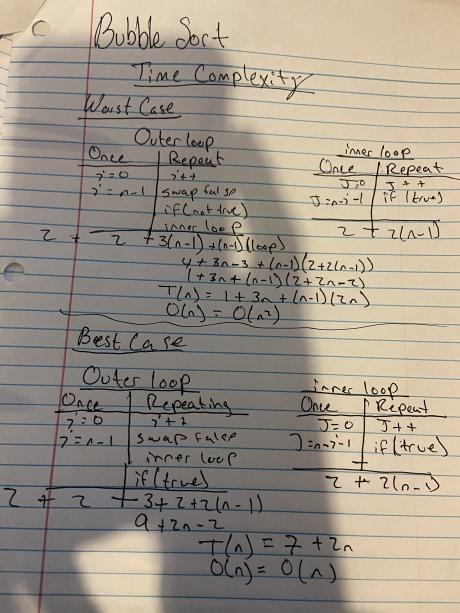
Time Complexities for the following Algorithms

Bubble, Selection, Insertion Sort and Binary Search



**Bubble Sort**

**Best Case**

The time complexity ends up being 7 + 2n. This is because the outer loop and the outside statements declare 4 things once through out the code. Then inside the repeating steps for the outer loop, it repeats a total of 3 \* (n – 1) + the inner loop being (n – 1) \*( loop). So far we have:

4 + 3 \* (n – 1) + (n – 1) \* (loop)

Then for the inside loop it declares 2 things once and ends up repeating 2 things inside time (n -1)

So we have after everything: 4 + 3 \* (n – 1) + (n – 1) \* (2 + 2(n – 1))

Since it is the best case the inner loop if statement did not meet the condition, that means the outer loop is going through go through a break statement cutting down the time to the following:

4 + 3 + 2 + 2 (n -1) -> 9 + 2n – 2 -> 7 + 2n

So Best case: T(n) = 7 + 2n

O(n) = O(n)

**Average & Worst Case**

The time complexity ends up being 7 + 2n. This is because the outer loop and the outside statements declare 4 things once through out the code. Then inside the repeating steps for the outer loop, it repeats a total of 3 \* (n – 1) + the inner loop being (n – 1) \*( loop). So far we have:

4 + 3 \* (n – 1) + (n – 1) \* (loop)

Then for the inside loop it declares 2 things once and ends up repeating 2 things inside time (n -1)

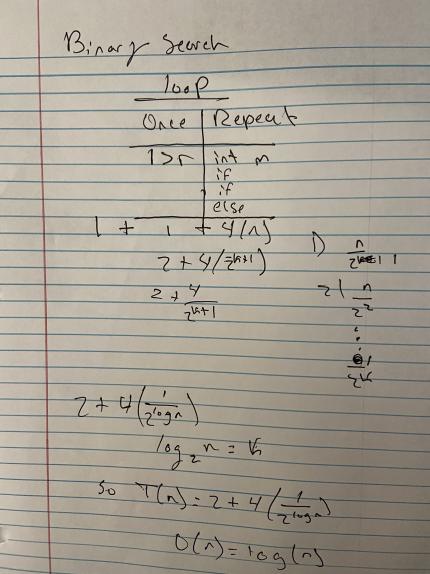
So we have after everything: 4 + 3 \* (n – 1) + (n – 1) \* (2 + 2(n – 1))

This time it is the worst case so the if statement in the inner loop will true hence repeating n times on the outer loop too so: 4 + 3 \* (n – 1) + (n – 1) \* (2 + 2(n – 1)) -> 1 + 3n + (n -1) \* ( 2 + 2n -2)

-> 1 + 3n + (n – 1)\*2n

So, T(n) = 1 + 3n + (n – 1) \* 2n

O(n) = O(n^2)

\

**Binary Search**

**Best Case**

Is 1 since that means the element we are looking for is in the middle, so it stops right there.

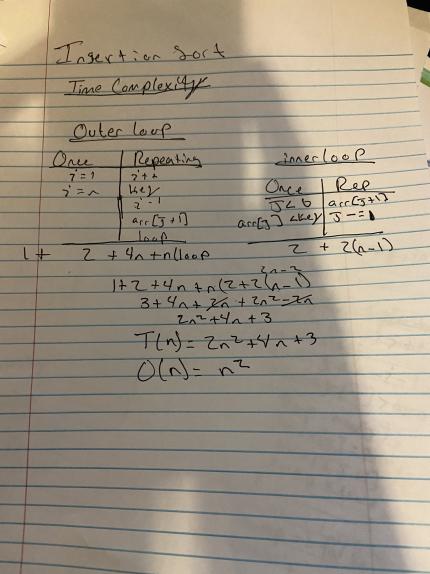
**Average and Worst Case**

It declares a variable on the outside so we have +1, then inside the loop we only do once the breaking condition which is l > r so our equation looks so far 1 + 1 +…, then we have the repeating which may go through each if so it can repeat 4 times. So we have 2 + 4(n). Since every time we are dividing our array by 2. It is the same as dividing 2^k, where k indicates how many times you have divided the array. So our equation is looking like: 2 + 4\*( 2^ -(k + 1)), the + 1 indicates the breaking condition. So we have log(base2) (n) = k + 1, which gives us 2 + 4( 1 / 2 ^ logn),

So,

T(n) = 2 + 4\* ( 1 / 2 ^ logn)

O(n) = logn



**Insertion Sort**

**Best, Average & Worst Case**

It initializes 1 thing outside the loop so 1++, then we get to the outer loop where it does two things once and it repeats a total of 4 things and the loop. So our equation looks like this: 1 + 2 + 4n + n(loop)

Then our inner declares two things once and repeats two things a total (n-1)

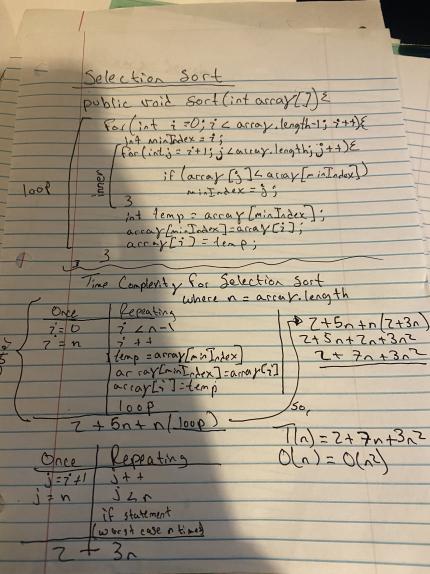
So combining out equations we get, the following:

1 + 2 + 4n + n(2 + 2(n – 1)) simplifying we get

2n ^2 +4n + 3

So, T(n) = 2n ^2 +4n + 3

O(n) = O(n ^ 2)

****

**Selection Sort**

**Best, Average & Worst Case**

It starts with an outer loop, where it two things happen once in the entire code which then repeats 5 things and a inner loop, so our equation up to this point looks like:

2 + 5n + n (loop)

Then the inner loop has two things were it only happens once, and repeats at the most 3 things

So the inner loop equation looks like: 2 + 3n

So combining it we have, 2 + 5n + n \*( 2 + 3n)

Then simplifying we get 2 + 7n + 3n ^2

So, T(n) = 2 + 7n + 3n ^ 2

And O(n) = O(n ^ 2)