Module 1: Algorithm Structures

1 With the pseudo-code below, determine the final value of `sum` when `n=23`.

Algorithm 1

```
procedure
   sum = 0
   for (i=2, i<n, i=i+3) do
      if i mod 2 == 0
        sum = sum + i
```

Solution: `sum = 44`

With `i` starting at 2 and incrementing by 3 every iteration, `i` will iterate through the values 2, 5, 8, 11, 14, 17, and 20. The `if` statement means `sum` will only be updated when `i` is even, adding 2, 8, 14, and 20 for a total of 44

2 Use the pseudo-code below to answer the following question.

(a) Without working through the pseudo-code, how many times with `val` be updated in Line 7?

Solution: 24

The loop in line 5 will iterate 4 times. The loop in line 6 will iterate 6 times. Since they are nested, we multiply and `val` will be updated $4 \cdot 6 = 24$ times.

(b) What is the final value of `val`.

Solution: `val = 210`

The algorithm loops through all possible combinations of x and y, multiplies them, and adds them to `val`. In the end, we get

$$1\cdot1+1\cdot2+1\cdot3+1\cdot4+1\cdot5+1\cdot6+$$
$$2\cdot1+2\cdot2+2\cdot3+2\cdot4+2\cdot5+2\cdot6+$$
$$3\cdot1+3\cdot2+3\cdot3+3\cdot4+3\cdot5+3\cdot6+$$
$$4\cdot1+4\cdot2+4\cdot3+4\cdot4+4\cdot5+4\cdot6 \;=210$$

Algorithm 2

```
1: procedure
2:    dFour = [1, 2, 3, 4]
3:    dSix = [1, 2, 3, 4, 5, 6]
4:    val = 0
5:    for (x in dFour) do
6:       for (y in dSix) do
7:          val = val + x·y
```

3 Using the pseudo-code below, what is the ending value for `count` when `n = 100`? When `n = 10,000`? When `n =` 10^{42}?

Algorithm 3

```
procedure
    count=0
    while (n>1) do
        count++
        n = n/10
```

Solution:

When `n = 100`, `count` will end at 2.

When `n = 10,000`, `count` will end at 4.

When `n =` 10^{42}, `count` will end at 42.

4 This question assumes familiarity with modular arithmetic discussed in Unit 2. In the pseudo-code below, Line 7 performs string addition (concatenation), for example:
`"a" + "b" = "ab"`. Even if the addends look like numbers, it will still do string addition:
`"2" + "3" = "23"`. Answer the questions below.

a. What is the output of the algorithm when $x = 99$ and $b = 2$?

Solution: Context for writing a number in base b is presented in Module 6 of Unit 2.

Expand(99, 2) $=$ `"1100011"`

b. What is the output of the algorithm when $x = 1642$ and $b = 8$?

Solution: Expand(1642, 8) $=$ `"3152"`

Algorithm 4

```
1: procedure EXPAND(x, b)
2:     Input:  integers x and b
3:     Output:  outVal, x written in base b
4:
5:     outVal = an empty string
6:     while (x>0) do
7:         outVal = string(x mod b) + outVal
8:         x = x div b
9:     return(outVal)
```

Modules 2 & 3: Analyzing Algorithms and Big-$\mathcal{O}$ Estimates

Clarification of notation

The following statements all mean the same thing:

$f(x)$ is $\mathcal{O}(g(x))$ OR

$f(x)$ is of $\mathcal{O}(g(x))$ OR

$f(x) = \mathcal{O}(g(x))$ OR

$f(x) \in \mathcal{O}(g(x))$

$\mathcal{O}(g(x))$ is a collection of functions (i.e. a set) so what we should say is $f(x) \in \mathcal{O}(g(x))$, but $f(x) = \mathcal{O}(g(x))$ is commonly used. This can be confusing since $\mathcal{O}(n) = \mathcal{O}(n^2)$ but $\mathcal{O}(n^2) \neq \mathcal{O}(n)$!!

See this interesting thread on StackExchange Big O Notation is element of or is equal. Note the Wiki cites Donald Knuth.

Big-$\mathcal{O}$ classification of common functions

Order of Asymptotic Behaviour. Remember that these are sets. So while it might be said that $2n + 3$ is $\mathcal{O}(n)$; actually its an element in that set: $2n + 3 \in \mathcal{O}(n) \subset \mathcal{O}(n^2) \subset \ldots$.

Here's a list of some functions listed from slowest to fastest function growth (shortest to longest runtime for an algorithm).

- 1 - Constant
- $\log(n)$ - Logarithmic

  Ignore log bases: $\mathcal{O}(\log_{10}(n)) = \mathcal{O}(\log_2(n)) = \mathcal{O}(\log(n))$

  Same growth as $\mathcal{O}(\log(n^k)) = \mathcal{O}(k\log(n)) = \mathcal{O}(\log(n))$
- n - Linear
- $n\log(n)$ - Linearithmic or Loglinear
- n^2 - Quadratic
- $n^2\log(n)$
- n^3 - Polynomial (including $n^4, n^5, n^6, \ldots$)
- 2^n - Exponential (including $3^n, 4^n, 5^n \ldots$)
- $n!$ - Factorial

5 Order each function according to their growth from slowest to fastest (This is the same as execution speed from fastest to slowest for an algorithm).

(a) $n \cdot \log n$

(b) $\log_{10} n$

(c) 10^n

(d) n^{10}

(e) $10n$

(f) 100

(g) n^2

Solution:

(f) 100

(b) $\log_{10} n$

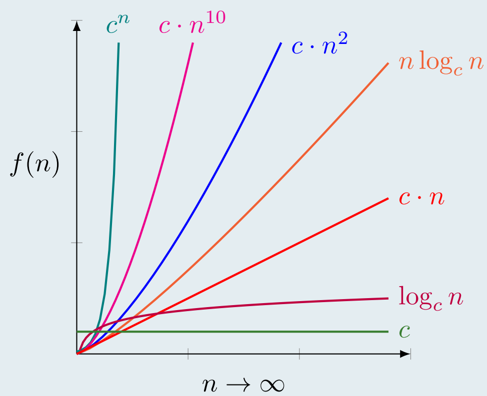
(e) $10n$

(a) $n \cdot \log 2n$

(g) n^2

(d) n^{10}

(c) 10^n



6 The provided expressions are the processing time of an algorithm for problems of size n. For each expression, find the *lowest possible* Big-$\mathcal{O}$ complexity stated in simplest form.

Solution: Given two real functions $f(x)$ and $g(x)$ such that $g(x)$ is strictly positive for sufficiently large x, we say that

$$f(x) \in \mathcal{O}(g(X))$$

if and only if there exist $M, x_0 \in \mathbb{R}$ such that

$$|f(x)| \leq M \cdot g(x)$$

for all $x_0 \leq x$ This formal definition of Big-$\mathcal{O}$ notation is not included in the text. We don't use it directly. Just remember that $f(x)$ is $\mathcal{O}(g(x))$ bound from above by $M \cdot g(x)$, i.e., $\mathcal{O}(g(x))$ is an upper bound.

(a) $100n^2 + 0.0002n^3 + 10,000$

Solution: $\mathcal{O}(n^3)$

From the definition above, we can see that coefficients are ignored. As $n \to \infty$, they have little effect. So we need only compare n^2 and n^3. Since $n^3 = n \cdot n^2$, we have $\mathcal{O}(n^2) \subset \mathcal{O}(n^3)$ and our answer is:

$$\mathcal{O}(n^3)$$

(b) $20n + 2n\log_{10} n + 30$

Solution: $\mathcal{O}(n \log n)$

To find this we use the following property:

$$\text{If } f \in \mathcal{O}(f')\text{and } g \in \mathcal{O}(g') \text{ then } f \cdot g \in \mathcal{O}(f \cdot g)$$

So for the product of two functions we can evaluate the factors.
$2n \in \mathcal{O}(n)$ and $\log_{10} n \in \mathcal{O}(\log_2 n)$. So then our answer is:

$$\mathcal{O}(n \log_n)$$

As a matter of convention, the base is often excluded, however $\mathcal{O}(\log_2 n)$ is sometimes written.

Why do we ignore the base in logs but not exponents? Recall that

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2}$$

So then

$$\log_{10} n \leq (\log_{10} 2) \log_2 n$$

for all n. Here our M is $\log_{10} 2$. The situation is different for the bases of *exponential* functions, but remember that logs are the inverses of exponential functions.

(c) $\log_{10} 5 + \log_2 7$

Solution: $\mathcal{O}(1)$

Both terms are constants. So the answer is: $\mathcal{O}(1)$

(d) $5n + \sqrt{n}$

Solution: $\mathcal{O}(n)$

$\sqrt{n} = n^{1/2}$ and $5n^1 = 5n^{1/2}n^{1/2}$. Now we can see that $\mathcal{O}(\sqrt{n}) \subset \mathcal{O}(n)$. So the answer is: $\mathcal{O}(n)$

(e) $0.01(2^n) + n^5 + \ln n$

Solution: $\mathcal{O}(2^n)$

$\mathcal{O}(\log n) \subset \mathcal{O}(n^a) \subset \mathcal{O}(b^n)$ for $a, b \in \mathbb{R}$ and $b > 1$. That is, an exponential function is slower than a polynomial which is slower than a $\log$. So the answer is

$$\mathcal{O}(2^n)$$

To see that $\mathcal{O}(n^a) \subset \mathcal{O}(b^n)$, note that $\log x < \log y \iff x < y$. So then,

$$\log n^a = a \log n < n \log b = \log b^n$$

for sufficiently large n as we know that $\mathcal{O}(\log n) \subset \mathcal{O}(n)$

(f) $100 \cdot 4^n + 200 \cdot 3^n$

Solution: The answer is:

$$\mathcal{O}(4^n)$$

The text incorrectly states that *"the base of the exponent is irrelevant"* in lesson 1.13. $2^n \leq 3^n$ for all n, but there is no constant M such that $3^n \leq M \cdot 2^n$ for all n. If there was then there would be an integer $x > 0$ such that $M < 3^x$, and then $3^x \cdot 3^{n-x} \leq 3^x \cdot 2^{n-x}$ for sufficiently large n. Which is a contradiction.

7. $f(x) = 6x \log_4 x$ is which of the following? **Mark all that apply**.

○ $\mathcal{O}(6)$ ○ $\mathcal{O}(\log x)$ ○ $\mathcal{O}(6 \log x)$ √ $\mathcal{O}(x \log x)$ √ $\mathcal{O}(x^2)$ √ $\mathcal{O}(2^x)$

Solution: Big-$\mathcal{O}$ is an *upper bound*. So if $f(x)$ is no slower than $x \log x$ then it is also no slower than x^2 and 2^x. In set notation,

$$f(x) \in \mathcal{O}(x \log x) \subset \mathcal{O}(x^2) \subset \mathcal{O}(2^x)$$

8. Let $h(x) = 6x^4 + 2x^3 + x + 100$. Which of the following is true? **Mark all that apply**.

√ $h(x) \in \mathcal{O}(x^4)$ √ $h(x) \in \Theta(x^4)$ √ $h(x) \in \Omega(x^4)$

Solution:

$\mathcal{O}(x^4)$ is an *upper bound*. $\Omega(x^4)$ is a *lower bound*. So then $h(x)$ is also $\Theta(x^4)$, a *tight bound*.

9 Let $h(x) = 6\log_2 x$. Which of the following is true? **Mark all that apply**.

√ $h(x) \in \mathcal{O}(4x^2)$ ○ $h(x) \in \Theta(4x^2)$ ○ $h(x) \in \Omega(4x^2)$

Solution: $6\log_2 x$ is faster than $4x^2$. So then $h(x) \in \mathcal{O}(4x^2)$ and $h(x) \notin \Omega(4x^2)$. Which means $h(x) \notin \Theta(4x^2)$

10 Let $f(x) = x^2 \log_3 x$. Which of the following is true? **Mark all that apply**.

○ $f(x) \in \mathcal{O}(2x\log_{10} x)$ ○ $f(x) \in \Theta(2x\log_{10} x)$ √ $f(x) \in \Omega(2x\log_{10} x)$

Solution: Recall that the base of a logarithmic function can be ignored. So $\log_{10} x$ and $\log_3 x)$ have the same asymptotic behavior, and we really only need to evaluate x^2 and $2x$. We know that $x^2 \notin \mathcal{O}(2x)$, and so neither can it be in $\Theta(2x)$ which gives us our answer.

11 Let $g(x) = 2x\log_{10} x$. Which of the following is true? **Mark all that apply**.

√ $g(x) \in \mathcal{O}(x^2\log_3 x)$ ○ $g(x) \in \Theta(x^2\log_3 x)$ ○ $g(x) \in \Omega(x^2\log_3 x)$

Solution: This is similar to question 10, only here $g(x)$ has a *faster* factor, $2x$. So then $g(x) \in \mathcal{O}(x^2\log_3 x)$ (it's faster) and $g(x) \notin \Omega(x^2\log_3 x) \Rightarrow g(x) \in \Theta(x^2\log_3 x)$.

12 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 5 Some more messy pseudocode

```
1: procedure SomeProcedure2
2:     j=0
3:     for i=0; i<n; i++ do
4:         j=i+j
```

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n\log n)$

C. $\mathcal{O}(n^2)$

D. $\mathcal{O}(n)$

Solution: $\mathcal{O}(n)$

The `for` loop in Line 3 will iterate n times $(0, 1, 2, ..., n-1)$ so line 4 runs n times.

13 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 6 Some messy pseudo-code

```
1: procedure SomeProcedure
2:     for i=1 and i<=n do
3:         j=1
4:         while j<n do
5:             j=j+2
```

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n \log n)$

C. $\mathcal{O}(n^2)$

D. $\mathcal{O}(n)$

Solution: $\mathcal{O}(n^2)$

The `for` loop in Line 2 will iterate n times and the `while` loop will iterate (about) $\frac{n}{2}$ times. Since they are nested, we multiply them together and Line 5 will run $n \cdot \frac{n}{2} = \frac{n^2}{2}$ times.

14 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation. Assume we know that **someMethod(n)** is $\mathcal{O}(\log n)$.

Algorithm 7 Some pseudocode with a method in it

```
1: procedure SomeProcedure3
2:     j=0
3:     for i=0; i<n; i++ do
4:         j=someMethod(n)
```

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n \log n)$

C. $\mathcal{O}(n^2)$

D. $\mathcal{O}(n)$

Solution: $\mathcal{O}(n \log n)$
The `for` loop in line 3 will run n times. Line 4 calls **someMethod(n)** which runs in $\mathcal{O}(\log n)$ time. Since line 4 will run n times, we end up with $n \log n$.

15 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 8 Some more messy pseudocode

```
procedure SomeProcedure4
    while n> 1 do
        n=n/2
```

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n \log n)$

C. $\mathcal{O}(n^2)$

D. $\mathcal{O}(n)$

Solution: $\mathcal{O}(\log n)$

Each iteration of the `while` loop reduces n by half, so we want to know how many times can we divide n by 2 and stay above 1. If $n = 2^m$, then the power m is approximately the value we're seeking. To determine m, take $\log_2$ on both sides giving, $\log_2(n) = \log_2(2^m) = m \cdot \log_2(2) = m$. Since this `while` loop will run $m = \log_2(n)$ times, we get $\mathcal{O}(\log n)$. Recall that the base of the logarithm does not affect the Big-$\mathcal{O}$ representation.

16 Using the provided pseudo-code, find the worst case performance in Big-$\mathcal{O}$ notation.

Algorithm 9

```
procedure
    a = 1
    c = 0
    while a < n do
        for i = 0; i < n; i++ do
            c = c + 1
        a = a * 3
```

A. $\mathcal{O}(\log n)$

B. $\mathcal{O}(n \log n)$

C. $\mathcal{O}(n^2)$

D. $\mathcal{O}(n)$

Solution: $\mathcal{O}(n \log n)$

The index, a, in the `while` loop is multiplied by 3 each iteration causing it to run about $\log_3 n$ iterations. The `for` loop iterates n times for each iteration of the `while` loop. With the nested loops, we multiply to get $\mathcal{O}(n \log n)$