Name: Peter Murphy

Spring 2021	CS5008 March 18

Assignment 8, Primer on proofs

We want to continue to get more comfortable with the mathematical notation used in Algorithms. In problem 1, you are going to write *in plain English* what the expression is and then solve the statement.

Problem 1 - Quantifiers

Write the following statements as English sentences, then decide whether those statements are true if x and y can be any integers. When deciding if x and y can be any integers, prove your claim with a convincing argument.

1. $\forall x \exists y : x + y = 0$

   For all x, there exists some y such that $x + y = 0$.

   Proof:

   Assume $x + y = 0$ is true. Therefore, $x = -y$ is also true.

   Case 1: $x > 0$
   $x = 1$
   $y = -x = -1$
   $1 + (-1) = 0$ ✓

   Case 2: $x < 0$
   $x = -1$
   $y = -x = 1$
   $-1 + 1 = 0$ ✓

   Case 3: $x = 0$
   $x = 0$
   $y = -x = 0$
   $0 + 0 = 0$ ✓

   True

2. $\exists y \forall x : x + y = x$

   There exists some y where every value of x satisfies $x + y = x$.

   Proof:

   Def → Identity property of addition: The sum of 0 and any number is that number.

   Therefore, $x + 0 = x$ is true for all x by the identity property of addition. → This statement is True ($y = 0$).

3. $\exists x \forall y : x + y = x$

   False. let $x = 1$, let $y = 1$.
   $x + y = x$
   $1 + 1 = 1$ → This proposition is False.

In problem 2 and problem 3, we want to solidify our understanding of Big-O notation. Remember, Big-O notation is about the growth of a function as n grows asymptotically large.

Problem 2 - Growth of Functions

Organize the following functions into six columns. Items in the same column should have the same asymptotic growth rates (they are big-O and big-Θ of each other). If a column is to the left of another column, all its growth rates should be slower than those of the column to its right.

n^2, $n!$, $n \log_2 n$, $3n$, $5n^2 + 3$, 2^n, 10000, $n \log_3 n$, 100, $100n$

$O(1)$	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(2^n)$	$O(n!)$
10000 100		$3n$ $100n$	$n \log_2 n$ $n \log_3 n$	n^2 $5n^2+3$	2^n	$n!$

Problem 3 - Function Growth Language

Match the following English explanations to the *best* corresponding Big-O function by drawing a line from the left to the right.

1. Constant time
2. Logarithmic time
3. Linear time
4. Quadratic time
5. Cubic time
6. Exponential time
7. Factorial time

$O(n^3)$
$O(1)$
$O(n)$
$O(log_2 n)$
$O(n^2)$
$O(n!)$
$O(2^n)$

def: $f(n) = O(g(n))$ means there are positive constants c and k | $0 \leq f(n) \leq cg(n)$ for all $n \geq k$. c and k must be fixed.

Problem 4 - Big-O

1. Using the definition of big-O, show $100n + 5 = O(2n)$.

$100n + 5 \leq c(2n)$

$5 \leq 2cn - 100n$

$5 \leq 2n(c - 50)$

$\frac{5}{2n} \leq c - 50$

$c \geq \frac{5}{2n} + 50$

for $n = k = 1$:

$c \geq \frac{5}{2(1)} + 50$

$\boxed{c \geq \frac{105}{2}}$

$100n + 5$ is $O(2n)$ because there exists a constant k (1) and a constant c ($\frac{105}{2}$) where $0 \leq f(n) \leq cg(n)$ is true for all $n \geq 1$.

2. Using the definition of big-O, show $n^3 + n^2 + n + 100 = O(n^3)$.

$n^3 + n^2 + n + 100 \leq cn^3$

$n^2 + n + 100 \leq cn^3 - n^3$

$n^2 + n + 100 \leq n^3(c - 1)$

$\frac{1}{n} + \frac{1}{n^2} + \frac{100}{n^3} \leq c - 1$

$c \geq \frac{1}{n} + \frac{1}{n^2} + \frac{100}{n^3} + 1$

for $n = k = 1$:

$c \geq \frac{1}{(1)} + \frac{1}{(1)^2} + \frac{100}{(1)^3} + 1 \rightarrow \boxed{c \geq 103}$

$0 \leq f(n) \leq cg(n)$ is true for all $n \geq k$ when $k = 1$ and $c \geq 103$

3. Using the definition of big-O, show $n^{99} + 10{,}000{,}000 = O(n^{99})$.

$n^{99} + 10{,}000{,}000 \leq cn^{99}$

$10{,}000{,}000 \leq n^{99}(c - 1)$

$\frac{10{,}000{,}000}{n^{99}} + 1 \leq c$

for $n = k = 1$:

$c \geq \frac{10{,}000{,}000}{(1)^{99}} + 1$

$\boxed{c \geq 10{,}000{,}001}$

$0 \leq f(n) \leq cg(n)$ is true for all $n \geq k$ when $k = 1$ and $c \geq 10{,}000{,}001$

Problem 4 - Searching

We will consider the problem of search in ordered and unordered arrays.

1. We are given an algorithm called *search* which can tell us *true* or *false* in one step per search query if we have found our desired element in an unordered array of length 2048. How many steps does it take in the worse possible case to search for a given element in the unordered array?

   worst-case is when the desired element is the very last entry. in this case, the algorithm would take 2048 steps.

2. Describe a *fasterSearch* algorithm to search for an element in an **ordered array**. In your explanation, include the time complexity using Big-O notation and draw or otherwise explain clearly why this algorithm is able to run faster.

   Since the array is ordered, we can tell if the element is greater than or less than a given element in the array. Because of this, binary search can be used to "prune" the area we are searching. this algorithm works by finding a midpoint in the array and asking if the desired element is > or < the midpoint. Depending on the result, the algorithm discards the half that does not contain the element and calculates a new midpoint in the half that does contain the element. This process is repeated until the element is found.

3. How many steps does your fasterSearch algorithm (from the previous part) take to find an element in an ordered array of length 256 in the worse-case? Show the math to support your claim

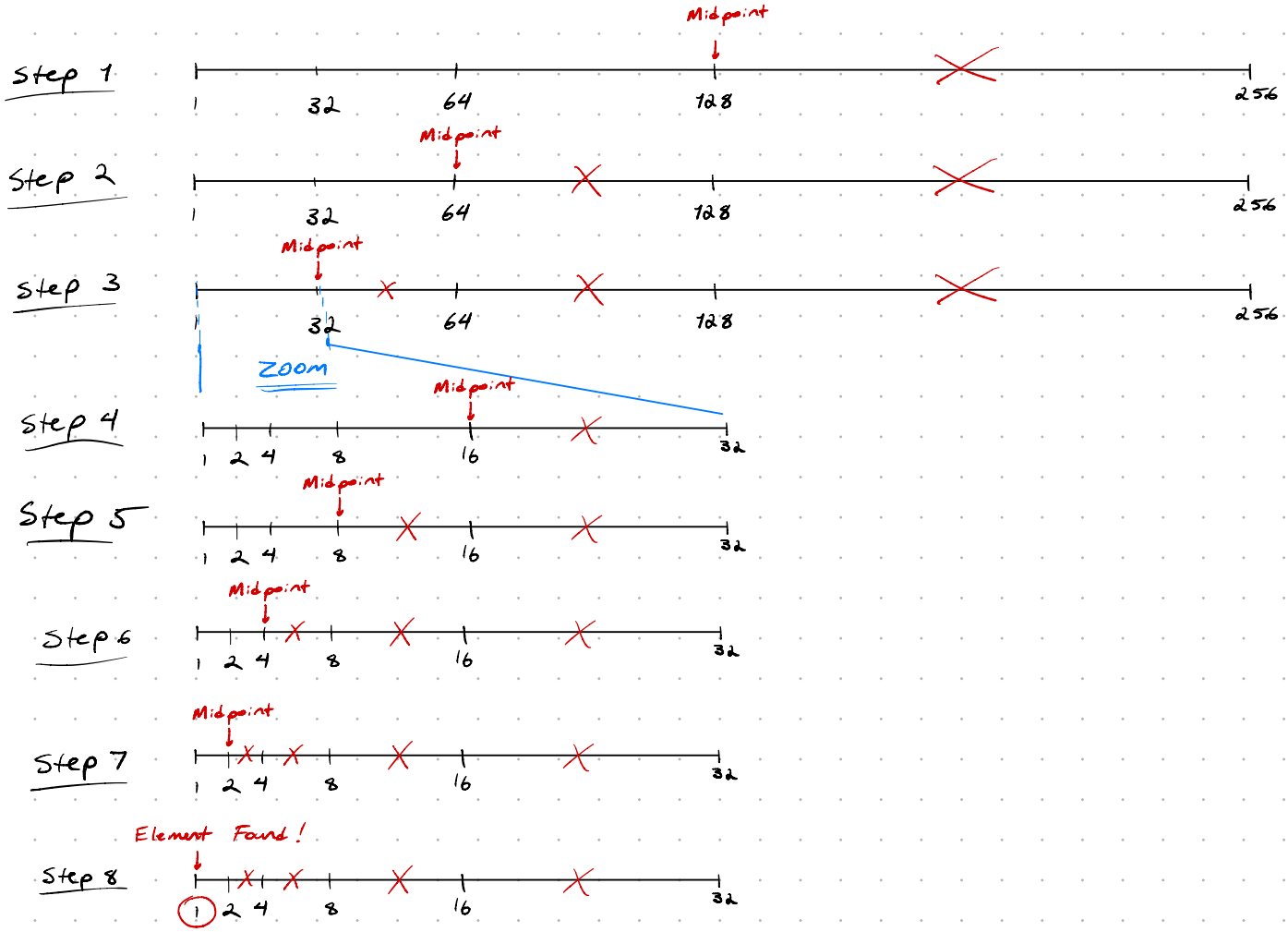
   Binary search is $O(\log_2 n)$. We can use this to calculate how many steps are needed in the worst case. (ex: looking for the first element).

   $\log_2 256 = x$ (x = num steps)

   $2^x = 256$

   $\boxed{x = 8}$ It will take 8 steps in the worst-case.

   See below for a visual explanation.


Step 1
Midpoint
1
32
64
128
256
Step 2
Midpoint
1
32
64
128
256
Step 3
Midpoint
1
32
64
128
256
Zoom
Step 4
Midpoint
1
2
4
8
16
32
Step 5
Midpoint
1
2
4
8
16
32
Step 6
Midpoint
1
2
4
8
16
32
Step 7
Midpoint
1
2
4
8
16
32
Step 8
Element Found!
1
2
4
8
16
32

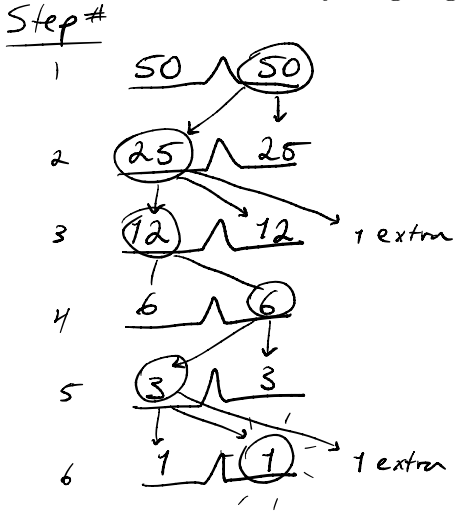
Problem 5 - Another Search Analysis

Imagine it is your lucky day, and you are given 100 golden coins. Unfortunately 99 of the gold coins are fake. The fake gold coins all weigh 1 oz. but the 1 real gold weighs 1.0000000001 oz. You are also given one balancing scale that can precisely weigh each of the two sides. If one side is heavier than the other side, you will see the scale tip.

1. Describe an algorithm for finding the real coin. You must also include the algorithm the time complexity. *Hint* Think carefully–or do this experiment with a roommate and think about how many ways you can prune the maximum amount of fake coins using your scale.

- Put 50 coins on each side. The side that tips has the coin, so discard the 50 coins on the other side.
- Repeat the above step by splitting the remaining 50 into 2 piles of 25. Keep the 25 that are weighed down and keep the remaining 25.
- Divide the 25 into 2 groups of 12 and keep the extra coin off the scale (the piles must have the same # of coins).
  If the piles are perfectly balanced, the coin you removed is real.
  If the scales tip, discard the lighter pile and the extra coin.
- Repeat the above steps until there are 3 coins left. Put 2 coins on the scale and hold on to the third coin. If the scales are balanced, take the extra coin. If not, take the heaviest coin.
- This algorithm is $O(\log n)$ since half the coins are removed each step.

2. How many weighing must you do to find the real coin given your algorithm?



6 steps