Name: Annica Chiu

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. n^2

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

   2. n^4

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

   3. n

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

   4. n^2

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

   5. 1

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

   6. 1

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

   7. n

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

   8. n

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

   9. n

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

    10. n^3

_____ 20 points

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

It is O(1) because, to get an item from a specific index, Java adds the address of the ArrayList by the specific index to get the address of the item. Adding and accessing the item is O(1), so overall it is O(n).

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

It is O(n).

13. (3 points) Why?

To remove an item, the program would remove the item and move every item after to the index before it. If an item were removed from the middle of an ArrayList, ever item after the middle would have to move back. The same function would occur on half the items in the list. O(n/2) simplifies to O(n).

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

It is O(n) because adding to the end does not require the function to move any items, unless the ArrayList needs to reallocate, which is rare.

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 would be O(n), which when the ArrayList needs to be reallocated. This is because when the ArrayList needs more space, and the entire ArrayList needs to be copied into an array with more space. Each item would be copied, which would result in O(n).
If you do not need to reallocate, the time to add an item to the end of an ArrayList is O(1).

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

An ArrayList would be best for when I need to use indexes to get items, only be adding to the end, and rarely removing items.

```
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?

O(n)
Because the two if statements take O(1) and the for loop executes for every index in vals.
The loop adds even numbers from 0 to the limit to the vals array, which would take 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.

O(n)
Because the algorithm requires an array. The array will store n items.

```
/*
*  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?

O(n^2) because in the worst case scenario, the while loop would have to execute every time the for loop executes. If the array to be passed in was numbers in descending order, the while loop would have to iterate on every value. Since the while loop is within a for loop, which is already iterating on every value, the algorithm is O(n^2).

`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`?

The worst case would be the program never ends.

21. (5 points) Why?

Because bogosort shuffles randomly every time, there is a possibility that the list is never sorted correctly for all of eternity. This is unlikely, but it is possible.

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

The average case run time for bogosort would be O(n!).

23. (5 points) Why?

This is because the total possible arrangements of n items is calculated by 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.

See next page...

_____ 20 points