) = loop 5 times

```
if x < 0: --> O(1)

digit_sum = 0 --> O(1)
while x > 0: --> O(1)
    digit_sum += x % 10 --> O(1)
    x = x // 10 --> O(1)
return digit_sum --> O(1)
```

Question 3 *Submitted Aug 23rd 2022 at 9:24:12 am*

If we consider n as the number of digits in x, how would you define the complexity of this function?

- [x] $O(n)$
- [ ] $O(1)$
- [ ] $O(n^2)$
- [ ] $O(log(n))$

Question 4 *Submitted Aug 23rd 2022 at 9:24:24 am*

How would you define this function's complexity in terms of x?

- [ ] $O(x)$
- [x] $O(log(x))$
- [ ] $O(1)$
- [ ] $O(x^2)$

Question 5 *Submitted Aug 23rd 2022 at 9:24:37 am*

What is the one true complexity of this function?

- [ ] $O(n)$

- [ ] $O(log(n))$
- [ ] Both are correct
- [x] None of them are correct

Question 6 *Submitted Aug 23rd 2022 at 9:24:58 am*

Now, use ALL your knowledge from above and answer - 'What is the complexity of this function?'

- [ ] $O(n)$ where n is the input variable
- [x] $O(log(n))$ where n is the input variable
- [ ] $O(log(n))$ where n is the number of digits in the input variable
- [x] $O(n)$ where n is the number of digits in the input variable

Congrats! Here's some more work :)

Okay! So now you're an ace at complexity analysis. For the fun of it, let's throw in a recap of exception handling, that you would have been familiarised with in a previous unit (hopefully). You will need this knowledge for the upcoming assignments A2 and A3.

Consider the following code:

```
a = [0, 1]
try:
    b = a[2]
except ValueError:
    print("no it didn't work!")
else:
    print('that worked!')
```

and answer the following questions:

Question 1 *Submitted Aug 23rd 2022 at 8:24:23 am*

What will be the output of this code?

- 1
- (The code won't execute)
- no it didn't work!
- that worked!

Question 2 *Submitted Aug 23rd 2022 at 8:25:16 am*

What is the reason for the behavior of the output while running the code?

There is no index 0 (it doesn't exist) in the list of a, maximum is 1 only.

Question 3 *Submitted Aug 23rd 2022 at 8:25:44 am*

The best way to fix this program is as below:

```
a = [0, 1]
try:
    b = a[2]
except:
    print("no it didn't work!")
else:
    print('that worked!')
```

- ○ True
- ◉ False

Question 4 *Submitted Aug 23rd 2022 at 8:26:26 am*

Select the best possible solution that will fix the issue in the code while being descriptive about the error to the user:

- ○
```
a = [0, 1]
try:
    b = a[2]
except IndexError:
    print("no it didn't work!")
else:
    print("That worked!")
```

- ○
```
a = [0, 1]
try:
    b = a[2]
except IndexError:
    print("no it didn't work!")
finally:
    print("That worked!")
```

- ○
```
a = [0, 1]
try:
    b = a[2]
except Exception:
    print("no it didn't work!")
else:
    print("That worked!")
```

```
a = [0, 1]
try:
    b = a[2]
except IndexError:
    print("Non-existing index selected")
else:
    print("That worked!")
```

Question 5 *Submitted Aug 23rd 2022 at 8:26:46 am*

Okay now consider the following modified code:

```
a = [0, 1]
a = "hello"
try:
    b = a[2]
except IndexError:
    print("Incorrect Index chosen")
except ValueError:
    print("Incorrect type of variable selected")
else:
    print("That worked!")
```

What will be the output of the code above?

- Incorrect Index chosen
- That worked!
- I
- Incorrect type of variable selected

Question 6 *Submitted Aug 23rd 2022 at 8:27:23 am*

Consider the following modification:

```
a = [0, 1]
x = 10
try:
    b = a[2]
    x[2]
except IndexError:
```

```
    print("Incorrect Index chosen")
except TypeError:
    print("Incorrect type of variable selected")
else:
    print("That worked!")
```

What will be the output of this code?

- Incorrect Index chosen
  Incorrect type of variable selected
- Incorrect type of variable selected
  Incorrect Index chosen
- **Incorrect Index chosen**
- Incorrect type of variable selected

HOORAY!

Phew! Alright, definitely one of the more challenging applied sessions so far! But it is incredibly important for you to understand the concept of complexity as you are going to be poised with decisions all through your life as a programmer/software engineer where you need to reduce the complexity of your algorithms and make them more efficient according to the requirements of the organization.

Please go through the practical exercises in the Applied Optional sessions to get an even better grip on the concepts by applying them through your own code.