Name: Jing Luo

CS5008
Spring 2021 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. $\forall x \exists y : x + y = 0$

```
True.

P(0):
  if y = 0
    x+y = 0+y = 0+0=0      ->Ture

 P(n+1):
  if y = -(n+1)
    x+y = n+1-(n+1) = 0    ->Ture
```

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

```
True.

   if y = 0:
   x+y = x + 0 = x
                  ->True
```

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

```
False.

P(0):
     if y = 0:
      x + y = x +0 = x           ->Ture
P(n+1) :
     if y = n+1:
       x+y = x+n+1 !=x           ->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

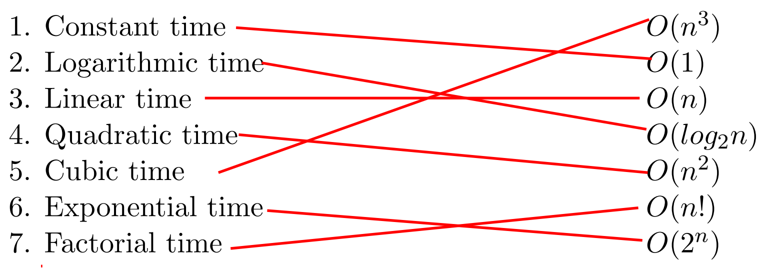
Organize the following functions into six columns. Items in the same column should have the same asymptotic growth rates (they are big-O and big-Θ of each other). If a column is to the left of another column, all its growth rates should be slower than those of the column to its right.

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

```
100,10000;
3n,100n;
n log3 n, n log2 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.



Problem 4 - Big-O

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

lim_{n->+}(100n+5)/(2n)

=lim_{n->+}(50+5/2n)

->50， a nonzero constant.

so 100n+5 = O(2n)

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

lim_{n->+}(n^3+n^2+n+100)/(n^3)

=lim_{n->+}(1+1/n+1/n^2+100/n^3)

->1 ， a nonzero constant.

so n^3+n^2+n+100 = O(n^3)

3. Using the definition of big-O, show $n^{99} + 10000000 = O(n^{99})$).

lim_{n->+}(n^99+10000000)/(n^99)

=lim_{n->+}(1+10000000/n^99)

->1 ， a nonzero constant.

so n^99+10000000 = O(n^99)

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?

O(n)=2048

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.

```
An ordered array(array[i]<array[i+1]).

To search value "A":

if searched_array[i] > A, then go on  searching array[0 : i-1].
   And if searched_array[j] < A, then go on searching array[ j :  i-1].
   "loop" until array[h] == A.
It is O(n) and range form "1" to "n" so it  can run faster.
```

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

```
It could be 256 steps.

for the array[256]:(1,2,3,4······, 253, 254, 255, 256)
   if the element we are going to find is 129.  the worse-case may be:
1,2 times:               1    -  256
3,4 times:               2    -  255
5,6 times:               3    -   254
  ······                   ······
253,254 times:       127 - 130
255,256 times:        128 - 129  (we got the 129, it  takes 256 times)
```

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.

1. Half of the coins in each side every times. the heavier side will go on and will be splited into two half . if "half" != "half", Keep go on. When we have reminder "1" and if "half" = "half",then the reminder is the real coin.
'50'vs'50'-->'25'vs'25'-->'12'vs'12'……1
-->'6'vs'6'-->'3'vs'3'-->'1'vs'1'……1
2. 25vs25, 25vs25, find the heavier.
6vs6, 6vs6……1, find the heavier of the last heavier.（Maybe you could get the real coin if all "6" is equal to each other）
3vs3, find the heavier of the last heavier.
1vs1……1. you get the real coin.

3. 33vs33vs33……1 if all "33" is equal to each other， the reminder 1 is the real. Else, find the heavier(need one weighing).
11vs11vs11, find the heavier(need one weighing). .
5vs5……1 if all "5" is equal to each other， the reminder 1 is the real. Else, find the heavier.
2vs2……1 if all "2" is equal to each other， the reminder 1 is the real. Else, find the heavier.
1vs1, the heavier is the real.

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

1. The first one: N = 3 or 6.

2. The second one: N = 4 or 6.

3. The third one: N = 2 or 3 or 4 or 5.

1 case1: if a"33" == b"33" == c"33",(two weifhing),the reminder 1 will be the real coin(N=2).

case2: else: we get the heavier.

if a'33' == b'33' , than we get the heavier c'33'.
2. it is same to "1 case2".
3.if "5" = "5", the reminder 1 will be the real coin(N=3).
else: we get the heavier.
4. if "2" = "2", the reminder 1 will be the real coin(N=4).
else: we get the heavier.
5. 1 VS 1 ,the heavier is the real coin(N=5).