Data Structures and Object Oriented Programming

Homework #3

Instructor: Marina Langlois

Project 3 is due January 29th, 2017, at 23:59:59.

Grading: 90 points

Submit your files using Vocareum. Make sure that you follow directions on how to submit files precisely. Read the submission report after you submitted your files.

Contents

Overview of the project

In this assignment you will practice with Big-O (and Big-Omega and Big-Theta), analyze the running time of small pieces of code, create and then run empirical tests on simple methods and write your first C-program,

1 Part 1. Analyzing running time. (20 points in total)

This section has two parts, the first part asks you to state whether each of the following equalities are true or false; the second part asks you to provide a proof for some of the statements.

1.1 True/False

Please create **HW3Part1a.txt** text file. Do NOT put your name and ID so that we can grade this file automatically. You should put the number of the question followed by either the word True or False. One answer per line. Eg: Please follow this format.

1. True
2. False

You should use the **true** (not abused) meaning of Big-O (and Big-Omega), not just the tightest bound.

Questions:

1. $10000n^4 + \log n = O(n^5)$
2. $8000n^2 + n^3 + n = O(n^2)$
3. $2^n + n! = O(3^n)$
4. $\frac{n^5}{n^2} + n^4 + 100n = O(n^4)$
5. $\frac{1}{n^2} + 5 = O(\frac{1}{n})$
6. $n^2 + \log^2 n^2 = \Omega(n)$
7. $n! = \Omega(n^n)$
8. $\log(\log n^n) + \log n = \Omega(\log n)$
9. $\frac{1}{n^2} = \Omega(1)$
10. $n^2 * n^4 + \log \frac{n}{8} = \Omega(n^6)$
11. $\sqrt{n} + n = \Theta(n \log n)$

12. $100n^3 + n^2 = \Theta(n^3)$
13. $\frac{1}{n} + \frac{n^2}{n} = \Theta(n)$
14. $\frac{1}{n^{100}} + \log 16 = \Theta(1)$
15. $n + \log n^2 = \Theta(5n)$

1.2 Explanation (proof)

Prove your answer for 11 - 15 only, using the definition of Big-Theta. Clearly state what **c** and n_0 you choose for the *True* answer (these constants do not have to be the same for big-O and big-Theta) or explain why such constants can not be chosen. Put your answers in **HW3Part1b.pdf** file.

2 Part 2. Analyzing Running time for simple programs. (20 points)

.

You will practice your skills of estimating running time of code snippets. For each piece of code below, state the running time of the snippet in terms of the loop limit variable, n. You should assume that all variables are already declared and have a value. You should express your answer using Big-O or Big-Θ (Theta) notation, though your Big-O bound will only receive full credit if it is a **tight bound**. We allow you to use Big-O because it is often the convention to express only upper bounds on algorithms or code, but these upper bounds are generally understood to be as tight as possible. In all cases, your expressed bounds should be simplified as much as possible, with no extra constant factors or additional terms. (for example, O(n) instead of O(2n+5)).

Answer format

Place your answers in your file **HW3Part2.pdf** file. For each piece of the question, state the running time of the code snippet and then give a short explanation on why that running time is correct. Your explanation should include an (approximate but reasonable) equation for how many instructions are executed, and then a relationship between your equation and your stated bound.

Sample question and answer

$for(i = 5; i < n; i++)$
$\quad sum++;$

Answer: Running time is O(n)
Explanation: There is a single loop that runs $n-5$ times. Each time the loop runs it executes

1 instruction in the loop header and 1 instruction in the body of the loop. The total number of instructions is $2 * (n - 5) + 1$ (for the last loop check) $= 2n - 9 = O(n)$. (also OK: $\Theta(n)$).

Questions

1) num=0;
for (i = 0; i<= 100n; i++)
num++;

2) num=0;
for (i = n*n*n; i>= 0; i=i-4)
num++;

3) num=0;
for (i = n; i>= 0; i=i/2)
num = num + n;

4) num=0;
for (i = 1; i<=100; i++)
for (j = 1; j<=10000; j=j*2)
num = num + i;

5) num=0;
for (i = 1; i<=n; i++)
for (j = 1; j<=i; j++)
num = num + i;

6) num=0;
for (i = 1; i<=n; i=i*2)
for (j = 1; j<=n; j=j+4)
num = num + i;

7)
for (i = 1; i<= 2*n; i++)
num++;
for (j = 0; j<=n*n; j++)
for (i=0; i<= n; i++)
num++;

8)num=0;
for (i = 0; i<n-1; i++)
for (j = 0; j< $i * i - 1$; j++)
num = num + i;

```
9) for (i = n-1; i > 0; i−−) {
        MPos = i;
        for (j = 0; j < i; j++) {
              if (a[j] > a[MPos])
              MPos = j;
        }
        swap(i, MPos); // has three instructions
   }

10) for (i = n; i > 0; i = i/2) {
        for (j = i; j > 0; j = j/2) {
              //constant time operations
        }
    }
```

3 Part 3. Running time of LinkedList and Sorted ArrayList functions. (20 points)

In this section, you will analyze minimum time needed to implement several operations on a Doubly LinkedList without a tail and Sorted ArrayList.

In your **HW3Part3.pdf** file, give the minimum required running time to execute each of the following operations in a Doubly Linked List with head (but without tail) pointer, and a sorted Array List in the worst case, using Big-Ω notation, assuming n is the number of elements in the list. Following each expression, include a 1-2 sentence argument about why an algorithm to perform the given operation could not run faster than the bound you give. As above, full credit will be given only for tight Big-Ω bounds. That is, it is not sufficient to say that all operations take $\Omega(1)$. This is trivially true for any piece of code. (Though in some cases this will be the tightest Big-Ω bound).

Example Question: Adding a value to the beginning of the linked list.
Example Answer: Running time: $\Omega(1)$. You have a pointer to the head node in the list, so adding an element involves creating a new node (which is not dependent on the length of the list), setting the links in the new node, and changing the values of the head references. This takes somewhere around 10 steps to perform, and $10=\Omega(1)$.

Example Question: Adding a value to the beginning of the array list.
Example Answer: Running time: $\Omega(n)$. You have to move everything over one cell to create a room for a new element. It will be done using a for loop that iterates through all elements. Since there are n elements in the list, it takes $\Omega(n)$ to add a new element to the beginning of the array list.

Questions (refers to both Linked list without a tail and Sorted Array List, so you need to have 10 answers.

1. Reversing the list (content of each node).
2. Adding a value to the end of the list.
3. Removing the value from the list at a given index.
4. Removing the first value from the list.
5. Determining whether the list contains some value v.

4 Part 4. Performing Runtime exploration

This problem asks you to estimate and then measure two search methods, one on the linked lists and the second one on the sorted arrays.

How to measure running time. (Depends on the size of your input).

Returns current time in milliseconds:
    static long System.currentTimeMillis()
Returns current time in nanoseconds:
    static long System.nanoTime()

Directions:

1) In a file named **Complexity.java**, you need to create a LinkedList and an ArrayList objects.

2) Populate with numbers so that both lists are sorted with unique numbers. It is up to how you proceed with this step. There are a lot of different ways you can do it. Do NOT include this step into performance measure.

3) Create two search methods that search for a given value and return its index if found, or -1 otherwise.

4) For each of the methods estimate its complexity theoretically. Use Big-Theta for your answer. You need to find the best algorithm possible. Hint: One method will be more efficient that another.

5) Then you will run these methods on various input sizes and calculate the average for these runs for more accurate estimates.

For example: Run method search1 1000 times for n=5000 compute the running time for

each run, and take the average over 1000 runs. Record this value. Repeat for different n: [6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000] Then repeat the the same procedure for method search2.

6) Using the values from the above experiments create two plots, one for each method. For each plot, the 10 data points will be connected by a smooth curve. The x-axis will be n (input size) and the y-axis will be time. You may use any software you wish, but you will be penalized if your graphs are not clear. Screenshot these graphs and include them in **Plots.pdf**.

7) In **HW3Part4.pdf**, write a few sentences explaining the results of your experiment. Clearly explain the expected (theoretical) running time and the observed running times. You must explain the runtimes in terms of their complexity.

5 Turning files in

In EACH AND EVERY FILE that you turn in (except **HW3Part1a.txt**), we need the following comments at the top of each file. These are essential so that we can more easily process your submissions and insure that you receive proper credit.

NAME: <your name>
ID: <your student ID>
LOGIN: <your class login>

If everything is correct with your submission, then you will see submission.txt file that says 'Everything looks good! This is a successful submission'. If something is wrong (name of the file, number of the files etc) there will be an error message. Make sure you correct your errors and resubmit. We are going to grade the latest version you submit. **Please remember that we will only grade a successful submission.**

Files and directories to submit

1. HW3Part1a.txt
2. HW3Part1b.pdf
3. HW3Part2.pdf
4. HW3Part3.pdf
5. Plots.pdf
6. HW3Part4.pdf
7. Complexity.java

6 Appendix

6.1 Style guide

Refer to the course website.

6.2 Converting different files to .pdf

It is not enough just to change extension from, say, .docx to .pdf. Try it, and then open your new "pdf" file. Did not work, right? It means you need a few extra steps to make a conversion.
Refer to HW1 write-up for instructions to convert doc to pdf.