Name: Eddie Lee

Here are some basic rules for calculating the Big O for some $T(n)$ or an algorithm.

1. Only the highest degree of n matters. For example

$$T(n) = n^3 + 5n^2 + 10^7 \rightarrow O(n^3)$$

since once n becomes super-massively huge, the other terms just stop mattering.

2. Constant factors don't matter. $T(n) = 500n$ and $T(n) = 0.005n$ both $O(n)$. Again, as n becomes bigger, these constants stop mattering; what matters is the rate of growth. Example:

$$T(n) = 50n^3 - 2n^2 + 400 \rightarrow O(n^3)$$

3. Counting the number of nested loops usually works.

You can turn in this assignment physically to a TA or me. You can also scan and upload your answers.

 _____ 0 points

For each of the following $T(n)$, write the corresponding Big O time complexity. Some series may require research.

1. (2 points) $T(n) = n^2 + 3n + 2$

   1. O(n^2)

2. (2 points) $T(n) = (n^2 + n)(n^2 + \frac{\pi}{2})$

   2. O(n^4)

3. (2 points) $T(n) = 1 + 2 + 3 + \ldots + n - 1 + n$

   3. O(n^2)

4. (2 points) $T(n) = 1^2 + 2^2 + 3^2 + \ldots + (n-1)^2 + n^2$

   4. O(n^3)

5. (2 points) $T(n) = 10$

   5. O(1)

6. (2 points) $T(n) = 10^{100}$

   6. O(1)

7. (2 points) $T(n) = n + \log n$

   7. O(n)

8. (2 points) $T(n) = 12\log(n) + \frac{n}{2} - 400$

   8. O(n)

9. (2 points) $T(n) = (n+1) \cdot \log(n) - n$

   9. O(n log n)

10. (2 points) $T(n) = \frac{n^4+3n^2+2n}{n}$

    10. O(n^3)

_____ 20 points

11. (4 points) What is the time complexity to get an item from a specific index in an ArrayList?

Using get() in an ArrayList always has a time complexity of O(1), or constant time

12. (3 points) What is the time complexity remove an item in the middle of an ArrayList?

Using remove() is O(n), or linear time

13. (3 points) Why?

Because in the worst case, removing the element will require shifting the entire list a position to the left

14. (3 points) What is the **average** time complexity to add an item to the end of an ArrayList?

The average time complexity is O(1)

15. (3 points) What is the **worst case** time complexity to add an item to the end of an ArrayList? What if you have to or don't have to reallocate?

The worst case time complexity is O(n). This happens when the ArrayList needs to be reallocated, and this happens by allocating new memory and copying all the elements to the new list. If reallocation isn't necessary, then the time complexity averages out to O(1)

16. (4 points) Taking this all into account, what situations would an ArrayList be the appropriate data structure for storing your data?

ArrayLists have the benefits of being dynamic, being ordered, and having constant time to access elements. These situations would include storing a list of changeable items like names, using a dynamic array that changes based on its elements, and for lists that are important in order like queues.

```
public static int[] allEvensUnder(int limit){
        if (limit <= 0){
                return new int[0];
        }
        if (limit < 2){
                return new int[1];
        }
        int[] vals =  new int[(limit+1)/2];
        for(int i  = 0;  i <(limit+ 1)/2  ; i++ ) {
                vals[i] = i*2;
        }
        return vals;
}
```

17. (5 points) What is the **time** complexity of the above algorithm?

It is O(n)

18. (5 points) What is the **space** complexity of the above algorithm? In other words, how much space is used up as a function if the input size? Think about it, we didn't cover this in the videos.

The space complexity is also O(n) because the size of the array is proportional to n

```
/*
*  https://rosettacode.org/wiki/Sorting_algorithms/Insertion_sort#Java
*/
public static void insertSort(int[] A){
        for(int i = 1; i < A.length; i++){
                int value = A[i];
                int j = i - 1;
                while(j >= 0 && A[j] > value){
                        A[j + 1] = A[j];
                        j = j - 1;
                }
                A[j + 1] = value;
        }
}
```

19. (10 points) What is the time complexity of the above algorithm?

The time complexity is O(n^2) because of the nested loops

`bogosort` attempts to sort a list by shuffling the items in the list. If the list is unsorted after shuffling, we continue shuffling the list and checking until it is finally sorted.

20. (5 points) What is the worst case run time for `bogosort`?

It would be O(infinity)

21. (5 points) Why?

Because it's a randomized sorting algorithm, the shuffle could theoretically never find the correct sort

22. (5 points) What is the average case run time for `bogosort` (Hint: think about a deck of cards)?

The average case run time is O(n * n!)

23. (5 points) Why?

There are n! possible permutations of the elements, so the average outcome of sorting before the correct permutation is n, and with the time of O(n) to check each permutation would make it O(n * n!).

 _____ 20 points

24. (20 points) For each of the methods you wrote in Lab 2, figure out the time complexity of the method you wrote. To turn in this portion, attach a printout of the code and specify the time complexity of each.

We already did this, so it's okay to leave out here right?

_____ 20 points