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)

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

   4. O(n^2)

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?

O(1) - constant

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

O(n) - linear

13. (3 points) Why?

The list gets reordered after an item is removed. This requires a for loop to reorder whatever n items occur after the item removed.

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

O(1) - constant

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?

O(n) if you're adding just space (reallocate) in the arrayList each time you hit the end of the ArrayList. As an example you get to the end of the list and add just one additional spot.

However if you're reallocating to a new ArrayList only rarely such as doubling each time then it'll be a much lower time complexity.

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

ArrayList would be great to use in a variety of situations, even when you're increasing the list by doubling each time you hit the end, the time complexity isn't very large (more than constant less than linear). However, the best time to use an ArrayList would be when you know the length of the list you need.

 _____ 20 points

```
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) - linear

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.

probably linear

```
/*
*  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) or quadratic

_____ 20 points

`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`?

I guess worst case run time would be that it never ends. O(infinity?)

21. (5 points) Why?

There could be a case in which you continue having to shuffle the deck to get it into the right order without it becoming correct. There's a small probablility that each time you shuffle it, that it doesn't become sorted.

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

O(n * n!)

23. (5 points) Why?

The time it will take to get a sorted permutation would be n! as there are n! total permutations the order could be. Thus the total time would be the number of times it has to run times the time it takes to run each one. The number of times it would need to run would be this expectation of n! and each one would take n time.

 _____ 20 points