Homework M6: Timing and Efficiency

Due Mar 14 at 11:59pm **Points** 100 **Questions** 25
Available until May 30 at 11:59pm **Time Limit** None

Instructions

Review the **Homework FAQ** page for details about submitting homework. In this homework, you will:

- estimate
- find the running time of an algorithm
- evaluate order of growth (efficiency, big o)
- write code with a specific order of growth

Notes

- When considering Big-O of code, be careful to consider the efficiency of methods invoked (sorting, contains, remove, etc.).
  - You can review the efficiency of a method in a class in the Java standard library (e.g., ArrayList, LinkedList) by looking at the class and method description on the API page.
  - Also review this week's announcements and practice problems, where some efficiencies are discussed.
  - If you aren't sure about a method's efficiency, post to the discussion board!
- **The biggest factors in efficiency are method calls and loops that are dependent on n.**
  - Nested loops that both loop n times give you O(n^2) code.
  - Invoking an O(n) method inside of an O(n) loop gives you O(n^2) code.

This quiz was locked May 30 at 11:59pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	4,877 minutes	98 out of 100

Score for this quiz: **98** out of 100
Submitted Mar 9 at 11:31pm
This attempt took 4,877 minutes.

Coding and Open-Ended Questions

Question 1

10 / 10 pts

Estimate an answer to the question below. Describe your reasoning and assumptions **and** your final numeric estimate.

How many new cell phones are sold each year in the U.S.?

- Pretend that you are in an interview and do not have access to the internet or your phone!
- You will not be graded on your answer. You will be graded on whether you describe your thought process and explain how you get your estimate.
- A single number with no explanation will receive 0 points.
- Numbers based on looking up the answer online will receive 0 points.
- Be sure to include your final answer **and** the reasoning you used to get that answer.

Your Answer:

Initial thought:

I want an equation that can represent current new cell phones sold in the U.S per year and then possibly extrapolate for future sales per year. I'm guessing as of 2022 a logarithmic equation is fitting to model the question. The graph would have the x-axis being "the year" and y-axis being "new cell phones sold in the U.S".

Assumptions for why it's logarithmic(just guesses):

I recall when the smartphone was first commercialized it was revolutionary. I'm going to assume the sales of new cell phones were exponential at the time.

In 2022 I’m going to assume near 100% market capitalization and possible market saturation.

I’m also assuming a bit of technological plateauing: the computing power for subsequent smartphones isn’t enough of a leap forward to incentivize people to buy future generations of cell phones.

I’m also assuming there are technological disruptions: current or future technologies(VR, AR, smart watches, or other smart gadgets with duplicate functionality) can possibly take market share.

I don’t keep track of the market share of new cell phones being sold in the U.S per year, but considering the above, since the conception of the smartphone, I believe it is plausible to consider the tapering of new cell phones sold in the U.S per year.

If not logarithmic then possibly a linear equation can model the question better, and if not linear then maybe it is still exponential in a growth phase.

Assumptions for estimating an actual number:

Need Total population of U.S.

Need a percentage of the population with cell phones + Need average number of cell phones per person + The percent of the population with cell phones that upgrade each year.

Need a percentage of population without cellphones + The percent of the population without cell phones that plan on buying a new one each year.

Average sales of new cell phones to corporate entities(not for personal use) each year.

Average sales of new cell phones to the public service sector(city/ government) each year.

Assuming only counting new cell phones sold from licensed distributors or direct sales from manufacturers. I also think that the categorized annual income statements (not sure if this is the actual name of the annual report) from the license distributors and manufacturers should be enough to get the information to answer the question we need, but let’s assume we have the data to calculate what we need based on the population.

Example: Let’s take a random year, say 2050. If Total U.S population = 1,000,000. Let’s say the average person has 1.3 cell phones. Let’s say 75% currently have cell phones and of the 75% with cell phones only 25% buy new cell phones each year. Of the 25% without cell phones only 10% buy one new cell phone each year. Let’s say companies and government/city entities together buy 250,000 cell phones per year for.

So we have:

totalWithCellPhonesBuyNewCellPhones = 1,000,000 * 0.75 * .025 * 1.3 = 243,750 phones

totalWithOutCellPhonesBuyNewCellPhones = 1,000,000 * 0.25 * 0.10 * 1 = 25,000 phones

totalCorporateAndGovernmentAndCity = 250,000 phones

currentNewCellPhonesSoldUSA2050 = 243,750 + 25,000 + 250,000 = 518750 phones

Conclusion:

So I would do the calculations above a few more times varying the population, other variables, or even readjusting the equation to account for more factors. After this is done I would plot the sum of total cell phone sales in the U.S against the year. Then modeling an equation off of the data using a large enough sample size and hopefully it'll look logarithmic.

Question 2

8 / 8 pts

Find the **running time T(n)** of the calculateSum method from this week's lecture video:

```
public static int calculateSum(int[] numbers) {
   int sum = 0;
   for(int i=0; i<numbers.length; i++) {
      sum += numbers[i];
   }
   return sum;
}
```

- Note that you are finding the **running time**, not the order of growth.
  - We know the order of growth is O(n).
- Review the examples in the lecture notes and videos for how to calculate the running time.
- List the steps leading to your calculation in case your answer is wrong but could get partial credit.
- Count *all* operations, including loop controls, declarations, updates, return, etc.
- i++ should be considered two statements (an addition and assignment).
- Accessing a value from an array does not need to be counted as a step.
  - For example, numbers[i] = x is 1 statement (an assignment), not 2 statements.

Your Answer:

```
// Question 2 T(n) "running time"
//public static int calculateSum(int[] numbers) {
//   int sum = 0;                            // 1 assignment
//   for(int i=0; i<numbers.length; i++) {    /* 1 assignment (first assign
ment) +...
//        ...1 conditional (first check) + n(1 addition + 1 assignment + 1
conditional)*/
//      sum += numbers[i];                    // n(1 addition + 1 assignmen
t)
//   }
//   return sum;                            // 1 return
//}
        // t(n) = 1 + 2 + n(3) + n(2) + 1
        // t(n) = 5n + 4
```

The method below creates a List of all duplicate integers found on a non-sorted List. This code runs in O(n^3) time. This is bad!

For this question, write a **new** method to accomplish the same task in O(n) time.

Before you begin: review the "bad" method.

- Make sure you understand why that is the order of growth... it's **not** because of the three loops!
- You'll notice that loop b and loop c will combine to only go through the list one time, so this is O(n) not O(n^2). So where does that O(n^3) come from? Look at the code inside of loop b to see if there are any O(n) method calls!
- If you aren't sure, post to the discussion board!

Write a new method that is linear.

- Write a **new** method to create a list of all duplicate integers found on a List.
- Note that you are **not** revising or fixing the method below. You are writing a **new** method.
- Your method should be linear O(n). Non-linear code will lose up to 15 points.

Details about the method:

- The numbers in the integer list passed in as a parameter are in the range -5n to 5n, where n is the size of the list.
  - Example: if the list is size 10, the numbers in the list are in the range -50 to 50.
  - Example: if the list is size 100, the numbers are in the range -500 to 500
- If a duplicate number shows up more than once on the original list, it should also show up more than once on the duplicate list.
  - Example: if the original list contains [1, 1, 1, 2, 2, 3, 4, 5, 5, 5], then the duplicates list should contain [1, 1, 2, 5, 5]
- The order of your duplicates list does not matter. For ease in testing/comparing, the driver program sorts the lists before printing.

Hints for Getting a Linear Solution:

- Remember that you can often improve efficiency by using more space/memory.
- Take advantage of the fact that you know something about the range of the numbers on the list. How can this help you keep track of whether or not you've seen a duplicate? What data structures can you use that have some O(1) methods you can use inside of loops?
- Carefully check the Big-O of any methods you use (e.g., sorting methods, contains methods, remove methods, etc.).
  - If a method is worse than O(n), you should not invoke it at all.
  - If a method is O(n), you should not invoke it from inside of a loop.
  - You can review a method's efficiency by reviewing the description on the API page for that class.
  - If you still aren't sure, post to the discussion board!

Use the Driver Program

- Use the driver program to test your code: **HomeworkM6Driver.java** (https://ccsf.instructure.com/courses/47904/files/7443089/download?download_frd=1)
- The driver program contains tests both to make sure your method works correctly and to estimate whether it is a linear solution.
- There are some variables at the start of main that set the size of the list and the number of tests you want to run. I recommend making these values smaller to start to help you find any fix any bugs. Once you've done that, you can make the numbers larger again for robust testing.

```
public static List<Integer> findDuplicatesBad(List<Inte
ger> numberList) {
   List<Integer> duplicateList = new ArrayList<Integer>
();

   // loop a: this loop is O(n)- it iterates over the w
hole list
   for(int i=0; i<numberList.size(); i++) {
      int numberEvaluating = numberList.get(i);
      boolean duplicateFound = false;

      // loop b: this loop starts at i+1 and goes to th
e end of the list OR until a duplicate is found
      for(int j=i+1; j<numberList.size() && !duplicateF
ound; j++) {
         int numberChecking = numbnumberListrs.get(j);

         // we have found a duplicate that hasn't yet b
een put on the duplicateList
         if(numberEvaluating==numberChecking && !duplic
ateList.contains(numberEvaluating)) {
            duplicateFound = true;

            // loop c: after a duplicate is found, we w
on't return to loop b
            // instead, loop c finishes checking the re
st of the list and puts all copies of
            // of the current duplicate on the duplicat
eList
            for(int k=j; k<numberList.size(); k++) {
               if(numberChecking==Integer.valueOf(numbe
rList.get(k))) {
                  duplicateList.add(numberChecking);
               }
            }
         }
      }
   }
   return duplicateList;
}
```

Question 3

30 / 30 pts

Paste the complete linear method here.

Your Answer:

```
    public static List<Integer> findDuplicatesLinear(List<Integer> numberLi
st) {

        int listSize = ((2 * 5 * numberList.size()) - 1); // 2 * 5n - 1 //
two times 5n possible values, 0 is shared
        int indexToValueCipher = ((5 * numberList.size()));
        /*
         * add indexToValueCipher to the value of the index(from indexValue
ListLocked)
         * to store it's(the value) frequency subtract indexToValueCipher f
rom the
         * index(from indexValueListLocked) to convert it to its original v
alue
         */
        // Holds the frequency of all possible values in range -5n to 5n
        int[] valuesListLocked = new int[listSize + 2];
        /*
         * use indexToValueCipher to get orignal value stored to prevent +
2 is to
         * prevent an out index of bounds exemption
         */
        // create duplicate list
        List<Integer> duplicateList = new ArrayList<Integer>();

        // return empty list if size of numberList < 2
        if (numberList.size() < 2) {
            return duplicateList;
        } else {

            // counting the frequency of all possible values in range -5n t
o 5n
            for (int i = 0; i < numberList.size(); i++) {
                valuesListLocked[numberList.get(i) + indexToValueCipher] +=
1; // stores the frequency of the value
            }

            // added recurring values to the ArrayList to be returned
            for (int j = 0; j < valuesListLocked.length; j++) {
                if (valuesListLocked[j] >= 2) {
                    for (int k = 1; k < valuesListLocked[j]; k++) { // ~ co
nstant time loop
                        duplicateList.add(j - indexToValueCipher);
                    }
                }
            }
        }
        return duplicateList;
    }
```

Multiple Choice Questions

For the next set of questions, determine the order of growth (Big O) of the code or the situation.

Question 4

2 / 2 pts

display all integers in an array of integers

- O(log n)
- none of these is correct
- O(1)
- O(n^2)
- [✓ O(n)] **Correct!**

Question 5

2 / 2 pts

display all integers in a chain of linked nodes

- O(n^2)
- O(log n)
- none of these is correct
- [✓ O(n)] **Correct!**

○ O(1)

Question 6

2 / 2 pts

display a value at a specified index in an array of integers

For example, display the 3rd element in the array, display the 10th element in the array, etc.

○ O(n)

○ none of these is correct

○ O(n^2)

○ O(log n)

Correct! [✓ O(1)]

Question 7

2 / 2 pts

display a value at a specified index in a linked chain of integers

For example, display the 3rd element in the chain, display the 10th element in the chain, etc.

Correct! [✓ O(n)]

○ none of these is correct

- O(log n)
- O(1)
- O(n^2)

Question 8

2 / 2 pts

display the last node in a chain of linked nodes for which you only have a head reference

Correct!

- [✓ O(n)]
- O(1)
- O(log n)
- none of these is correct
- O(n^2)

Question 9

2 / 2 pts

display the last node in a chain of linked nodes for which you have a head and a tail reference

- O(n^2)
- none of these is correct

○ O(log n)

○ O(n)

Correct! [✓ O(1)]

Question 10

2 / 2 pts

```
for(int i=1; i<=n; i++) {
   for(int j=0; j<n; j++) {
      for(int k=1; k<10; k++) {
         // code here that is independent of n
      }
   }
}
```

Correct! [✓ O(n^2)]

○ none of these is correct

○ O(log n)

○ O(1)

○ O(n)

Question 11

2 / 2 pts

```
for(int i=1; i<=n; i++) {
   for(int j=0; j<n; j++) {
      for(int k=1; k<n; k++) {
         // code here that is independent of n
      }
```

```
    }
}
```

- O(1)

Correct!

- [✓ none of these is correct]
- O(n)
- O(log n)
- O(n^2)

Question 12

0 / 2 pts

```
int sum=0;

for(int counter = 1; counter < n; counter = 2 * counte
r) {
    sum = sum + counter;
}
```

Hint: pay careful attention to how the update function of the loop changes how often the loop runs! What happens if you double the problem size, how does the number of loops change? I recommend writing out a table and counting how many times the loop runs if n = 2, 4, 8, 16, 32, etc.

- O(1)
- O(n^2)

ou Answered

- [✓ O(n)]

orrect Answer

O(log n)

none of these is correct

the problem size is cut in half each time through (since the counter is being doubled); this is logarithmic growth

Question 13

2 / 2 pts

```
int sum=0;

for(int counter = n; counter > 0; counter = counter-2)
{
   sum = sum + counter;
}
```

Hint: pay careful attention to how the update function of the loop changes how often the loop runs! What happens if you double the problem size, how does the number of loops change? I recommend writing out a table and counting how many times the loop runs if n = 2, 4, 8, 16, 32, etc.

none of these is correct

O(log n)

Correct!

[✓ O(n)]

O(1)

O(n^2)

If thinking about actual time or running time, this loop will execute fewer times than a loop with the update counter--. But this loop is still O(n) because when the problem size doubles, the running time doubles.

Question 14

2 / 2 pts

```
for(int i=0; i<100; i++) {
   System.out.println(n);
}
```

- O(n^2)
- none of these is correct
- O(log n)
- **Correct!** [✓ O(1)]
- O(n)

Question 15

2 / 2 pts

```
for(int i=0; i<10; i++) {
   for(int j=0; j<100; j++) {
      // code that is O(n)
   }
}
```

- O(log n)
- none of these is correct
- O(1)
- O(n^2)
- **Correct!** [✓ O(n)]

Question 16

2 / 2 pts

```
for(int i=0; i<n; i++) {
  // code that is independent of n
}

for(int j=0; j<n; j++) {
   // code that is independent of n
}
```

- O(n^2)
- O(1)
- **Correct!** [✓ O(n)]
- none of these is correct
- O(log n)

Question 17

2 / 2 pts

```
// arrayList is type ArrayList

for(int i=0; i<arrayList.size(); i++) {
   System.out.println(arrayList.get(i));
}
```

- O(1)
- O(n^2)
- none of these is correct
- O(log n)
- **Correct!** [✓ O(n)]

Question 18

2 / 2 pts

```
// arrayList is type ArrayList<String> filled with text o
f numbers that match their index: "0", "1", "2", "3", "
4", ...

int n = arrayList.size();

for(int i=0; i<n; i++) {
   String numberString = Integer.toString(i);
   arrayList.remove(numberString);
}
```

- **Correct!** [✓ O(n^2)]
- none of these is correct
- O(1)

O(n)

O(log n)

Question 19

2 / 2 pts

```
// arrayList is type ArrayList

for(int i=0; i<5; i++) {
   Collections.shuffle(arrayList);
}
```

O(1)

O(log n)

O(n^2)

none of these is correct

Correct! [✓ O(n)]

Question 20

2 / 2 pts

```
// myAList is type AList

for(int i=1; i<myAList.getLength() / 2 ; i++) {
   System.out.println(myAList.getEntry(i));
}
```

O(1)

( ) none of these is correct

Correct! [✓ O(n)]

( ) O(log n)

( ) O(n^2)

Question 21

2 / 2 pts

```
// numberArray is type int[]

Arrays.sort(numberArray);

for(int num : numberArray) {
   System.out.println(num);
}
```

Correct! [✓ none of these is correct]

( ) O(1)

( ) O(log n)

( ) O(n)

Arrays.sort is O(n log n). The following loop is O(n), but since the code is consecutive, the higher big o wins out.

Question 22

2 / 2 pts

```
// myLList is type LList<Integer>

for(int i=0; i<10; i++) {
    myLList.add(Integer.valueOf(i));
}
```

- ○ O(n^2)
- ○ none of these is correct
- ○ O(1)
- **Correct!** [✓ O(n)]
- ○ O(log n)

Question 23

2 / 2 pts

```
// myLList is type LList<Integer>

for(int i=1; i<myLList.getLength(); i++) {
   System.out.println(myLList.getEntry(i));
}
```

- none of these is correct
- O(1)
- O(log n)
- O(n)
- Correct! [✓ O(n^2)]

Question 24

6 / 6 pts

Consider the ArrayBag class, which uses a fixed-size array. What is the order of growth of the following methods for this class?

Correct!	**add(T)**	O(1)
Correct!	**remove()**	O(1)
Correct!	**remove(T)**	O(n)
Correct!	**contains(T)**	O(n)

Other Incorrect Match Options:

- O(log n)
- O(n^2)

Question 25

6 / 6 pts

Consider the LinkedBag class, which uses a linked nodes. What is the order of growth of the following methods for this class?

Correct!	**add(T)**	O(1)
Correct!	**remove()**	O(1)
Correct!	**remove(T)**	O(n)
Correct!	**contains(T)**	O(n)

Other Incorrect Match Options:

- O(n^2)
- O(log n)

Quiz Score: **98** out of 100