ECS 122B: Conceptual Homework #1

Instructor: Aaron Kaloti

Summer Session #2 2020

1 Changelog

You should always refer to the latest version of this document.

- v.1: Initial version.

2 Grading

- **Due date**: the night of Wednesday, 08/12. (Due to the "grace period", Gradescope will say 12:30 AM on Thursday, 08/13.)
- A subset of the problems will be graded for correctness. The rest will be graded on completion.

3 Submission Requirements

- Your submission must be typed with LaTeX. **Handwritten/scanned solutions, or solutions not typed with LaTeX, will earn no credit.**
  - *Especially if you have never used LaTeX before*, you should avoid doing this assignment at the last minute, because it will take a bit of time to get used to LaTeX and to type your answers up into a LaTeX document.
  - I will not teach LaTeX during lecture, because that would be a waste of time. You should be able to pick up LaTeX through a combination of inspecting the `.tex` file corresponding to this document (which is uploaded to Canvas) and looking up certain aspects of LaTeX (e.g. on the Overleaf website) that you do not understand. It will take a bit of time, but you'll quickly become comfortable enough with LaTeX to not have to think about it.
  - If you have never used LaTeX, then you may find Overleaf, an online LaTeX editor, convenient. I believe that Overleaf also supports collaboration. (As stated below, you can partner with at most one other student.) There are also many means of compiling LaTeX directly on your computer, and there are even some IDEs that support LaTeX. (In case you are curious, I use `pdflatex` to compile LaTeX documents on the command line / terminal.)
- Your submission must consist of two files:
  1. `hw1_answers.tex`: the LaTeX file containing your answers.
     - The file `conceptual_hw1.tex` on Canvas is the LaTeX file that was used to make this PDF (`conceptual_hw1.pdf`). Use it as a template for your answers. (This is especially important because it will keep the numbering of the problems consistent between your answers and the questions.) Just make sure that your `.tex` file has the correct name. If you wish, you can experiment with the style of the document too, e.g. the background color of the code segments, the margin sizes, etc. Again, just make sure that the numbering of the sections is the same as it originally was; this matters even if you mark the pages properly on Gradescope (see below).
  2. `hw1_answers.pdf`: the PDF generated from your LaTeX file.
- You may be penalized if, when submitting on Gradescope, you mark the wrong page for a given homework problem, because dealing with this slows down grading. At the beginning of the 08/06 lecture, I will talk (or did already talk, depending on when you are reading this) about how to mark the pages of your submission on Gradescope.
- When using LaTeX, you should make use of the math notation/mode where sensible. (The math mode refers to when you put mathematical equations, etc. between dollar signs so that LaTex makes them look nice.) Repeated failures to do this, to the point that the purpose of using LaTeX is defeated or the readability of your answers is impeded, may result in a penalty on this assignment.

4 Regarding Collaboration

- You may not copy answers from any sources, including online sources such as Chegg, StackOverflow, or any solutions manual of any textbook.
- You may partner with ***at most one*** other student on this assignment. In other words, you can work in *pairs*. You do not have to partner with anyone. (In fact, I think it is better that you do not.) If you partner with someone, it must be a committed partnership; that is, you two will have the same submission, and you must mark on Gradescope that you have partnered for this assignment by following the directions here.
- If students that were not in the same pair seem to have excessively similar answers, they will be reported to the OSSJA for suspicion of academic misconduct. Do not copy answers from (or share answers with) any student who you are not partnered with.

5 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:

Julio Beas

Pair member #2:

Han Nguyen

6 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.

6.1 Big-O

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

6.1.1

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

True

6.1.2

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

True

6.1.3

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

False

6.1.4

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

True

6.1.5

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

True

6.1.6

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

False

6.1.7

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

True

6.2 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.)

6.2.1

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

True

6.2.2

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. At $c = 3$ and $n_0 = 1$, we have for $n = 1 \geq n_0$, $T(n) = 5n + 3n^4 + 9 > 3n^4$, which does not align with the definition

of Big-O.

6.2.3

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

False. At $c = 2$ and $n_0 = 3$, we have for $n = 3 \geq n_0$, $T(n) = 6n + 2 > 2n^2$, which does not align with the definition of Big-O.

6.2.4

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

True

6.2.5

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

False. At $c = 1$ and $n_0 = 1$, we have for $n = 4 \geq n_0$, $T(n) = 3n^2 - 10n + 5 > n$, which does not align with the definition of Big-O.

6.2.6

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

6.3 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.

6.3.1

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).

Not possible

6.3.2

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).

$c = 1$

6.4 Big-O and Code #1

Consider the function shown below.

```
int foo(int n)
{
    int i = 3;
    int 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`.

6.4.1

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

True

6.4.2

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

False

6.4.3

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

False

6.5 Big-O and Code #2

Consider the function shown below.

```
int foo(const std::vector<int>& vals)
{
    int i = 4;
    int s = 0;
    while (i < vals.size())
    {
        s += vals[i];
        i += 1;
    }
    return s;
}
```

[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.

Answer each of the following questions about the above function.

6.5.1

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

False. If `vals.size()` takes constant time then `foo()` takes $O(1)$ time.

6.5.2

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

True

6.6 Big-Ω and Big-Θ

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

6.6.1

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

True

6.6.2

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

True

6.6.3

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

False

6.6.4

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

[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 array will always be traversed.

True

6.6.5

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

False

6.7 Big-Ω and Big-Θ of Code

Consider the function shown below.

```
int searchEveryOther(const std::vector<int>& vals, int target)
{
    for (unsigned i = 0; i < vals.size(); i += 2)
        if (vals[i] == target)
            return i;
    return -1;
}
```

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

6.7.1

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

True

6.7.2

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

True

6.7.3

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

$f(n) = n$

6.7.4

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

True

6.7.5

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

False. In our best case we would want to find `target` right away which takes $O(1)$ time.

6.7.6

Provide a function $f(n)$ such that, in the *average* case, `searchEveryOther()` 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 array.

$f(n) = n$

6.8 2D Arrays

Consider the function shown below.

```
// @vals1 and @vals2 are 2D arrays of integers of the same dimensions and are squares.
// This function returns True if @vals1 and @vals2 have the same values.
bool compare(const std::vector<std::vector<int>>& vals1,
             const std::vector<std::vector<int>>& vals2)
{
    for (unsigned r = 0; r < vals1.size(); r++)
    {
        for (unsigned c = 0; c < vals1[r].size(); c++)
            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 `vals.size()`. (As stated above, `vals1` and `vals2` are each squares.)

6.8.1

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

False

6.8.2

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

True

6.8.3

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

True

6.8.4

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

True

6.9 Last One

Consider the function shown below.

```
// @vals is a n-by-n 2D array of integers, where n = vals.size().
std::vector<int> foo(const std::vector<std::vector<int>>& vals)
{
    std::vector<int> x;
    for (int i = 0; i < vals.size(); ++i)
        x.push_back(vals[i][i]);
    return x;
}
```

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

6.9.1

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

The worst-case time complexity of `foo()` is $\Theta(n)$. Since $T(n) = n + C$, where $C \leq n$, we observe that $T(n) = O(n)$ at $c = 2$ and for all $n \geq n_0 = 1$. Also, we observe that $T(n) = \Omega(n)$ at $c = 1$ and for all $n \geq n_0 = 1$.

6.9.2

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

The worst space complexity of `foo()` is $\Theta(n)$. Since `.push_back()` takes $O(1)$ time, the function will take $\Theta(n)$ space complexity when it loops through $n =$ `vals.size()` values.