Name: Yong Shi

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

Statement: when given any x, there exist y, that can make x+y=0.

True.

Proof: x+y=0 -> y=-x. So given any x, there exist -x can assign to y, can make x+y=0. So it is True.

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

Statement: there exist y, when given any x, that can make x+y=x.

True.

Proof: x+y=x -> y=x-x -> y=0. So there exist y=0, for any x, that can make x+y=x. So it is True.

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

Statement: there exist x, when given any y, that can make x+y=x.

False.

Proof: x+y=x -> y=x-x -> y=0. So only when y=0, can make x+y=x. So it 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.

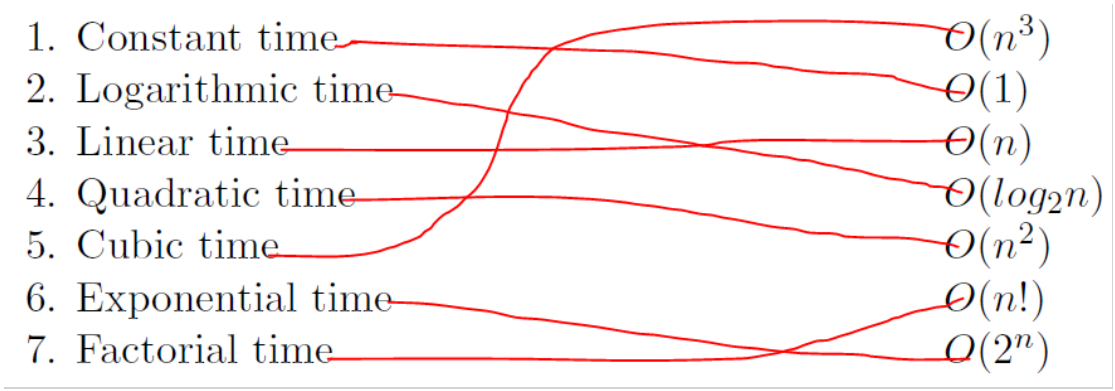
n^2, $n!$, $n \log_2 n$, $3n$, $5n^2 + 3$, 2^n, 10000, $n \log_3 n$, 100, $100n$

Slower -> growth faster:

10000 100	3n 100n	n log2 n n log3 n	n2 5n2+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 $100n + 5 = O(2n)$.

When n -> ∞， 100n -> n and 100n+5 -> n;

When n -> ∞， 2n -> n also.

So 100n+5=O(2n)

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

when n!=0: n3+n2+n+100=n3(1+1/n+1/n2+100/n3)

When n -> ∞， 1/n -> 0; 1/n2 ->0, 100/n3 ->0.

So n3(1+1/n+1/n2+100/n3) -> n3(1+0+0+0) -> n3.
So n3+n2+n+100=O(n3)

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

When n -> ∞， n99+10000000 -> n99.

So n99+10000000=O(n99).

Problem 4 - Searching

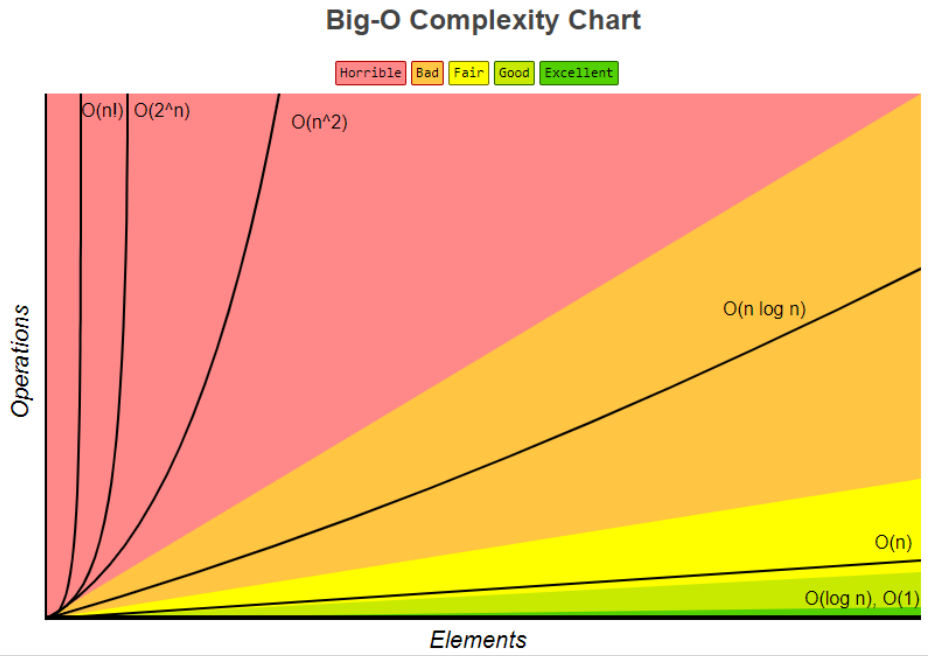
We will consider the problem of search in ordered and unordered arrays.

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?

So this is the linear search, the bigO as O(n), so the worse case is 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.

So this is as the binary search, the bigO as O(log n). From below chart, can know O(log n) is faster then the linear search with O(n).



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

1 time	2	3	4	5	6	7	8	9	...
256/2	128/2	64/2	32/2	16/2	8/2	4/2	2/2		
128	64	32	16	8	4	2	1		

So from above form calculation for the binary search, 8 times can get the value for the worse case. This is also can be calculated thru bigO, that log2(256)=8.

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.

First to my mind is the binary search algorithm. First time 50 – 50 coins put on each side; 2nd time 25 – 25 coins put on each side; … as below form, in worse case after 6 times can get the real coin.

1 time	2	3	4	5	6	7	8	9	…
100/2	50/2	25/2	12/2	6/2	3/2				
50	25	12	6	3	1				

Then how about try divide to 3 groups each time? As below form, worse case after 4 times can get it.

1 time	2	3	4	5	6	7	8	9	…
100/3	33/3	11/3	3/3						
33	11	3	1						

Then move on to divide to 4 groups each time, worse case need after 6 times.

1 time*2	2*2	3*2	4*2	5*2	6*2	7*2	8*2	9*2	…
100/4	25/4	6/4							
25	6	1							

So also can try divided by 5, or n. Then the algorithm is as below form. And also for the total times we can know is the 1*n/2, or 2*n/2, or …

To use 5 as example: 5^3>100, so n=3, worse case need 3*5/2 = 6 times.

1*n/2 time	2*n/2	3*n/2	4*n/2	5*n/2	6*n/2	7	8	9	…
100/n	100/n2	100/n3	100/n4	100/n5	100/n6				
100/n	100/n2	100/n3	100/n4	100/n5	100/n6				

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

So my conclusion is divide to 3 groups, 33-33-34, put 33 and 33 on the balance scale, if balance, move on to the 34; then 11-11-12, put 11 and 11 on the balance scale, if balance, move on to the 12; then 4-4-4; then move on to the left 4, put as 1-1-2; then move to the last 2. Total 5 times.