ECS 32B: Conceptual Homework #1

Instructor: Aaron Kaloti

Summer Session #2 2020

1 Identification

Enter the members of your pair. (You can partner with at most one other student.) If you are not partnered with anyone, then leave the second box empty. You can remove the use of `\vspace` in the `.tex` file.

Pair member #1:

Bohao Zou, ID: 917796070

Pair member #2:

Zhikuan Quan, ID: 917800911

2 Problems

Unless explicitly specified, you do not need to justify your answer. Place your answer into the answer boxes; you can remove the use of `\vspace` in the answer boxes in the `.tex` file.

2.1 Big-O

For each question in this subsection, answer with True or False; do not justify your answer.

2.1.1

$2n^2 + 3n + 8$ is $O(n^2)$.

True

2.1.2

$4n^2 + 17 - 8n$ is $O(n^2)$.

True

2.1.3

$5n^2 + 20$ is $O(n)$.

False

2.1.4

$17 - 3n + 8n$ is $O(n)$.

True

[*] This content is protected and may not be shared, uploaded, or distributed.

2.1.5

$8n^4 + 3n^3 - 5n^2 + 9n^4 + 6 + 16n^2$ is $O(n^4)$.

True

2.1.6

$8n^4 + 3n^3 - 5n^2 + 9n^4 + 6 + 16n^2$ is $O(n^3)$.

False

2.1.7

$8n^4 + 3n^3 - 5n^2 + 9n^4 + 6 + 16n^2$ is $O(n^5)$.

True

2.2 Big-O Proofs

Consult the formal definition if big-O in the lecture slides (slide 18). For each question in this subsection, answer with True or False. If your answer is True, then do not justify/explain your answer. If your answer is False, then you should explain why the proposed constants c and n_0 do not form a correct proof. (Naming constants c and n_0 that *do* work – which you do not need to do anyways – is insufficient.)

2.2.1

If we are trying to prove that $4n^2$ is $O(n^2)$, then choosing $c = 5$ and $n_0 = 1$ suffices.

True. if $c = 5$ and $n_0 = 1$, we have $T(n) = 4n^2 \leq 5n^2$ for all $n \geq n_0 = 1$

2.2.2

If we are trying to prove that $5n + 3n^4 + 9$ is $O(n^4)$, then choosing $c = 3$ and $n_0 = 1$ suffices.

False. if we choose $c = 3$ and $n_0 = 1$, the $T(n) = 5n + 3n^4 + 9 = 17$ when $n = 1$, however, $f(n) = 3n^4 = 3$ when $n = 1$. It can be showed that $T(n) \geq cf(n)$ when $n = 1$.

2.2.3

If we are trying to prove that $6n + 2$ is $O(n^2)$, then choosing $c = 2$ and $n_0 = 3$ suffices.

False. if we choose $c = 2$ and $n_0 = 3$, the $T(n) = 6n + 2 = 20$ when $n = 3$, however, $f(n) = 2n^2 = 18$ when $n = 3$. It can be showed that $T(n) \geq cf(n)$ when $n = 3$.

2.2.4

If we are trying to prove that $3n^2 - 10n + 5$ is $O(n^2)$, then choosing $c = 2$ and $n_0 = 4$ suffices.

False. if we choose $c = 2$ and $n_0 = 4$, when we choose $n = 10$ the $T(n) = 3n^2 - 10n + 5 = 205$ when $n = 10$, however, $f(n) = 2n^2 = 200$ when $n = 10$. It can be showed that $T(n) \geq cf(n)$ when $n = 10$.

2.2.5

If we are trying to prove that $3n^2 - 10n + 5$ is $O(n)$, then choosing $c = 1$ and $n_0 = 1$ suffices.

False, there dose not exit constant $c > 0$ and $n_0 \geq 0$ such that for all $n \geq n_0$, we have $T(n) \leq cf(n)$

2.2.6

If we are trying to prove that f_1 is $O(f_2)$, where $f_1(n) = 5n^3 + 6$ and $f_2(n) = 6n^3 - 10n$, then choosing $c = 2$ and $n_0 = 6$ suffices.

True. if $c = 2$ and $n_0 = 6$, we have $f_1(n) = 5n^3 + 6 \leq cf_2(n) = 12n^3 - 20n$ for all $n \geq n_0 = 6$

2.3 Logarithms

Suppose that $f_1(n) = \log_3 n$ and $f_2(n) = \log_7 n$. Answer the following two questions.
Hint for the below two problems: Remember the change-of-base formula for logarithmic functions.

2.3.1

Provide the *smallest* value of c that, when used with $n_0 = 1$, proves that f_1 is $O(f_2)$. Alternatively, if this is not possible, then just say "Not possible" (no justification needed).

For $log_3 n \leq clog_7 n$ when $n \geq 1$, then we have $c \geq \frac{log_3 n}{log_7 n} = log_3 7$. So, the smallest value of c is $log_3 7$

2.3.2

Provide the *smallest* value of c that, when used with $n_0 = 1$, proves that f_2 is $O(f_1)$. Alternatively, if this is not possible, then just say "Not possible" (no justification needed).

For $clog_3 n \geq log_7 n$ when $n \geq 1$, then we have $c \geq \frac{log_7 n}{log_3 n} = log_7 3$. So, the smallest value of c is $log_7 3$

2.4 Big-O and Code #1

Consider the function shown below.

```python
# @n is an integer.
def foo(n):
    i = 3
    x = 0
    while i < n:
        x += 1
        i += 1
    return x
```

For each of the following questions regarding the above function, answer True or False. Do not justify your answer.
Note that since big-O is concerned with asymptotic (i.e. long-run) growth, the fact that the loop in `foo()` does not do any iterations when `n <= 3` can be ignored, since there is a consistent growth rate when we look at how `n` increases beyond `n > 3`.

2.4.1

In the "worst case"[1], `foo()` takes $O(n)$ time.

True

2.4.2

In the worst case, `foo()` takes $O(n^2)$ time.

True

2.4.3

In the worst case, `foo()` takes $O(n \lg n)$ time.

True

[1]In this particular problem, you can think of the worst case as being identical to the best case, since there is no meaningful distinction between the two.

2.5 Big-O and Code #2

Consider the function shown below.

```python
# @vals is a list of integers.
def foo(vals):
    i = 4
    s = 0
    while i < len(vals):
        s += vals[i]
        i += 1
    return s
```

Answer each of the following questions about the above function.

2.5.1

As stated in lecture, `len(vals)` takes constant time. State whether this is true or false: `foo()` runs in $O(n)$ time[2], where n is `len(vals)`. If your answer is false, explain why.

True

2.5.2

Suppose that `len(vals)` now takes linear time to evaluate. State whether this is true or false: `foo()` runs in $O(n)$ time, where n is `len(vals)`. If your answer is false, explain why.

False. It should be take $O(n^2)$ time. This is because we must take $O(n)$ time to find the length of vals in each iteration.

2.6 Big-Ω and Big-Θ

For each question in this subsection, answer with True or False; do not justify your answer. $f(n) = 15n^3 + 18n^4 - 6 + 12n^2 + 2n^3$.

2.6.1

$f(n)$ is $\Omega(1)$.

True

2.6.2

$f(n)$ is $\Omega(n^3)$.

True

2.6.3

$f(n)$ is $\Omega(n^5)$.

False

2.6.4

$f(n)$ is $\Theta(n^3)$.

False

2.6.5

$f(n)$ is $\Theta(n^4)$.

[2] As with the previous problem's function, in this function, there is no meaningful distinction between worst case and best case, since the entire list will always be traversed.

True

2.7 Big-Ω and Big-Θ of Code

Consider the function shown below.

```python
# @vals is a list of integers.
# @target is an integer.
# This function returns the index of the first occurrence
# of @vals in the given list.
# Only EVERY OTHER element of @vals is considered; the rest are ignored.
def search_every_other(vals, target):
    for i in range(0, len(vals), 2):
        if vals[i] == target:
            return i
    return -1  # not found
```

Answer each of the following questions about the above function. Let n refer to `len(vals)`.

2.7.1

Is the following statement true or false? In the worst case, `search_every_other()` runs in $O(n!)$ time. Do not justify your answer.

True

2.7.2

Is the following statement true or false? In the worst case, `search_every_other()` runs in $\Omega(\lg n)$ time. Do not justify your answer.

True

2.7.3

Provide a function $f(n)$ such that, in the worst case, `search_every_other()` runs in $\Theta(f(n))$ time. $f(n)$ should not have any leading coefficients and should only have one term.

$f(n) = n$

2.7.4

Is the following statement true or false? In the *best* case, `search_every_other()` runs in $\Theta(1)$ time. If your answer is true, do not justify it; if your answer is false, then justify it.

True

2.7.5

Is the following statement true or false? In the *best* case, `search_every_other()` runs in $O(n)$ time. If your answer is true, do not justify it; if your answer is false, then justify it.

False. In the best case, this function should runs in $O(1)$ time. The best case is that the target is in the first index of vals. So, it should be $O(1)$ time.

2.7.6

Provide a function $f(n)$ such that, in the *average* case, `search_every_other()` runs in $\Theta(f(n))$ time. $f(n)$ should not have any leading coefficients and should only have one term. Here, we define the "average case" as occurring when the target is first found halfway through the list.

$f(n) = n$

2.8 2D Lists

Consider the function shown below.

```python
# @vals1 and @vals2 are 2D lists of integers of the same dimensions and are squares.
# This function returns True if @vals1 and @vals2 have the same values.
# Note that since the == can compare 2D lists, this function is useless.
def compare(vals1, vals2):
    for r in range(len(vals1)):
        for c in range(len(vals1[r])):
            if vals1[r][c] != vals2[r][c]:
                return False
    return True
```

For each of the following questions regarding the above function, answer True or False. Do not justify your answer. Let n refer to `len(vals)`. (As stated above, `vals1` and `vals2` are each squares.)

2.8.1

In the worst case, `compare()` runs in $O(n)$ time.

False

2.8.2

In the worst case, `compare()` runs in $\Omega(n^2)$ time.

True

2.8.3

In the worst case, `compare()` runs in $\Theta(n^2)$ time.

True

2.8.4

In the worst case, `compare()` uses $\Theta(n^2)$ auxiliary space.

False

2.9 Last One

Consider the function shown below.

```python
# @vals is a n-by-n 2D list of integers, where n = len(vals).
def foo(vals):
    x = []
    for i in range(len(vals)):
        x.append(vals[i][i])
    return x
```

Answer the following questions. Let n refer to `len(vals)`.

2.9.1

What is the worst-case time complexity of `foo()` in big-Θ notation? Explain.

The worst-case time complexity in big-Θ notation is $\Theta(n)$. The worst-case is that this function will iterate n times by searching elements of vals. $T(n) = n + 2$, So this function runs in $O(n)$ and $\Omega(n)$.

2.9.2

What is the worst-case space complexity (auxiliary space) of `foo()` in big-Θ notation? Explain.

The worst-case space complexity complexity in big-Θ notation is $\Theta(n)$. This is because the x will contain n element from vals[i][i] in the worst-case. So the auxiliary space of this function is $O(n)$ and $\Omega(n)$.