CPSC 320: Intermediate Algorithm Design and Analysis
Assignment #1, due Monday, January 20th, 11:59pm

[6] 1. The pair of functions below indicate algorithm runtimes as a function of n. Assume that $k \in \{1, 2, \cdots\}$, $\epsilon > 0$, and $c > 1$. For each pair we want to determine which functions are **Big-Ω** of one another. Write:

- **LEFT:** if left $\in \Omega$(right), i.e., the function on the left is Big-Ω of the function on the right.
- **RIGHT** if right $\in \Omega$(left).
- **SAME** if left $= \Theta$(right).

There is no need to justify your answers.

[1] (a) 2^n vs. $2^{n/2}$

Solution: LEFT: $2^{n/2} = \sqrt{2}^n \leq 2^n$ for $n \geq 0$, thus $2^n \in \Omega(2^{n/2})$.

[1] (b) $\log^k n$ vs. n^ϵ

Solution: RIGHT: Consider the limit $\lim_{n\to\infty} \frac{n^\epsilon}{\log^k(n)}$. By apply l'Hospital's rule k times this limit is equivalent to $\lim_{n\to\infty} \frac{n^k \epsilon(\epsilon-1)\cdots 1}{k!}$ which diverges. Thus $n^\epsilon \in \Omega(\log^k(n))$.

[1] (c) $\sqrt[c]{n}$ vs. $\log(c^n)$

Solution: RIGHT: $\sqrt[c]{n} \leq n \log(c)$ for $\log(c)^{-c/c-1} \leq n$, thus $\log(c^n) \in \Omega(\sqrt[c]{n})$.

[1] (d) c^n vs. $2^{\log(n^\epsilon)}$

Solution: LEFT: $2^{\log(n^\epsilon)} = n^\epsilon \in O(c^n)$ thus $c^n \in \Omega(n^\epsilon)$.

[1] (e) $\frac{\log(n)}{n}$ vs. 1

Solution: RIGHT: $\frac{\log(n)}{n} \leq 1$, thus $1 \in \Omega(\frac{\log(n)}{n})$

[1] (f) $\frac{1}{n}\sum_{i=1}^n i$ vs. $\sqrt{2}^{\log_2(n)}$

Solution: LEFT: $\frac{1}{n}\sum_{i=1}^n = \frac{n+1}{2}$, $\sqrt{2}^{\log_2(n)} = \sqrt{n}$, thus $\frac{1}{n}\sum_{i=1}^n i \in \Omega\sqrt{2}^{\log_2(n)}$

[6] 2. The Hamiltonian Path Problem is a famous algorithmic problem on graphs. Given a graph G with n vertices, the objective is to find a path containing every vertex exactly once (or decide that no such path exists). In other words, the objective is to find a sequence $a_1, \ldots, a_n$ of integers such that

- each integer in $\{1, \ldots, n\}$ appears exactly once in the sequence,
- there is an edge between vertex a_i and vertex a_{i+1} for all $1 \leq i < n$.

[3] (a) Donald thinks he is pretty smart and designs an algorithm for solving this problem with running time $\Theta(n^n)$. Boris thinks he is smarter and designs an algorithm with running time $\Theta(n!)$.

Donald claims that his algorithm is asymptotically as good as Boris'. In other words, he claims that n^n is $O(n!)$. Is he right? Prove your answer (using the definition of big-O).

Solution: It is not true: $n!$ is $o(n^n)$. To prove this, for each $\epsilon > 0$, we must find an $n_0 \in \mathbb{N}$ such that $n! \leq \epsilon n^n \ \forall n \geq n_0$.
Consider any $\epsilon > 0$. The desired inequality is equivalent to $\prod_{i=1}^{n} \frac{n}{i} \geq 1/\epsilon$. It suffices to prove that $\prod_{i=1}^{n/2} \frac{n}{i} \geq 1/\epsilon$ because then

$$\prod_{i=1}^{n} \frac{n}{i} \;=\; \underbrace{\left(\prod_{i=n/2+1}^{n} \frac{n}{i}\right)}_{\geq 1} \cdot \left(\prod_{i=1}^{n/2} \frac{n}{i}\right) \;\geq\; \prod_{i=1}^{n/2} \frac{n}{i} \;\geq\; 1/\epsilon,$$

which is what we want to prove. For $i \leq n/2$, each factor $\frac{n}{i}$ is at least 2, so $\prod_{i=1}^{n/2} \frac{n}{i} \geq 2^{n/2}$. By choosing $n \geq 2\log_2(1/\epsilon)$ we ensure that $2^{n/2} \geq 1/\epsilon$.
Thus, for any $\epsilon > 0$, we have $\prod_{i=1}^{n} \frac{n}{i} \geq 1/\epsilon$ for all $n \geq n_0 \coloneqq 2\log_2(1/\epsilon)$.

[3] (b) Justin and Angela think they can outsmart those two. Justin develops an algorithm with running time $O(4^n)$. Angela develops an algorithm[1] with running time $\Theta(n^2 2^n)$. Justin scoffs at Angela and claims that her algorithm is worse because of the ugly n^2 factor. Angela disagrees: she claims that $n^2 2^n$ is $O(4^n)$. Is she right? Prove your answer (using the definition of big-O).

Solution: It is true: $n^2 2^n$ is $O(4^n)$. To prove this, we must find $c > 0$ and $n_0 \in \mathbb{N}$ such that $n^2 2^n \leq c4^n \ \forall n \geq n_0$. Set $c = 1$. The desired inequality is equivalent to $n^2 \leq 2^n$. This holds so long as $n \geq 2\log_2(n)$, which holds for $n \geq 8$. Thus, the desired inequality holds, taking $n_0 = 8$.

[10] 3. You have been hired by *Western Wells* - a water well company - to find a water source for the city of Saskatoon. The company has plotted out a 1-by-n grid in which it knows exactly one grid contains an underground stream that is suitable. You however have also minored in hydrology[2] and will be able to determine, if when a pilot hole does not find the water source, to which end of the grid the source is.

One approach is to use binary search - drill in the grid at position $n/2$ and recursively drill at $n/4$ or $3n/4$ if needed, and repeat.

[1] (a) In the worst-case, how many holes will need to be drilled?

Solution: Suppose the source in the first location (i.e. location one). As binary search runs in $O(\log n)$ time in the worst-case, approximately $\log n$ holes will be drilled to find a source at this location.

Before commencing operations you are told the following: if a hole is drilled to the right of the actual source, the drill bit will break due to the geology of the area but drilling to the left of the source will not break the bit and thus can be reused. (That is, if the true location of the source is at location i, drilling at any location $j > i$ will break the bit but at any location $j < i$ will not.)

[1] Such an algorithm actually exists. It is called the Bellman-Held-Karp algorithm. It is based on dynamic programming, which we will cover later in the term.

[2] Hydrology - The study of underground water flows.

[1] (b) Suppose you have only one drill bit. Describe an algorithm that is guaranteed to find the source. Compare the runtime of your algorithm to that of the binary search algorithm - which one would you expect to be faster in the worst-case?

Solution: We can start drilling at the first grid and incrementally move right until the source is found. In the worst-case this would require $O(n)$ holes to be drilled and would take the same amount of time. The binary search approach is faster and guaranteed to find the source if we are not constrained by the number of drill bits.

[6] (c) The city's water department is only able to acquire two drill bits for you to use. Further the city's conservation department will not allow you to drill n holes. Describe an algorithm that will find the water source that will drill at most $f(n)$ holes where $f(n) \in o(n)$. (E.g. $f(n) = n/2$ is not acceptable.)

Solution: The idea of our approach is to use the first drill bit to shrink our search to an interval of some size in $o(n)$, then to do a linear search using the second bit on this interval.

Consider drilling at locations 1, $\lfloor\sqrt{n}\rfloor$, $\lfloor 2\sqrt{n}\rfloor$, $\lfloor 3\sqrt{n}\rfloor\ldots$, and suppose the bit breaks at location $i\lfloor\sqrt{n}\rfloor$. From these we have that the source is located in between $\lfloor(i-1)\sqrt{n}\rfloor$ and $\lfloor i\sqrt{n}-1\rfloor$. We then use the second drill bit to drill from the lower end of the interval until we find the source.

[2] (d) What is the runtime of your algorithm? Asymptotically, how many holes will it dig in the worst-case?

Solution: Consider at most how many times each bit was used - this is how many holes were drilled and is proportional to the runtime of the algorithm. For the first bit we drill at most $\sqrt{n}$ holes. For the second bit we will drill at most $\sqrt{n}+1$ holes to find the source. Together both bits result in at most $2\sqrt{n}+1$ holes which is $O(\sqrt{n})$ and in $o(n)$.

[8] 4. For any two real numbers x and y, everyone knows that exactly one of "$x < y$" or "$x \geq y$" must hold.

In class we said that $f \in \Omega(g)$ is analogous to "$f \geq g$", and $f \in o(g)$ is analogous to "$f < g$". Let us see if that analogy extends to exactly one of those properties being true.

[4] (a) Do there exist functions $f, g : \mathbb{N} \to \mathbb{R}_{>0}$ such that $f \notin \Omega(g)$ **and** $f \notin o(g)$? Prove your solution.

Solution: Yes. The same example from class works. Define

$$f(n) = \begin{cases} 1 & (n \text{ even}) \\ n & (n \text{ odd}) \end{cases} \qquad g(n) = \begin{cases} n & (n \text{ even and } n > 0) \\ 1 & (\text{otherwise}). \end{cases}$$

Suppose that $g \in O(f)$. Then there exists $c > 0$ and $n_0 \in \mathbb{N}$ such that $g(n) \leq c \cdot f(n)$ for $n \geq n_0$. Consider $n > \max\{c, n_0\}$ such that n is even. Then $g(n) = n$ and $f(n) = 1$. So $g(n) > c = c \cdot f(n)$, which is a contradiction. Thus $g \notin O(f)$, i.e., $f \notin \Omega(g)$.

Suppose that $f \in o(g)$. Then for all $\epsilon > 0$, there exists $n_0 \in \mathbb{N}$ such that $f(n) < \epsilon g(n)$ for all $n \geq n_0$. Consider $n > \max\{\epsilon, n_0\}$ such that n is odd. Then $g(n) = 1$ and $f(n) = n$. So $f(n) > \epsilon = \epsilon \cdot g(n)$, which is a contradiction. Thus $f \notin o(g)$.

[4] (b) Do there exist functions $f, g : \mathbb{N} \to \mathbb{R}_{>0}$ such that $f \in \Omega(g)$ **and** $f \in o(g)$? Prove your solution.

Remarks. You should not use the limits "definition" of Big-O. Also, note that f and g cannot evaluate to 0.

Solution: No. Suppose that $f \in o(g)$ and $f \in \Omega(g)$. Then

$$f \in o(g) \implies \forall \epsilon > 0,\ \exists n_0(\epsilon) \in \mathbb{N} \text{ s.t. } f(n) < \epsilon g(n)\ \forall n \geq n_0(\epsilon) \tag{1}$$

$$f \in \Omega(g) \implies \exists \alpha > 0,\ \exists m \in \mathbb{N} \text{ s.t. } f(n) \geq \alpha g(n)\ \forall n \geq m. \tag{2}$$

Let $q = \max\{n_0(\alpha), m\}$. Then

$$\alpha g(q) \ \leq\ f(q)\ <\ \alpha g(q),$$

where the first inequality is by (2) (using $q \geq m$) and the second is by (1) (using $q \geq n_0(\alpha)$). This is a contradiction.

[2] 5. **OPTIONAL BONUS QUESTION:** You have taken over Barry the Botanist's research project. Barry has found a miracle food supplement that, when sprayed onto a plant, will increase its yield ten-fold within a week's time! Unfortunately Barry has left n unique types of food supplements that were being tested without telling you which one was the sole miracle food supplement, and which are just mediocre fertilizers.

Having access to only $O(\log n)$ laboratory plants, describe an algorithm that will determine which of the sprays is the miracle food supplement within a week's time by applying the food supplements to the plants.[3]

Solution: We must apply each of the n sprays to a plant to be able to determine whether or not it is the miracle food supplement. However, we do not have enough plants to a one-to-one application as there are only about a $O(\log(n))$ amount. As $\lceil log(n) \rceil \in O(\log(n))$, we will assume that we have this number of plants.

A strategy to identify the desired spray works as follows. Label each plant uniquely as a number in $[0, n-1]$, in binary representation. Thus each spray is a binary string with 0s and 1s. For each spray, if the leftmost bit is a one, apply it to the first plant. If the second-leftmost bit is a one, apply it to the second plant, and so on.

By within a week's time, a set of our test plants (possibly none!) will show the results of the miracle food supplement. We can then infer which single spray is the desired one by reflecting on our strategy.

Eg. For 6 sprays we have 3 plants. Below are sprays labelled as binary numbers and to which plants they were applied. Suppose plants 2 and 3 showed the miracle effect. We can then infer that it was spray "011" that is the miracle food supplement.

```
000,    {}
001,    {3}
010,    {2}
011,    {2,3}
100,    {1}
101,    {1,3}
```

[3] You can assume using multiple sprays together will not cancel the "miracle" effect.