Section 2

Thursday, October 11, 2018 3:07 PM

quickcheck 02

Quickcheck 02: Asymptotic Analysis

Name:

Definition: Dominated by

A function $f(n)$ is dominated by $g(n)$ when...

- There exists two constants $c > 0$ and $n_0 > 0$...
- Such that for all values of $n \geq n_0$...
- $f(n) \leq c \cdot g(n)$ is true.

Demonstrate that $2n^3 - 3 + 9n^2 + \sqrt{n}$ is dominated by n^3 by finding a c and n_0. Show your work.

$2n^3 \le 2n^3$ when $n \ge 5$ — $c = 2$, $n_0 = 5$

$-3 \le 1 \cdot n^3$ when $n \ge 0$ — $c = 1$, $n_0 = 0$

$2n^3 \le 2n^3$ when $n \ge 5$
$-3 \le n^3$ when $n \ge 0$
$9n^2 \le 9n^3$ when $n \ge 1$
$\sqrt{n} \le n^3$ when $n \ge 0$

when $n \ge 5$

$2n^3 - 3 + 9n^2 + \sqrt{n} \le 13n^3$

Another question

Do you have any questions about this course? It could be about policy, content, instructors, TAs, etc.

$9n^2 \le 9n^3$ for $n \ge 1$ — $c = 9$, $n_0 = 1$

$\sqrt{n} \le n^3$ when $n \ge 1$ — $c = 1$, $n_0 = 1$

section02

Section 02: Asymptotic Analysis

Section Problems

1. Comparing growth rates

(a) Order each of the following functions from fastest to slowest in terms of asymptotic growth. (By "fastest", we mean which function increases the most rapidly as n increases.)

- $\log_4(n) + \log_2(n)$
- $\frac{n}{2} + 4$
- $2^n + 3$
- $750,000,000$
- $8n + 4n^2$

$750,000,000 \in O(\log_4 n + \log_2 n) \in O(\frac{n}{2} + 4) \in O(8n + 4n^2) \in O(2^n + 3)$

$O(1)$, $O(\log n)$, $O(n)$, $O(n^2)$, $O(2^n)$

(b) For each of the above expressions, state the simplified tight $\mathcal{O}$ bound in terms of n.

(c) Order each of these more esoteric functions from fastest to slowest in terms of asymptotic growth. (By "fastest", we mean which function increases the most rapidly as n increases.) Also state a simplified tight $\mathcal{O}$ bound for each.

- $2^{n/2}$
- 3^n
- 2^n

2. True or false?

(a) In the worst case, finding an element in a sorted array using binary search is $\mathcal{O}(n)$.

(b) In the worst case, finding an element in a sorted array using binary search is $\Omega(n)$.

(c) If a function is in $\Omega(n)$, then it could also be in $\mathcal{O}(n^2)$.

(d) If a function is in $\Theta(n)$, then it could also be in $\mathcal{O}(n^2)$.

(e) If a function is in $\Omega(n)$, then it is always in $\mathcal{O}(n)$.

$O(\log(n)) \in O(n)$
$O(\log(n)) \in \Omega(n)$

counter-example!

3. Modeling code

For each of the following code blocks, give a summation that represents the worst-case runtime in terms of n.

(a)

```
int x = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        x++;
    }
}
```

(b)

```
int x = 0;
for (int i = n; i >= 1; i /= 2) {
    x += i;
}
```

This is 3a: $0 + 1 + 2 + \dots + (n-2) + (n-1) \rightarrow \frac{n(n+1)}{2} - n \rightarrow \Theta(n^2)$

$0 + 1 + 2 + \dots + n-2 + n-1 + n$; $\frac{n(n+1)}{2} \in \Theta(n^2)$

This term is missing so subtract it off (doesn't change final Θ result though!)

This is 3b: $\log(n)$

4. Finding bounds

For each of the following code blocks, construct a mathematical function modeling the worst-case runtime of the code in terms of n. Then, give a tight big-$\mathcal{O}$ bound of your model.

(a)

```
int x = 0;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n * n / 3; j++) {
        x += j;
    }
}
```

(b)

```
int x = 0;
for (int i = n; i >= 0; i -= 1) {
    if (i % 3 == 0) {
        break;
    } else {
        x += n;
    }
}
```

(c)

```
int x = 0;
for (int i = 0; i < n; i++) {
    if (i % 5 == 0) {
        for (int j = 0; j < n; j++) {
            if (i == j) {
                x += i * j;
            }
        }
    }
}
```


(d)

```
int x = 0;
for (int i = 0; i < n; i++) {
    if (n < 100000) {
        for (int j = 0; j < n; j++) {
            x += 1;
        }
    } else {
        x += 1;
    }
}
```

(e)

```
int x = 0;
if (n % 2 == 0) {
    for (int i = 0; i < n * n * n * n; i++) {
        x++;
    }
} else {
    for (int i = 0; i < n * n * n; i++) {
        x++;
    }
}
```

5. Applying definitions

For each of the following, choose a c and n_0 which show $f(n) \in \mathcal{O}(g(n))$. Explain why your values of c and n_0 work.

(a) $f(n) = 3n + 4, g(n) = 5n^2$

(b) $f(n) = 33n^3 + \sqrt{n} - 6, g(n) = 17n^4$

(c) $f(n) = 17\log(n), g(n) = 32n + 2n\log(n)$

6. Memory analysis

For each of the following functions, construct a mathematical function modeling the amount of memory used by the algorithm in terms of n. Then, give a Θ bound of your model.

(a)

```
List<Integer> list = new LinkedList<Integer>();
for (int i = 0; i < n * n; i++) {
    list.insert(i);
}
Iterator<Integer> it = list.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```

(b)

```
int[] arr = {0, 0, 0};
for (int i = 0; i < n; i++) {
    arr[0]++;
}
```


(c)

```
ArrayDictionary<Integer, String> dict = new ArrayDictionary<>();
for (int i = 0; i < n; i++) {
    String curr = "";
    for (int j = 0; j < i; j++) {
        for (int k = 0; k < j; k++) {
            curr += "?";
        }
    }
    dict.put(i, curr);
}
```

Note: for simplicity, assume the dictionary has an internal capacity of exactly n.

Food for thought

7. Analyzing recursive code

For each of the following recurrences, use any of the methods discussed in class to find a Θ bound on the function in terms of n.

(a) $T(n) = \begin{cases} 1 & \text{if } n = 0 \\ T(n-1) + 3 & \text{otherwise} \end{cases}$

(b) $T(n) = \begin{cases} 10 & \text{if } n = 1 \\ T(n/2) + 4 & \text{otherwise} \end{cases}$

(c) $T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n/3) + n & \text{otherwise} \end{cases}$

8. Application: TreeMap implemented as a Binary Search Tree

Consider the following method, which is a part of a Binary Search Tree implementation of a TreeMap class.

```
public V find(K key) {
    return find(this.root, key);
}

private V find(Node<K, V> current, K key) {
    if (current == null) {
        return null;
    }
    if (current.key.equals(key)) {
        return current.value;
    }
    if (current.key.compareTo(key) > 0) {
        return find(current.left, key);
    } else {
        return find(current.right, key);
    }
}
```

(a) We want to analyze the runtime of our `find(x)` method in the best possible case and the worst possible case. What does our tree look like in the best possible case? In the worst possible case?

(b) Write a recurrence to represent the worst-case runtime for `find(x)` in terms of n, the number of elements contained within our tree. Then, provide a Θ bound.

(c) Assuming we have an optimally structured tree, write a recurrence for the runtime of `find(x)` (again in terms of n). Then, provide a Θ bound.

9. LRU Caching

When writing programs, it turns out to be the case that opening and loading data in files can be a very slow process. If we plan on reading information from those files very frequently (for example, if we want to implement a database), what we might want to do is *cache* the data we loaded from the files – that is, keep that information in-memory.

That way, if the user requests information already present in our cache, we can return it directly without needing to open and read the file again.

However, computers have a much smaller amount of RAM then they have hard drive space. This means that our cache can realistically contain only a certain amount of data. Often, once we run out of space in our cache, we get rid of the items we used the *least recent*. We call these caches **Least-Recently-Used (LRU)** caches.

Discuss how you might apply or adapt the ADTs and data structures you know so far to develop an LRU cache. Your data type should store the most recently used data, and handle the logic of whether it can find the data in the cache, or if it needs to read it from the disk. Assume you have a helper function that handles fetching the data from disk.

Your cache should implement our `IDictionary` interface and optimize its operations with the LRU caching strategy. After you've decided on a solution, describe the tradeoffs of your structure, possibly including a worst-case and average-case analysis.