Problem Set 1

Instructions: This problem set is to be completed **INDIVIDUALLY**. Your solutions can be typed (preferred) or hand-written, and submitted as a PDF. You should strive for your solutions to be concise. Convoluted or obtuse answers will not receive maximum credit, even if the answer given is correct. You must cite all sources referenced in your solutions that are not on the course website. Submit your solution (1 per group) to `submit.cs.usna.edu` with the following command:
submit -c=SI335 -p=ps1 x.pdf
where x is your pdf filename.
Name:
GEORGE PRIELIPP (265112)

Problem 1-1. Big-O Fun

(**A good answer is:** For each part, a correct, yet not lengthy, proof using the knoweldge you have about Big-O.)

(a)) Explain why the statement: "The running time of the algorithm A is at least $O(n^2)$," has little meaning.

(b) Is $2^{n+1} \in O(2^n)$? Explain your answer.

(c) Is $2^{2n} \in O(2^n)$? Explain your answer.

a) The statement has little meaning because big-O notation is used for giving the upper bounds of a function. Furthermore, there are more sets of functions above $O(n^2)$ than below. So, the statement doesn't narrow down the bounds of the function at all.

b) $\underline{2^{n+1} \in O(2^n)}$

$c = 2 \quad , \quad n_0 = 0$

$2^{n+1} \le c \cdot 2^n \quad \forall n \ge n_0$

$2^{n+1} \le 2 \cdot 2^n \quad \forall n \ge 0$

$2^{n+1} \le 2^{n+1} \quad \forall n \ge 0 \quad \checkmark$ true

$\boxed{\text{Yes. See above.}}$

c) $\underline{2^{2n} \in O(2^n)}$

$2^{2n} \le c \cdot 2^n \quad \forall n \ge n_0$

$(2^n)^{\cancel{2}} \le c \cdot \cancel{2^n}$

$2^n \le c \quad \times$ false

$\boxed{\text{No. See above}}$

Problem 1-2. I'm thinking of a number...

You are at a STEM carnival, and you and your friends decide to play a game from one of the booths. The teller of the game is thinking of a positive integer number. Your goal is to find the integer k that the the teller has chosen. To play the game, you ask the teller a positive integer number and he will tell you whether his number k is higher, lower or is the correct number, and he will do so in constant time. Your friends, not being computer scientists, will find k in $\Theta(k)$ time. In order beat your friends, devise an algorithm to find k and prove that it is faster than your friend.

(**A good answer is:** Pseudocode for an algorithm to solve this problem, supported by a written description of how your algorithm works. To prove speed, A runtime analysis of your pseudocode that should result in a Big-O/Θ, or Ω runtime.)

```
def findK():
  lb = 1
  ub = 2
  While ask(ub) == higher:
      lb = ub
      ub = ub * 2
  K = ⌊(ub + lb) ÷ 2⌋
  while ask(K) != correct:
    if ask(k) == higher:
        lb = K
    else:
        ub = K
    K = ⌊(ub + lb) ÷ 2⌋
  return K
```

My algorithm relies on binary search However, An upper bound is not given. So, first the algorithm finds an upper and lower bound. Then it performs a binary search until K is found.

Big-Θ

1st while loop (finding upper bound)

- constant amount of work, c_1 until $ub \geq K$
- ub at iteration i:
  $ub_i = 2^{i+1}$ because $ub_0 = 2$, and grows by 2 for each iteration after. Eg $ub_1 = 4$, $ub_2 = 8$ $\therefore ub_i = 2^{i+1}$
- How many iterations (what value of i) until $ub \geq K$?

  $2^{i+1} \geq k$

$\log(2^{i+1} \geq K)$

$i+1 \geq \log K$

$\overset{-1}{i} \geq \log K \overset{-1}{-1}$

$i \geq \log K - \log 2$

$i \geq \log \frac{K}{2}$

So, at a minimum:

$\boxed{i = \log \frac{K}{2}}$

- Using $i = \log\frac{K}{2}$, the total amount of work is $\sum_0^i c_1 = c_1 \log\frac{K}{2} \in \boxed{\Theta(\log K)}$

2nd While loop

- Let $n = ub - lb$ = the total search space size. The worst case is when K is near ub or lb.
- At each iteration "K" $= \lfloor \frac{ub+lb}{2} \rfloor$ = midpoint of the search space.

And, the bounds change according to what the teller says. If "K" < K, then the lower half of the search space is also less than K, and lb can be moved up to "K". Otherwise, "K" ≥ K, meaning the upper half can be discarded, and ub can be moved down to "K". So, the search space is halved each iteration.

In math:
i = iteration, x = last iteration

i	0	1	2	3	...	x
n	n	$\frac{n}{2}$	$\frac{n}{4}$	$\frac{n}{8}$	...	$\frac{n}{2^x}$

So, the total work $= \sum_{i=0}^{x} c_2$

But the loop goes at a worst case until the final element looked at is K. In math: $\frac{n}{2^x} = 1$

At this point, solving for $x \rightarrow$ $x = \log n$

Solving for big-Θ of the total work:

$$\sum_{i=0}^{\log n} c_2 = c_2 \log n$$

Putting n in terms of K:

$$ub = 2^{i+1} = 2^{\log \frac{K}{2} + 1} = 2^{\log K \cancel{-1+1}} = K$$

$$lb = 2^{i} = 2^{\log \frac{K}{2}} = \frac{K}{2}$$

$$n = K - \frac{K}{2} = K(1-\tfrac{1}{2}) = \frac{K}{2}$$

$$\boxed{c_2 \log \frac{K}{2} \in \Theta(\log K)}$$

Combining the two parts:

$$\Theta(\log K) + \Theta(\log K) = \Theta(\log K)$$

$\boxed{\text{Therefore, findK() } \in \Theta(\log K)\text{, which is faster than } \Theta(K).}$