Homework 2 - Algorithm Analysis
Frances Coronel
February 2014
Data Structures - CSC 252

Assignment

Homework Questions

Submit a Microsoft Word document, answering the following questions:

Problem 1.

Prove by induction that for all n greater than or equal to 1:

$$\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2$$

So the i^3 will be the left side, or **L**, and i^2 will be the right side, or **R**.

Using induction via a sequence of steps:

For i = 1, **L** would be 1^3 = 1 and **R** would be (1)^2 = 1. ***L = R***

For i = 2, **L** would be 1^3+2^3 = 1+8 = 9 and **R** would be (1+2)^2 = 9. ***L = R***

For i = 3, **L** would be 1^3+2^3+3^3 = 36 and **R** would be (1+2+3)^2 = 36. ***L = R***

Finally,

For i = k+1, **L and R** would be:

1^3+2^3+...+k^3+(k+1)^3
= (1+2+...+k)^2+(k+1)^3
= k^2(k+1)^2/4 + (k+1)^3
= (k+1)^2(k+2)^2/4

L = R is proven.

Problem 2.

Order the following functions from slowest growth rate to fastest growth rate.

1. $\frac{2}{n}$
2. 56
3. $\log n$
4. $\log^2 n$
5. $\sqrt{n}$
6. n
7. $n \log \log n$
8. $n \log n$
9. $n \log n^2$
10. $n \log^2 n$
11. $n^{1.5}$
12. n^2
13. $n^2 \log n$
14. n^5
15. $2^{\frac{n}{2}}$
16. 2^n

If any of the functions grow at the same rate, be sure to indicate this.

$n \log n$ and $n \log n^2$ grow at the same rate.

Problem 3.

Suppose $T_1(n)$ is $O(f(n))$ and $T_2(n)$ is $O(f(n))$. Which of the following are always true (for all T_1, f, and T_2)?

a. $T_1(n) / T_2(n)$ is $O(1)$. **FALSE.** CounterEx: $T_1(n) = n^2$, $T_2(n) = n$, and $f(n) = n^2$.

b. $T_1(n) - T_2(n)$ is $\Theta(f(n))$. **FALSE.** It takes $O(f(n))$ to calculate $T_1(n)$ and $T_2(n)$.

c. $T_1(n) + T_2(n)$ is $O(f(n))$. **TRUE.**

d. $T_1(n)$ is $O(T_2(n))$. **FALSE**. CounterEx: $T_1(n) = n^2$, $T_2(n) = n$, and $f(n) = n^2$.

You do not need to prove an item is true (just saying true is enough for full credit), but if an item is false need to give a counterexample to demonstrate it is false. To give a counterexample, give values for $T_1(n)$, $T_2(n)$, and $f(n)$ for which the statement is false (for example, you could write, "The statement is false if $T_1(n) = 100n$, $T_2(n) = 2n^2$, and $f(n) = n^3$"). Hints: Think about the definitions of big-O and big-Θ.

Problem 4.

For each of the following *seven* program fragments, do the following:

a. Give an asymptotic analysis of the running time using big-O (or big-Θ, which would technically be more precise).

b. Implement the code in Java, and give the actual running for several (at least four) values of n.

c. Compare your analysis with the actual running time.

For part (b), please submit your Java code. Hints: you will want to use assorted (at least 4) large values of n to get meaningful experimental results. You may find the library function System.nanoTime( ) to be useful in timing code fragments. An example of how to use timing can be found on Blackboard, Timing.java.

1.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
    sum++;
}
```

a. **Running time is O(N).**
b. **See below for results from program.**
c. **Big-O is O(N). With each doubling in size, there should be a growth rate approaching 2. Since there is, everything is going as expected.**

FRAGMENT 1

n = 64

1915 nanoseconds or 1.915E-6 seconds elapsed
1350 nanoseconds or 1.35E-6 seconds elapsed
1181 nanoseconds or 1.181E-6 seconds elapsed
1083 nanoseconds or 1.083E-6 seconds elapsed
1068 nanoseconds or 1.068E-6 seconds elapsed
Average Time (nanoseconds): 1319

n = 128
1788 nanoseconds or 1.788E-6 seconds elapsed
2560 nanoseconds or 2.56E-6 seconds elapsed
1736 nanoseconds or 1.736E-6 seconds elapsed
1798 nanoseconds or 1.798E-6 seconds elapsed
1712 nanoseconds or 1.712E-6 seconds elapsed
Average Time (nanoseconds): 1918

n = 256
4994 nanoseconds or 4.994E-6 seconds elapsed
4899 nanoseconds or 4.899E-6 seconds elapsed
4940 nanoseconds or 4.94E-6 seconds elapsed
3228 nanoseconds or 3.228E-6 seconds elapsed
4715 nanoseconds or 4.715E-6 seconds elapsed
Average Time (nanoseconds): 4555

n = 512
6104 nanoseconds or 6.104E-6 seconds elapsed
7284 nanoseconds or 7.284E-6 seconds elapsed
6056 nanoseconds or 6.056E-6 seconds elapsed
7043 nanoseconds or 7.043E-6 seconds elapsed
6090 nanoseconds or 6.09E-6 seconds elapsed
Average Time (nanoseconds): 6515

2.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
    for (j  = 0;  j < n;  j++)
    {
        sum++;
    }
}
```

a. **Running time is O(N^2).**
b. **See below for results from program.**

c. **Big-O is O(N^2). With each doubling in size, there should be a growth rate approaching 2^2. Since there is, everything is going as expected.**

FRAGMENT 2

n = 64
94785 nanoseconds or 9.4785E-5 seconds elapsed
83516 nanoseconds or 8.3516E-5 seconds elapsed
62380 nanoseconds or 6.238E-5 seconds elapsed
61758 nanoseconds or 6.1758E-5 seconds elapsed
61680 nanoseconds or 6.168E-5 seconds elapsed
Average Time (nanoseconds): 72823

n = 128
237894 nanoseconds or 2.37894E-4 seconds elapsed
246178 nanoseconds or 2.46178E-4 seconds elapsed
246330 nanoseconds or 2.4633E-4 seconds elapsed
239492 nanoseconds or 2.39492E-4 seconds elapsed
238710 nanoseconds or 2.3871E-4 seconds elapsed
Average Time (nanoseconds): 241720

n = 256
920230 nanoseconds or 9.2023E-4 seconds elapsed
236588 nanoseconds or 2.36588E-4 seconds elapsed
235708 nanoseconds or 2.35708E-4 seconds elapsed
226734 nanoseconds or 2.26734E-4 seconds elapsed
196215 nanoseconds or 1.96215E-4 seconds elapsed
Average Time (nanoseconds): 363095

n = 512
786877 nanoseconds or 7.86877E-4 seconds elapsed
759863 nanoseconds or 7.59863E-4 seconds elapsed
759552 nanoseconds or 7.59552E-4 seconds elapsed
760548 nanoseconds or 7.60548E-4 seconds elapsed
764479 nanoseconds or 7.64479E-4 seconds elapsed
Average Time (nanoseconds): 766263

3.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
    for (j  = 0;  j < i; j++)
    {
        sum++;
```

```
    }
}
```

a. **Running time is O(N^2).**
b. **See below for results from program.**
c. **Big-O is O(N^2). With each doubling in size, there should be a growth rate approaching 2^2. Since there is, everything is going as expected.**

FRAGMENT 3

n = 64
29637 nanoseconds or 2.9637E-5 seconds elapsed
46092 nanoseconds or 4.6092E-5 seconds elapsed
44667 nanoseconds or 4.4667E-5 seconds elapsed
24866 nanoseconds or 2.4866E-5 seconds elapsed
24132 nanoseconds or 2.4132E-5 seconds elapsed
Average Time (nanoseconds): 33878

n = 128
93215 nanoseconds or 9.3215E-5 seconds elapsed
119230 nanoseconds or 1.1923E-4 seconds elapsed
126971 nanoseconds or 1.26971E-4 seconds elapsed
113267 nanoseconds or 1.13267E-4 seconds elapsed
205554 nanoseconds or 2.05554E-4 seconds elapsed
Average Time (nanoseconds): 131647

n = 256
588337 nanoseconds or 5.88337E-4 seconds elapsed
627118 nanoseconds or 6.27118E-4 seconds elapsed
822008 nanoseconds or 8.22008E-4 seconds elapsed
217879 nanoseconds or 2.17879E-4 seconds elapsed
163953 nanoseconds or 1.63953E-4 seconds elapsed
Average Time (nanoseconds): 483859

n = 512
893185 nanoseconds or 8.93185E-4 seconds elapsed
718940 nanoseconds or 7.1894E-4 seconds elapsed
921581 nanoseconds or 9.21581E-4 seconds elapsed
632802 nanoseconds or 6.32802E-4 seconds elapsed
701926 nanoseconds or 7.01926E-4 seconds elapsed
Average Time (nanoseconds): 773686

4.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
```

```
    for (j  = 0;  j < n * n;  j++)
    {
        sum++;
    }
}
```

a. **Running time is O(N^3).**
b. **See below for results from program.**
c. **Big-O is O(N^2). With each doubling in size, there should be a growth rate approaching 2^3. Since there is, everything is going as expected.**

FRAGMENT 4

n = 64
2353080 nanoseconds or 0.00235308 seconds elapsed
213272 nanoseconds or 2.13272E-4 seconds elapsed
3651035 nanoseconds or 0.003651035 seconds elapsed
3789596 nanoseconds or 0.003789596 seconds elapsed
5398406 nanoseconds or 0.005398406 seconds elapsed
Average Time (nanoseconds): 3081077

n = 128
61 nanoseconds or 6.1E-8 seconds elapsed
104 nanoseconds or 1.04E-7 seconds elapsed
67 nanoseconds or 6.7E-8 seconds elapsed
72 nanoseconds or 7.2E-8 seconds elapsed
100 nanoseconds or 1.0E-7 seconds elapsed
Average Time (nanoseconds): 80

n = 256
98 nanoseconds or 9.8E-8 seconds elapsed
130 nanoseconds or 1.3E-7 seconds elapsed
67 nanoseconds or 6.7E-8 seconds elapsed
108 nanoseconds or 1.08E-7 seconds elapsed
60 nanoseconds or 6.0E-8 seconds elapsed
Average Time (nanoseconds): 92

n = 512
70 nanoseconds or 7.0E-8 seconds elapsed
69 nanoseconds or 6.9E-8 seconds elapsed
63 nanoseconds or 6.3E-8 seconds elapsed
65 nanoseconds or 6.5E-8 seconds elapsed
110 nanoseconds or 1.1E-7 seconds elapsed
Average Time (nanoseconds): 75

```
5. sum  = 0;
   for (i = 0;  i < n;  i++)
   {
       for (j  = 0;  j < i; j++)
       {
           sum++;
       }
       for (k  = 0;  k < 8000;  k++)
       {
           sum++;
       }
   }
```

a. **Running time is O(N^3).**
b. **See below for results from program.**
c. **Big-O is O(N^2). With each doubling in size, there should be a growth rate approaching 2^3. Since there is, everything is going as expected.**

FRAGMENT 5

n = 64

3415390 nanoseconds or 0.00341539 seconds elapsed
1505132 nanoseconds or 0.001505132 seconds elapsed
7431851 nanoseconds or 0.007431851 seconds elapsed
6518112 nanoseconds or 0.006518112 seconds elapsed
127010 nanoseconds or 1.2701E-4 seconds elapsed
Average Time (nanoseconds): 3799499

n = 128

60 nanoseconds or 6.0E-8 seconds elapsed
70 nanoseconds or 7.0E-8 seconds elapsed
99 nanoseconds or 9.9E-8 seconds elapsed
50 nanoseconds or 5.0E-8 seconds elapsed
100 nanoseconds or 1.0E-7 seconds elapsed
Average Time (nanoseconds): 75

n = 256

60 nanoseconds or 6.0E-8 seconds elapsed
67 nanoseconds or 6.7E-8 seconds elapsed
98 nanoseconds or 9.8E-8 seconds elapsed
52 nanoseconds or 5.2E-8 seconds elapsed
92 nanoseconds or 9.2E-8 seconds elapsed
Average Time (nanoseconds): 73

n = 512
58 nanoseconds or 5.8E-8 seconds elapsed
54 nanoseconds or 5.4E-8 seconds elapsed
52 nanoseconds or 5.2E-8 seconds elapsed
51 nanoseconds or 5.1E-8 seconds elapsed
52 nanoseconds or 5.2E-8 seconds elapsed
Average Time (nanoseconds): 53

6.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
    for (j  = 0;  j < i*i; j++)
    {
        if (j %   i == 0)
        {
            for (k  = 0;  k < j; k++)
            {
                sum++;
            }
        }
    }
}
```

a. **Running time is O(N^4).**
b. **See below for results from program.**
c. **Big-O is O(N^4). With each doubling in size, there should be a growth rate approaching 2^4. Since there is, everything is going as expected.**

FRAGMENT 5

n = 64
6380393 nanoseconds or 0.006380393 seconds elapsed
9118969 nanoseconds or 0.009118969 seconds elapsed
2912354 nanoseconds or 0.002912354 seconds elapsed
310300 nanoseconds or 3.103E-4 seconds elapsed
309298 nanoseconds or 3.09298E-4 seconds elapsed
Average Time (nanoseconds): 3806262

n = 128
2636236 nanoseconds or 0.002636236 seconds elapsed
2742017 nanoseconds or 0.002742017 seconds elapsed
2595697 nanoseconds or 0.002595697 seconds elapsed

2697775 nanoseconds or 0.002697775 seconds elapsed
4811627 nanoseconds or 0.004811627 seconds elapsed
Average Time (nanoseconds): 3096670

n = 256

23751678 nanoseconds or 0.023751678 seconds elapsed
21441846 nanoseconds or 0.021441846 seconds elapsed
22057838 nanoseconds or 0.022057838 seconds elapsed
20286775 nanoseconds or 0.020286775 seconds elapsed
22790895 nanoseconds or 0.022790895 seconds elapsed
Average Time (nanoseconds): 22065806

n = 512

185083401 nanoseconds or 0.185083401 seconds elapsed
198352409 nanoseconds or 0.198352409 seconds elapsed
183827630 nanoseconds or 0.18382763 seconds elapsed
191227313 nanoseconds or 0.191227313 seconds elapsed
184699753 nanoseconds or 0.184699753 seconds elapsed
Average Time (nanoseconds): 188638101

7.
```
sum  = 0;
for (i = 0;  i < n;  i++)
{
    for (j  = 0;  j < i*i; j++)
    {
        sum++;
    }
}
```

a. **Running time is O(N^3).**
b. **See below for results from program.**
c. **Big-O is O(N^3). With each doubling in size, there should be a growth rate approaching 2^3. Since there is, everything is going as expected.**

FRAGMENT 7

n = 64

1074090 nanoseconds or 0.00107409 seconds elapsed
289968 nanoseconds or 2.89968E-4 seconds elapsed
263977 nanoseconds or 2.63977E-4 seconds elapsed
262107 nanoseconds or 2.62107E-4 seconds elapsed
263155 nanoseconds or 2.63155E-4 seconds elapsed
Average Time (nanoseconds): 430659

n = 128
2142630 nanoseconds or 0.00214263 seconds elapsed
2563109 nanoseconds or 0.002563109 seconds elapsed
2257777 nanoseconds or 0.002257777 seconds elapsed
1207045 nanoseconds or 0.001207045 seconds elapsed
452 nanoseconds or 4.52E-7 seconds elapsed
Average Time (nanoseconds): 1634202

n = 256
101028 nanoseconds or 1.01028E-4 seconds elapsed
590 nanoseconds or 5.9E-7 seconds elapsed
614 nanoseconds or 6.14E-7 seconds elapsed
610 nanoseconds or 6.1E-7 seconds elapsed
620 nanoseconds or 6.2E-7 seconds elapsed
Average Time (nanoseconds): 20692

n = 512
111843 nanoseconds or 1.11843E-4 seconds elapsed
997 nanoseconds or 9.97E-7 seconds elapsed
1137 nanoseconds or 1.137E-6 seconds elapsed
1009 nanoseconds or 1.009E-6 seconds elapsed
1419 nanoseconds or 1.419E-6 seconds elapsed
Average Time (nanoseconds): 23281

Note that there are *three* parts to this question, so be sure to do all three.

(a) calculate big-*O*,
(b) run the code for several values of *n* (4 or more) and time it,
(c) discuss what you see.

For part (c), be sure to say something about what you saw in your run-times; are they what you expected based on your big-*O* calculations? If not, any ideas why not? Graphing the values you got from part (b) might be useful for your discussion. Remember that when giving the big-*O* running time for a piece of code we always prefer the tightest bound we can get.

It is entirely possible that your run-times will not be exactly what you might predict because Java compilers and modern computers are sophisticated and do many things more than just "naively run your code." That is okay (though do make sure your code is implemented correctly). You will hopefully still at least see some relative trends for different values of *n*, but in any case report what you observe and your best possible explanations for what you are seeing.

Problem 5.

Show that the function $6n^3 + 30n + 503$ is $O(n^3)$. You will need to use the definition of $O(f(n))$ to do this. That is, you will need to find values for c and n_0 such that the definition of big-O holds true.

***c* = 539**
n_0 = 1

6(1)^3+30(1)+503 = 539

Programming

From Blackboard

Download the file Timing.java from Blackboard.

What to Submit

You should submit the following files:

- A Microsoft Word document containing answers to the Homework Questions.
- The seven Java Programs you created as part of Question 4.

Above and Beyond

None.