CMPT 280

Topic 3: Timing Analysis

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 3

Timing Analysis

Express the time required by an algorithm for an input of size n in Big-Oh notation. Also referred to as find the *time complexity* of an algorithm.

1. Determine time required as a function of the input size n (may use best-case or worst-case analysis).
   - Statement counting
   - Active operation
2. Simplify the resulting function to Big-Oh notation.
   - By inspection
   - By formal proof
3. If best and worst case analysis yield same result, may write as Big-Theta.

Exercise 1

Consider this method which returns true if the array a contains the string s.

```
public static bool contains(String[ ] a, String s)
{
     int count = 0;
     int i = 0;
     while (i < a.length && !s.equals(a[i]))
     {
          i = i + 1;
     }
     return i < a.length;
}
```

a) What is the best case input?

b) What is the worst case input?

c) Exactly how many statements are executed in the worst case? Best case?

Exericse 2

```
public static <T extends Comparable<T>>
               void insertionSort(T[ ] a)
{
     // For each string in the array...
     int i = 1;
     while (i < a.length)
     {
          // Get the i-th element.
          T  temp = a[i];

          // Examine all of the elements that come before it, starting at
          // the rightmost.  If 'temp' comes before a[j], move a[j] one
          // index to the right and continue.

          int j = i - 1;
          while (j >= 0 && temp.compareTo(a[j]) < 0)
          {
               a[j+1] = a[j];
               j = j - 1;
          }

          // We've found the insertion point.  Copy the string here
          // and advance to the next insertion.
          a[j+1] = temp;
          i = i + 1;
     }
}
```

- Exactly how many statements are executed in the worst case?

Exercise 3

- Repeat exercise 1 and 2 using the active operation approach. Exactly how many times does the active operation get executed?

Exercise 4

What is the Big-Oh membership of each of the following functions? We would like the tightest upper bound. Simplify by inspection.

- n
- $47n \log n + 10000n$
- $100n + 500 \log n + 1000$
- $\log n + 100\sqrt{n} + 76$
- n^2
- 5
- $2^n + n + \log n$
- $T^B_{contains}(n) = 4$
- $T^W_{contains}(n) = 2n + 4$
- $T^W_{insort}(n) = 1.5n^2 + 4.5n - 4$

A More Elaborate Example

Suppose method $q()$ calls another method `int k(int i)` with $T_k(m) = O(\log(m))$ for some m independent of i.

```
public void q()
{
    int c = 0;
    for (int i = 1; i < n + 1; i++)
        c = c + k(i);
}
```

Determine the time complexity (i.e. the time execution time as a function of input size expressed in Big-Oh notation) of this method using the active operation approach.

- What is the active operation?
- How many times does it execute?
- What is the cost of the active operation?
- What is the total amount of time required expressed in Big-Oh notation.

Exercise 5

Again, assume method `k` requires $O(\log m)$ time, m independent of i and n.

```
public void s()
{
     int x = 0;
     for (int i = 0; i < n + 1; i++)
          x = x + k(i)*k(2*n - i);
     System.out.println(x);
}
```

- What is the active operation?
- How many times does it execute?
- What is the cost of the active operation?
- What is the total amount of time required expressed in Big-Oh notation.

Exercise 6

```java
static public int f(int n)
{
      int result = 0;
      double p =n;
      while (p > 1)
      {
            p = p/2;
            result = result + 1;
      }
      return result;
}
```

What is the time complexity of the method?

Combining Growth Functions

- Since $O(f(n))$ and $O(g(n))$ are sets, we can use these set properties and the definition of Big-Oh to show:

$$
\begin{aligned}
O(f(n)) + O(g(n)) &= O(f(n) + g(n)) \\
&= O(2 \cdot \max(f(n), g(n))) \\
&= O(\max(f(n), g(n))) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
kO(f(n)) &= O(kf(n)) = O(f(n)) \text{ for any constant } k.
\end{aligned}
$$

- Same holds for Big-Theta. We will need at least one of these "rules" to solve our next problem.

Exercise 7

```
public void r()
{
     q();
     for (int i = 1; i < m; i = 2*i)
          p();
     s();
}
```

- Assuming that we know that:
  - $T_q(n, m) \in O(n \log m)$
  - $T_p(n) \in O(n^2)$
  - $T_s(m) \in O(m \log m)$

  What is the time complexity of the method `r`? Use the active operation approach. There are three candidate active operations, what are they? Determine time required by all three; the one that requires the most time is reflective of the overall time complexity.

Exercise 8

- Now we have the tools to return to our list class and consider the time complexity of our list operations.
- Determine the worst-case time complexity of all of the methods in the LinkedList class.
- Are there any methods where the time complexity could be made better?

Do Growth Rates Matter?

- Does the growth rate make a difference, or are computers so fast that any algorithm can be done quickly?
- Some algorithms take huge amounts of time on even small problems.
- Linear vs. quadratic alone is big difference.

Do Growth Rates Matter?

Time to execute $f(n)$ operations at 10^6 operations per second.

n	$\log_2 n$	n	$n \log_2 n$	n^2	2^n
10	$3.3 \times 10^{-6}s$	$1.0 \times 10^{-5}s$	$3.3 \times 10^{-5}s$	$1.0 \times 10^{-4}s$	$1.0 \times 10^{-3}s$
100	$6.6 \times 10^{-6}s$	$1.0 \times 10^{-4}s$	$6.6 \times 10^{-4}s$	$1.0 \times 10^{-2}s$	$4.0 \times 10^{16}y^*$
1000	$9.97 \times 10^{-6}s$	$1.0 \times 10^{-3}s$	$9.97 \times 10^{-3}s$	$1.0s$	
10^5	$1.66 \times 10^{-5}s$	$0.1s$	$1.66s$	$2.78h$	
10^6	$1.99 \times 10^{-5}s$	$1s$	$19.9s$	$11.57d$	
10^7	$2.33 \times 10^{-5}s$	$10s$	$233s$	$3.17y$	

Do Growth Rates Matter?

Time to execute $f(n)$ operations at 10^6 operations per second.

n	$\log_2 n$	n	$n \log_2 n$	n^2	2^n
10	$3.3 \times 10^{-6}s$	$1.0 \times 10^{-5}s$	$3.3 \times 10^{-5}s$	$1.0 \times 10^{-4}s$	$1.0 \times 10^{-3}s$
100	$6.6 \times 10^{-6}s$	$1.0 \times 10^{-4}s$	$6.6 \times 10^{-4}s$	$1.0 \times 10^{-2}s$	$4.0 \times 10^{16}y^*$
1000	$9.97 \times 10^{-6}s$	$1.0 \times 10^{-3}s$	$9.97 \times 10^{-3}s$	$1.0s$	
10^5	$1.66 \times 10^{-5}s$	$0.1s$	$1.66s$	$2.78h$	Why
10^6	$1.99 \times 10^{-5}s$	$1s$	$19.9s$	$11.57d$	bother?
10^7	$2.33 \times 10^{-5}s$	$10s$	$233s$	$3.17y$	

* Est. age of universe: $1.3 * 10^{10}$ years.

Next Class

- Next class reading: Chapter 4: Iteration and Cursors