Week 2 feedback and quiz answers

(read only after passing the programming assignment and the quiz)

You Are Doing Well!

Congratulations on completing Week 2 of Data Structures: Measuring and Optimizing Performance! To complete this week's quiz it requires having a critical understanding of Big O Notation. If you struggled with this week's quiz, that's okay! Big O is tricky for many students; it takes time and practice. Here are some helpful hints for conquering Big O:

1. Create a chart of some of the common orders of growth like O(1), O(N), O(n log n), etc., along with what they mean and coding examples for each of the orders. Refer back to this list every time you get confused or want to refresh your memory.
2. As you continue on your Computer Science education, try to look at the time complexity of your code and always try to optimize your code when possible!

If working with Efficiency Analysis and Benchmarking really interested you, explore computational complexity theories and scalability! If this week did not interest you, don't worry because next week you will be learning about a new topic: Abstraction, Interfaces, and Linked Lists.

Before you begin next week's material, here is a helpful hint:

If you find the programming assignments easy, go to the Discussion Forum and help out the people who have questions or are struggling. Teaching others is a great way to solidify the information yourself, so if you are able to explain a concept then that means you understand the concept! Similarly, if you are struggling, go to the Discussion Forum for help! Computer Science is a collaborative endeavor and there are other learners out there, mentors, and professors that will be happy to help!

We hope you're having fun and really understanding what you're learning, but it's always good to review both what you got right and what you got wrong, so the rest of the reading contains explanations and answers for this week's quiz questions. Correct answers to the quiz questions

are shown in **bold**. Incorrect answers are shown in *italics*. (Note that log_2(n) means "log base 2 of n").

Good luck and have fun learning!

Sincerely,

Our MOOC Team

Quiz Answers and Explanations

Correct answers are in **bold**. Incorrect answers are shown in *italics*.

1. Consider the function f(n) = 3nlog_2(n)+100log_2(n). What is the tightest Big-O class for f(n)?

- *O(n) (Incorrect)*

Notice that 3nlog(n) is the dominant term in f(n) and nlog(n) grows more quickly than n.

- **O(nlog(n)) (Correct)**

Correct! 3nlog(n) is the dominant term in f(n)

- *O(log(n)) (Incorrect)*

Notice that 3nlog(n) is the dominant term in f(n) and nlog(n) grows more quickly than log(n).

- *O(n^2) (Incorrect)*

3nlog(n) is the dominant term in f(n) and nlog(n) grows more slowly than n^2.

2. Does the following equality give the tightest Big-O equivalence class?

log_10(n)+25log_5(n)=O(log_2(n))

- **Yes (Correct)**

Correct! Logs can be converted into any other base by multiplying by a constant factor.

- *No (Incorrect)*

Remember that logs can be converted into any other base by multiplying by a constant factor.

3. Consider the following code:

```
int sumSome(int[] arr){
   int sum = 0;
   for (int i=0; i<arr.length;  i++) {
      for (int j=0; j<arr.length; j+=2) {
         if (arr[i] > arr[j])
            sum += arr[i];
```

```
            sum += arr[i];
      }
   }
   return sum;
}
```

What is the tightest Big-O running time for this code in terms of the length of the array, n?

- *O(n) (Incorrect)*

The running time of the inner loop must be multiplied by the number of times the outer loop runs because every time through the outer loop you complete the entire inner loop. The outer loop runs n times, while the inner loop runs n/2 times.

- **O(n^2) (Correct)**

Correct. The outer loop runs n times, while the inner loop runs n/2 times, and then you multiply them together to get (n^2)/2 or O(n^2)

- *O(log(n)) (Incorrect)*

The running time of the inner loop must be multiplied by the number of times the outer loop runs because every time through the outer loop you complete the entire inner loop. The outer loop runs n times, while the inner loop runs n/2 times.

- *O(nlog(n)) (Incorrect)*

The running time of the inner loop must be multiplied by the number of times the outer loop runs because every time through the outer loop you complete the entire inner loop. The outer loop runs n times, while the inner loop runs n/2 times.

4. Consider the following code:

```
int sumSome(int[] arr){
   int sum = 0;
   for (int i=0; i<arr.length;  i++) {
      for (int j=1; j<arr.length; j = j*2) {
         if (arr[i] > arr[j])
            sum += arr[i];
```

```
            sum += arr[i];
        }
    }
    return sum;
}
```

What is the tightest Big-O running time of this code in terms of the length of the array, n?

- *O(n^2) (Incorrect)*

This is an upper bound, but not the tightest upper bound.

- **O(nlog(n)) (Correct)**

Correct! The inner loop runs approximately log(n) times while the outer loop runs n times.

- *O(n) (Incorrect)*

This is not quite enough. This would be the answer if the inner loop ran in constant time, but it does not.

5. You have two programs, program A and program B, that do the same thing. Program A runs in O(n) (tightest bound) and program B runs in O(n2) (tightest bound) in the best, worst and average cases.

True or false: if you give both programs the same input, program A will always run faster than program B no matter what the input was.

- *True (Incorrect)*

Big-O only classifies what the running time will do as the inputs get large, so for small inputs, this might not necessarily be true.

- **False (Correct)**

Big-O only classifies what the running time will do as the inputs get large, so for small inputs, this might not necessarily be true.

6. You have two programs, program A and program B, that do the same thing. Both programs run in O(n) (tightest bound) in the best, worst and average cases.

True or false: If you give both programs the same input, you can be sure they will take the same amount of time to complete no matter what the input was.

- *True (Incorrect)*

Remember that Big-O notation describes only the shape of the function. There may be

constants that make one program run faster or slower than the other.

- **False (Correct)**

Big-O notation describes only the shape of the function. There may be constants that make one program run faster or slower than the other.

7. What is the tightest Big-O running time to calculate the Flesch readability score for the BasicDocument class, where n is the length of the document?

- *O(1) (Incorrect)*

It must pass through the whole document at least once.

- **O(n) (Correct)**

Correct! Even though it makes multiple passes through the document, each time is simply a linear pass through.

- *O(n^2) (Incorrect)*

Even though it makes multiple passes through the document, each time is simply a linear pass through.

- *O(n^3) (Incorrect)*

Even though it makes multiple passes through the document, each time is simply a linear pass through.

8. What is the tightest Big-O running time to calculate the Flesch readability score the first time in the EfficientDocument class, where n is the length of the document, assuming you include the time it takes to initialize the numSyllables, numWords, and numSentences variables?

- *O(1) (Incorrect)*

Remember to include the time it takes to initially count the number of words, sentences and syllables.

- **O(n) (Correct)**

Correct! The initialization step must pass through the whole document one time.

- *O(n^2) (Incorrect)*

The initialization step must pass through the whole document one time, but after that the number of sentences, words, and syllables can be accessed in constant time.

9. What is the tightest Big-O running time to calculate the Flesch readability score in the EfficientDocument class, where n is the length of the document, assuming you DO NOT include the time it takes to initialize the numSyllables, numWords, and numSentences variables?

- **O(1) (Correct)**

Correct! The initialization step must pass through the whole document one time, but after that the number of sentences, words, and syllables can be accessed in constant time.

- *O(n) (Incorrect)*

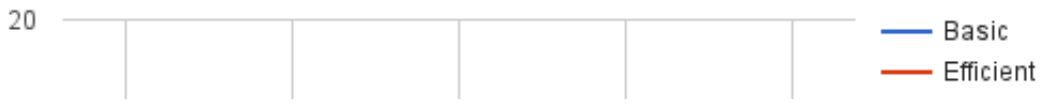
The initialization step must pass through the whole document one time, but after that the number of sentences, words, and syllables can be accessed in constant time.

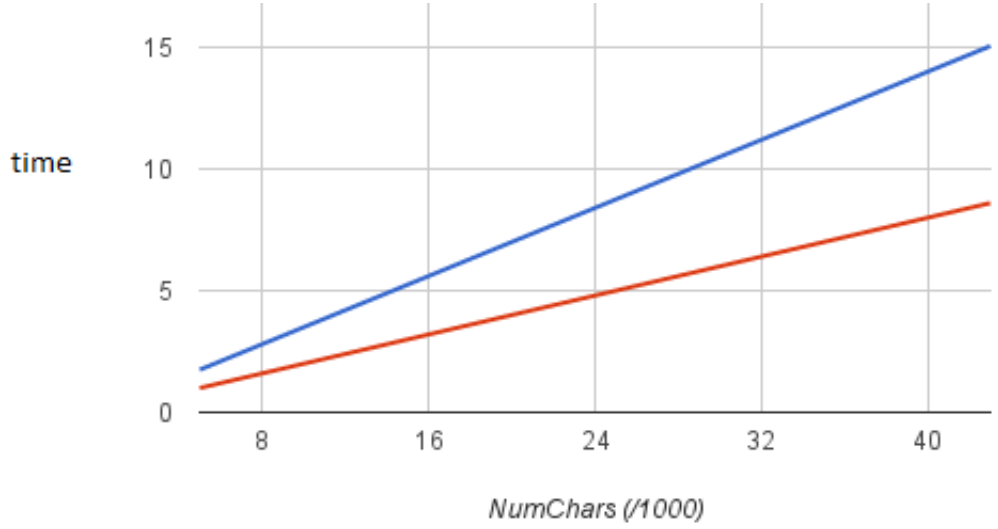
- *O(n^2) (Incorrect)*

The initialization step must pass through the whole document one time, but after that the number of sentences, words, and syllables can be accessed in constant time.

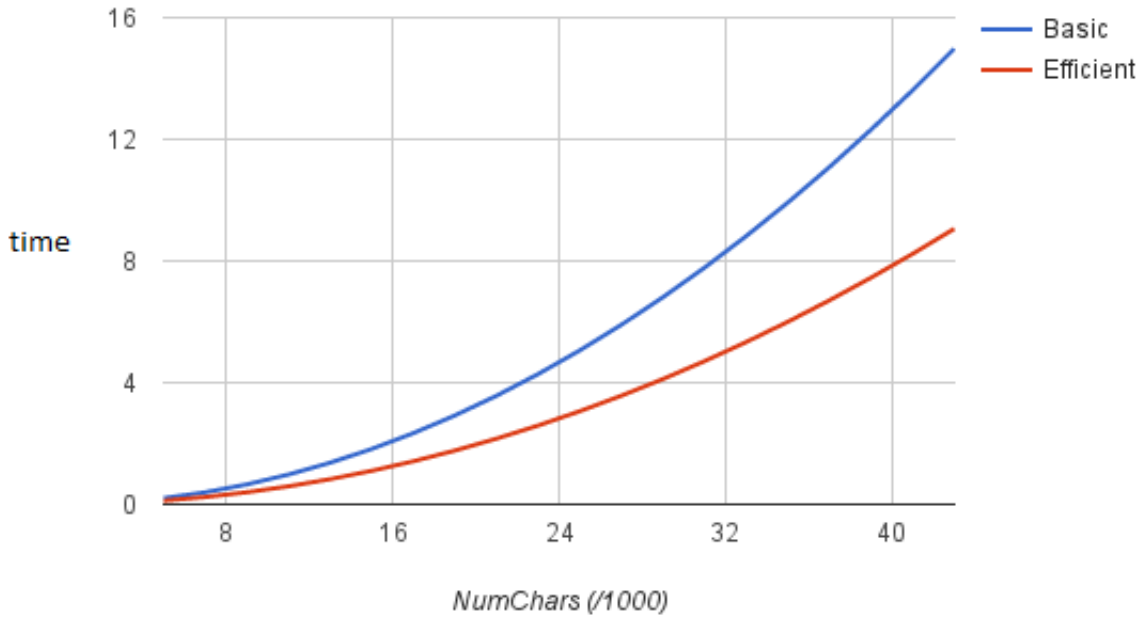
10. Which of the following graphs looks most like the graph you (should have) produced in part 2 of the programming assignment? We are looking for you to compare the shapes of the functions only--these graphs are much smoother than what you would have seen. Also, ignore the specific numbers, and concentrate on the shapes and the relationship between the two lines. Note that the correct answer assumes that you have correctly implemented your two Document classes and your DocumentBenchmarking class.

- **(Correct)** The image below is **correct**! Both have an approximate linear relationship with the number of characters, but BasicDocument has a bigger slope.

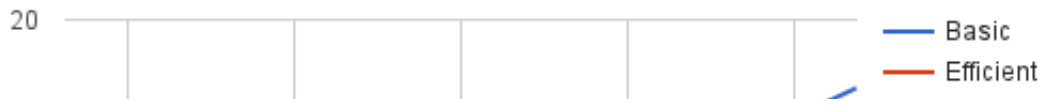


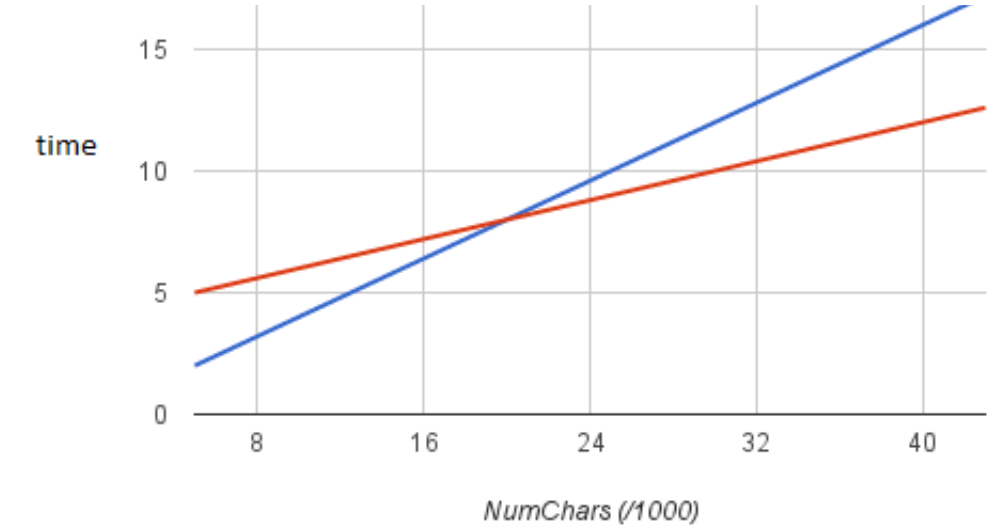


- *(Incorrect)* The image below is *incorrect.* This graph shows the wrong order of growth for both classes.



- *(Incorrect)* The image below is *incorrect*. It should not be the case that EfficientDocument is slower for small documents. It should be faster for all sizes of documents (or at least the same speed, approximately).





- *(Incorrect)* The image below is *incorrect*. Basic document is slower than efficient document, but it grows at the same order of growth. This graph shows it growing quadratically, while efficient document grows linearly.