EX1: Algorithmic Analysis I

Due date: Friday April 16, 2021 at 11:59 pm PDT

Instructions:

Submit your responses to the "EX1: Algorithmic Analysis I" assignment on Gradescope here: https://www.gradescope.com/courses/259163/assignments/1154087/. Make sure to log in to your Gradescope account using your UW email to access our course.

Runtime Bound Review:

$\mathcal{O}$: **Upper bound** on how quickly function grows. Informally: "Less than or equal to".

Ω: **Lower bound** on how quickly function grows. Informally: "Greater than or equal to".

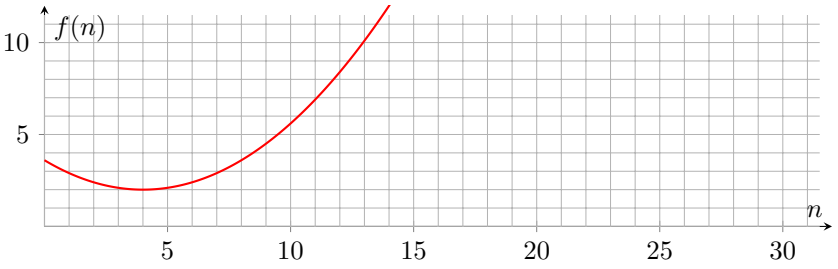
Θ: **Tight bound** on how quickly function grows. Informally: "Equal to".

1. Asymptotic analysis: Visually

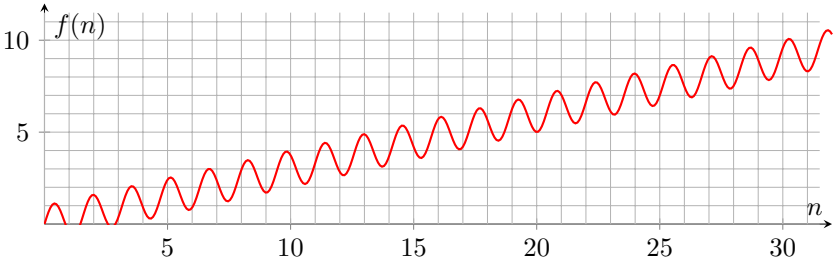
For each of the following plots, provide a tight big-$\mathcal{O}$ bound, a tight big-Ω bound, and a big-Θ bound. You do not need to show your work; just list the bounds. If a particular bound doesn't exist for a given plot, briefly explain why. Assume that the plotted functions continue to follow the same trend shown in the plots as n increases. Each provided bound must either be a constant or a simple polynomial, **from the following possible answers**.

$$n^2, 1, n, log(n), \frac{1}{n}$$

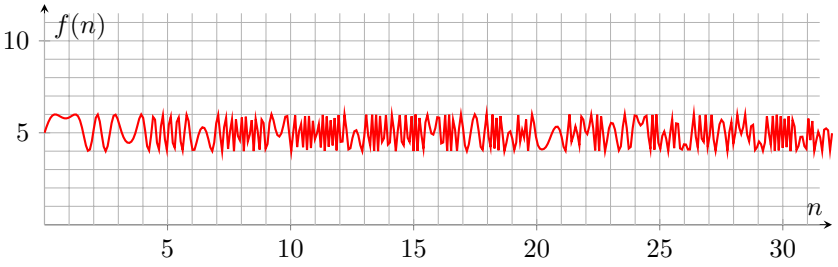
(a)



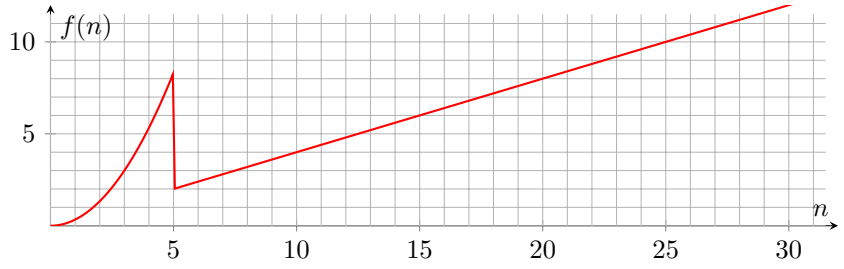
(b)



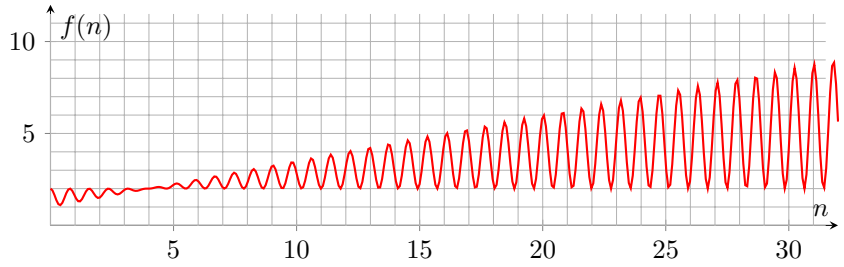
(c)



(d)



(e)



2. Case and Asymptotic Analysis

In this problem we will analyze code in the `printSmilies` method below. The code is not particularly efficient (i.e. you should not use this code snippet as a model for how to use data structures). You should assume that this code uses the Java implementations of `ArrayList` and `LinkedList` – in particular, that `add` on a `LinkedList` takes constant time because the object stores a reference to the back of the list.

```
1  public void printSmilies(ArrayList<Integer> input, int target) {
2      LinkedList<Integer> products = new LinkedList<>();
3      for (int i = 0; i < input.size(); i++) {
4          for (int j = 0; j < input.size(); j++) {
5              products.add(input.get(i) * input.get(j));
6          }
7      }
8
9      LinkedList<Integer> occurrencesOfToFind = new LinkedList<>();
10
11     while (products.size() != 0) {
12         int next = products.remove(0); // remove the value at index 0
13         if (next == target) {
14             occurrencesOfToFind.add(next);
15         }
16     }
17
18     for (int i = 0; i < occurrencesOfToFind.size(); i++) {
19         for (int j = 0; j < occurrencesOfToFind.size(); j++) {
20             System.out.println(":D");
21         }
22     }
23 }
```

Answer the following questions about the runtime of the `printSmilies` method. In this problem, `input` is defined to have n elements. For each of the problems asking for asymptotic bounds, if your answer uses a variable, your answers should be in terms of n. All big-$\mathcal{O}$, big-Ω, and big-Θ bounds should be simplified. You can assume `System.out.println` calls will always run in constant time.

Remember that "best/worst-case" refer to the *inputs* that yield the fastest or slowest possible runtime functions, respectively.

(a) Give the simplified big-Θ bound for runtime of the first loop on lines 3-7. (Note: the runtime bound is the same across all cases). If your answer uses a variable, it should be in terms of n, the size of the input.

(b) Note that the runtime for last loop is based on the size of `occurrencesOfToFind` (note: this is not the same variable as n, the input size to this method). To figure out that last loop's runtime, let's break it down by analyzing the previous loop on lines 11-16:

- What sort of state of the parameters `input` and `target` will trigger the worst case where `occurrencesOfToFind` is as big as possible? Give an example 5-element list value for `input` and an example value for `target` that would trigger the worst case. Then, describe the general pattern that would trigger the worst case in no more than 1 sentence.

- Give a simplified big-Θ bound for the size of `occurrencesOfToFind` in this worst case. If your answer uses a variable, it should be in terms of n, the size of `input`.

- What sort of state of the parameters `input` and `target` will trigger the best case where `occurrencesOfToFind` is as small as possible? Give an example 5-element list value for `input` and an example value for `target` that would trigger the best case. Then, describe the general pattern that would trigger the best case in no more than 1 sentence.

- Give a simplified big-Θ bound for the size of `occurrencesOfToFind` in this best case. If your answer uses a variable, it should be in terms of n, the size of `input`.

(c) Consider the worst case runtime situation discussed in the previous question for the loop on lines 18-22 for the following problems.

- Give a tight big-$\mathcal{O}$ for the runtime of lines 18-22. If your answer uses a variable, it should be in terms of n, the size of `input`.

- Give a tight big-Ω for the runtime of lines 18-22. If your answer uses a variable, it should be in terms of n, the size of `input`.

(d) Consider the best case runtime situation discussed in the previous question for the loop on lines 18-22 for the following problems.

- Give a tight big-$\mathcal{O}$ for the runtime of lines 18-22. If your answer uses a variable, it should be in terms of n, the size of `input`.

- Give a tight big-Ω for the runtime of lines 18-22. If your answer uses a variable, it should be in terms of n, the size of `input`.