Math 560 Homework 5 Fall 2021

Due: Wednesday, October 20

1. **Order Statistics** [1 pt]

   For the following, indicate True or False:

   (a) $2n^3 - 8n^2 + 32n + 9 \in O(n^3)$
   (b) $2n^3 - 8n^2 + 32n + 9 \in \Omega(n^3)$
   (c) $2n^3 - 8n^2 + 32n + 9 \in o(n^3)$
   (d) $n^p \in O(e^n)$, where $p \in \mathbb{R}$ and $p \geq 0$
   (e) $e^n \in O(n^p)$, where $p \in \mathbb{R}$ and $p \geq 0$
   (f) $n^p \in o(e^n)$, where $p \in \mathbb{R}$ and $p \geq 0$
   (g) $\sqrt{n} \in O(1)$ (side note: $O(1)$ is called 'constant time')
   (h) $\sqrt{n} \in O(n)$
   (i) $\log n \in o(n^p)$, where $p \in \mathbb{R}$ and $p > 0$
   (j) $n^p \in o(\log n)$, where $p \in \mathbb{R}$ and $p > 0$

   It may help to know an additional way of defining little-o: We say $f \in o(g)$ if

   $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

2. **Logarithms** [1 pt]

   What is the relationship between $\log(n)$ and $\log(n^2)$? Indicate which of the following are True (provide a sentence explaining why your choices are correct):

   (a) $\log(n^2) \in O(\log n)$
   (b) $\log(n^2) \in o(\log n)$
   (c) $\log(n^2) \in \Omega(\log n)$
   (d) $\log(n^2) \in \omega(\log n)$
   (e) $\log(n^2) \in \Theta(\log n)$

3. **Factorials** [1 pt]

   True or False:

   $$(n+1)! \in \Theta(n!)$$

   You must provide an explanation of your answer.

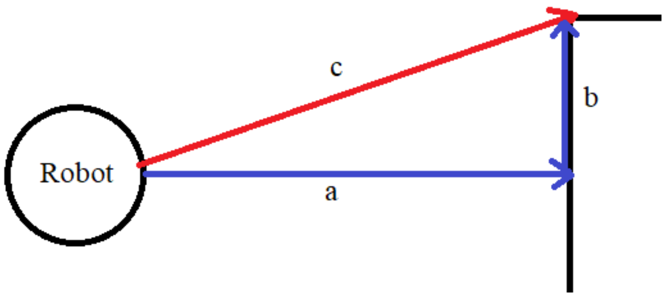
4. **Robots and Triangles** [2 pts]

When assigning robots to perform automated tasks, an important problem that often needs to be addressed is how to route the robot around obstacles in its path. There are two broad approaches to this problem:

Approach #1: scan the room in advance, and plot a route that goes around the obstacles (a smart-car approach).

Approach #2: walk towards the destination, and simply go around any obstacles that the robot runs into (a roomba approach).

If the obstacle is some rectangular table in the way, then the following image shows the two different approaches (#1 in red and #2 in blue), with the distances labeled a, b, and c.



Prove that, in terms of total distance traveled, these two approaches are computationally equivalent, i.e., prove that $(a + b) \in \Theta(c)$ by finding positive real numbers k_1 and k_2 such that

$$k_1 \cdot c \;\leq\; a + b \;\leq\; k_2 \cdot c.$$

You may assume that the obstacle has flat sides, and that the robot collides with the obstacle perpendicular to its surface.

5. **Resizing an Array** [6 pts]

When we discuss algorithmic complexity, we assume that read/write times are constant. What we are really assuming is that the people who designed the memory management of our machines did a good job and guarantee us that all reads or writes will take constant time.

This is an easy promise to keep for fixed size data stored on the stack, but what about dynamically allocated memory stored in the heap?

- Allocating the initial array is easy, because its size is originally given by the user. This allows the machine to allocate the initial memory just like it would on the stack.
- Deallocating the final array is easy, because the size is tracked throughout computation. This allows the machine to deallocate the final memory just like it would on the stack.
- Resizing the array is more complicated...

Consider an initial array of size 1, which then has n objects appended to the end of it. For each of these writes to be an average constant time, we need the total complexity of appending n objects to be $O(n)$. Let's consider three different approaches. For each approach, calculate the complexity of appending n objects to the end of the array and determine if the approach will give us constant time read/write.

(a) Every time that the array runs out of space, its length is increased by 1. If the array is currently size k: we allocate space for a new array of size $k+1$, copy over the original k elements, then put the new object into the final $k+1$ location.

(b) Every time that the array runs out of space, its length is increased by 100. If the array is currently size k: we allocate space for a new array of size $k+100$, copy over the original k elements, then put the new object into the $k+1$ location (leaving the locations $k+2$ through $k+100$ empty for now, but available for storing more elements in the future).

(c) Every time that the array runs out of space, its length is doubled. If the array is currently size k: we allocate space for a new array of size $2k$, copy over the original k elements, then put the new object into the final $k+1$ location (leaving the locations $k+2$ through $2k$ empty for now, but available for storing more elements in the future).

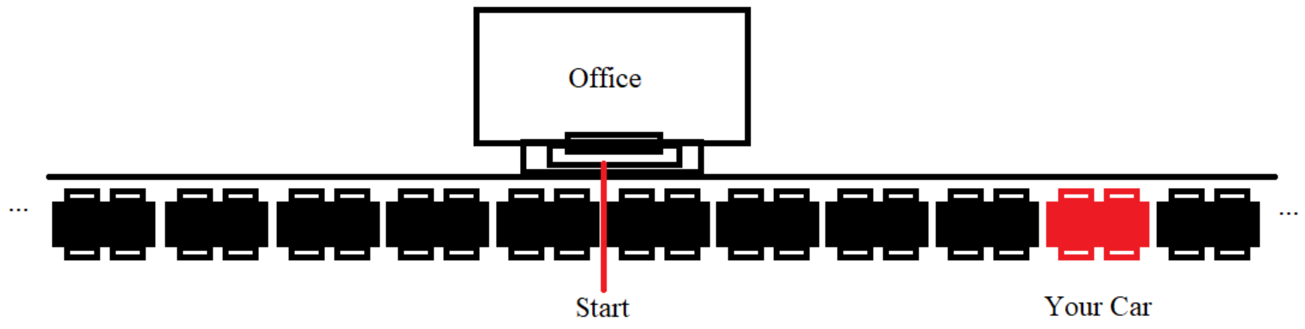
6. **Car Finding** [5 pts]

Let's say you work in an office building, with street parking in front of it. One day, you leave work and cannot remember where on the street you parked your car. Unfortunately, it is a foggy day and you can only see each car on the street when you are next to it. How should you find your car?

Specifics:

- You can only look at one car at a time, as you pass it.
- The cars are parked in single file along the 1D street.
- You do not know which direction your car is in.
- As far as you know, the parked cars go off to infinity in both directions.

An example image:



For the following questions, assume that your car is located exactly n cars away from your starting position. When asked how many cars you will pass, give the answer using big-O notation because we are concerned with the complexity of the different car finding algorithms.

(a) What is the minimum number of cars you would have to pass before reaching your car (assuming that your algorithm was 'perfect' and took you straight to your car)?

(b) What if you just tried walking in one direction? What is the best case? What is the worst case? Will this method always work?

(c) What happens if you check 1 car to each side, then 2 cars to each side, then 3 cars to each side, etc? How many cars will you pass before reaching your car?

(d) Describe a faster algorithm that guarantees you will always reach your car. (Hint: think about Problem 5 - is there a way to adapt that strategy for this new problem?)

(e) Prove that your algorithm is correct (i.e., briefly explain why your algorithm will always find your car).

(f) Derive the runtime of your algorithm, i.e., how many cars will you pass before reaching your car using your algorithm?

7. **Dragon Hunting** [4 pts]

You are a knight defending a group of small farms spread across the country side. A very hungry dragon has arrived, and has started eating the animals at the farms. The dragon flies to the largest uneaten farm (by total weight), eats all of the animals there, burns down the farm, and then moves to the next largest remaining farm. Both you and the dragon know the total weight of animals at each farm.

Unfortunately, you are not aware of where the dragon is right now, and the dragon flies faster than your horse can gallop. Assuming that it takes the same time to travel between any two farms, in what order should you visit the farms to catch the dragon while minimizing destruction?

Specifics:

- You have a list of n farms to protect, sorted from largest to smallest.
- The dragon has already burned down the first k farms and will start moving towards the $(k + 1)$st farm as you begin the hunt.
- You do not know the value of k.
- It takes a constant time to move between any two farms.
- The dragon will move slightly faster than you, but can only burn down 1 farm in the time it takes you to travel between farms.
- You will catch the dragon if, at any iteration, you arrive at the farm the dragon is currently attacking.

(a) Describe your algorithm for catching the dragon in a way that minimizes the damage *in the worst case.*

(b) What is the worst case damage of your algorithm, i.e., how many **more** farms can the dragon destroy before you catch it?