5.0 - Week 5 - Workshop (MA)

Learning Objectives

- Understanding Big-O notation
- Understanding the implications of different run-time complexities.

Week 5 Padlet Discussion Board link: https://monashmalaysia.padlet.org/fermi/2022week5

Computational Complexity

In this part of the workshop we will study Computational Complexity.

Big-O notation and complexity classes

Question 1 *Submitted Aug 23rd 2022 at 8:08:31 am*

Order these complexities from higher to lower.

Exponential

Linear

Logarithmic

Constant

Question 2 *Submitted Aug 23rd 2022 at 8:08:36 am*

If

$$T(n) = n \cdot 2^n$$

then

$$T(n) = O\left(2^{2n}\right)$$

Provide a proof or a counter-example, as appropriate, to justify your answer.

- (●) True
- ( ) False

Question 3 *Submitted Aug 23rd 2022 at 8:08:47 am*

What is the complexity of the following algorithm?

```
def f(n):
    for i in range(n):
        print(i)
        for j in range(i):
            for k in range(n):
                print(k)
        for j in range(i):
            print(j)
```

- [ ] $O(n^2)$
- [ ] $O\left(n^2 \log\left(n\right)\right)$
- [ ] $O(n \log^2\left(n\right))$
- [x] $O\left(n^3\right)$

Counting arithmetic operations

Question *Submitted Aug 23rd 2022 at 8:09:07 am*

Bound the number of arithmetic operations (+, -, *, /) of this function:

```
def binomial_coefficient(n: int, k: int) -> int:
    """
    Computes and returns the binomial coefficient "n choose k"

    :pre: 0<=k<=n
    """
    assert 0 <= k <= n

    #we use the multiplicative formula
    result = 1
    for i in range (1, k+1):
        result *= (n+1-i)/i

    return int(result)
```

```
def binomial_coefficient(n: int, k: int) -> int:
    """
    Computes and returns the binomial coefficient "n choose k".

    :pre: 0<=k<=n
    """
    assert 0 <= k <= n

    #we use the multiplicative formula
    result = 1
    for i in range(1, k+1):
        result *= (n+1-i)/i

    return int(result)
```

Choose the tightest $O()$ bound, i.e. the smallest correct upper bound.

- $O(n)$
- **$O(k)$**
- $O(n \cdot k)$

- $O(n!)$
- $O(1)$
- None of the above

Not counting arithmetic operations

In this question, rather than doing it by hand, we want to programmatically count the number of arithmetic operations of the function

```
def binomial_coefficient(n: int, k: int) -> int:
    """
    Computes and returns the binomial coefficient "n choose k".

    :pre: 0<=k<=n
    """
    assert 0 <= k <= n

    #we use the symmetry (n,k) = (n, n-k) to reduce
    #the number of operations
    k = min(k, n-k)

    #we use the multiplicative formula
    result = 1
    for i in range(1, k+1):
        result *= (n+1-i)/i

    return int(result)
```

In order to do so, we introduce a variable `ops` in the code, which we will use to keep track the number of such operations directly in the code.

1. Finish the implementation provided in the scaffold so that the function returns the correct value of `ops`.
2. For a given n, what value of k requires the highest number of operations?
3. How does the number of operations behave as a function of n?

(Note that `min` uses no arithmetic operations. It can indeed be implemented as shown below)

```
def min(a, b):
    return a if a <= b else b
```

Analysing four examples

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

We are now considering the complexity of functions for which the variable `ops` has already been added (as we have just done), and the other instructions have been stripped away.

```
def example_1(n: int) -> int:
    ops = 50
    for i in range(1, 1+ n//2):
        ops += 50
    for i in range(1, 2*n):
        ops += 2
    return ops

def example_2(n:int) -> int:
    ops = 100
    if n > 1:
        ops += example_2(n//2)
    return ops

def example_3(n: int) -> int:
    ops = 10
    for i in range(1, n//2):
        for j in range(0, i):
            ops += 2
    return ops

def example_4(n:int) -> int:
    ops = 1
    if n > 1:
        ops+= example_4(n-25) + example_4(n-50)
    return ops
```

```
def example_1(n: int) -> int:
    ops = 50
    for i in range(1, 1+ n//2):
        ops += 50
    for i in range(1, 2*n):
        ops += 2
    return ops

def example_2(n:int) -> int:
    ops = 100
    if n > 1:
        ops += example_2(n//2)
    return ops

def example_3(n: int) -> int:
```

```
    ops = 10
    for i in range(1, n//2):
        for j in range(0, i):
            ops += 2
    return ops

def example_4(n:int) -> int:
    ops = 1
    if n > 1:
        ops+= example_4(n-25) + example_4(n-50)
    return ops
```

Order the functions by smaller to highest run-time complexity.

example_2

example_1

example_3

example_4

Not analysing four examples

We've done all the hard work! Now all that's left to do is to plot the behaviour of these examples and see how their run-time evolves as the input n grows.

1. Press **Run**!
2. Suppose that all four algorithms solve the same problem. What is the best algorithm for $n \leq 200$?
3. Suppose that all four algorithms solve the same problem. What is the worst algorithm for large n, i.e. which algorithm has the highest complexity?

Work Work Work

Analyse the worst-case time complexity of the algorithms below and provide a $O()$ bound.

Question 1

What is the complexity of the function work?

```
def work(n:int):
    print(n)

work(9876)
```

In order to analyse the complexity of the function above, we will suppose it is equivalent to the one below:

```
import math

def work(n:int):
    size = 0
    output = [0]*(1+int(math.log10(n)))
    while n >= 1:
        output[size] = n%10
        size += 1
        n = n //10
    for i in range(size-1, -1, -1) :
        print(output[i], end='')
    print()
work(9876)
```

We suppose that we can compute the log10 of an integer in logarithmic time, and that other arithmetic operations take constant time.

No response

Question 2

What is the complexity of the function workwork?

```
def work(n:int):
    print(n)

def workwork(n:int):
    for i in range(1, n+1):
        work(i)

workwork(20)
```

No response

Question 3

What is the complexity of the function workworkwork?

```
def work(n:int):
    print(n)

def workwork(n:int):
    for i in range(1, n+1):
        work(i)

def workworkwork(n:int):
    i = 0
    while n > 0:
        print("worwork({}):".format(n))
        workwork(n)
        n = n//2

workworkwork(20)
```

No response

Feedback Form

Weekly Workshop Feedback Form

Question 1

I am enrolled in:

- □□ Australia
- □□ Malaysia

Question 2

What needs improvement?

No response

Question 3

What worked best?

No response

Question 4

How engaged were you by the workshop?

- □□□ Very engaged
- □□□ Engaged
- □😐□ Not impressed
- 😕😴□ Lost