COMP 3760

Algorithm Analysis and Design

Lesson 4: Algorithm Efficiency

Rob Neilson

rneilson@bcit.ca

Question 1 (example):

- solve $$\sum_{i=1}^{n} 1$$

- ans

  = 1 + 1 + 1 + ... + 1 (n times)
  = 1*n = n

Question 1

- Who is the ex-Nirvana drummer who went on to form the Foo Fighters?
  - *ans: Dave Grohl*

Question 2

(a) what is a “RAM machine”

- hypothetical computer where all memory accesses are constant time and all instructions take the same amount of time (one time unit)

(b) why do we concern ourselves with RAM machines when discussing algorithm complexity?

- gives us a common model in which we can compare different algorithms

- if we did not assume a RAM machine, performance would be machine dependent

Question 3

(a) how do we measure the “time-efficiency” of an algorithm?

- count the total number of instructions used by an algorithm

(b) how do we measure the “space-efficiency” of an algorithm?

- count the total number of bytes in memory (RAM) used by an algorithm

(c) why do we concern ourselves time rather than space efficiency?

- RAM is relatively cheap
- RAM is typically not the limiting factor in most of our algorithms
  - because ...
- it is the size of the input (ie: need to process each input item) that makes it slow, and we will run out time (ie: run too slow) before we run out of RAM

Question 4

- what does it mean if an algorithm is said to be a member of the set $O(n^2)$
  - the time it takes to execute the algorithm will be no worse than $\mathbf{c \cdot n^2 + b}$ (where c and b are constants and n is the size of the input)

Question 5

- Give an informal mathematical definition of "big-oh"

Definition:

- a function **f(n)** is in the set **O(g(n))** [*denoted: $f(n) \in O(g(n))$*] if there is a constant **c** and a positive integer $\mathbf{n_0}$ such that

$$f(n) \le c * g(n) \text{ , for all } n \ge n_0$$

- ie: f(n) is bounded above by some constant multiple of g(n)

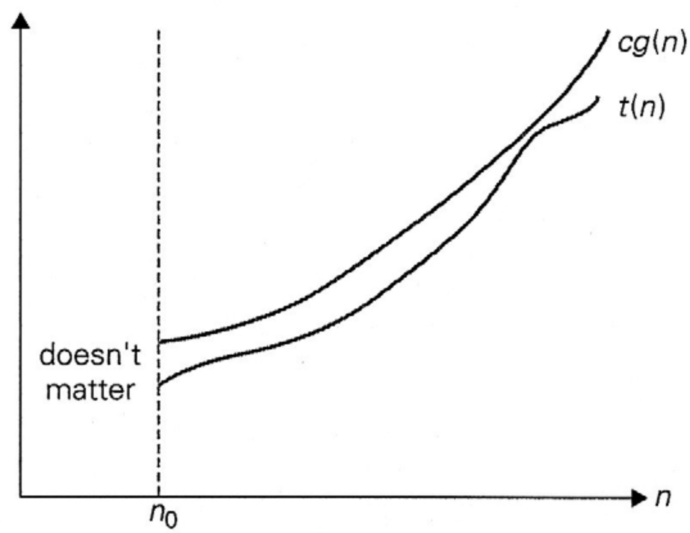

FIGURE 2.1 Big-oh notation: $t(n) \in O(g(n))$

Question 6

- what is big-omega - $\Omega(n)$
  - the best case efficiency class of an algorithm

$$f(n) \geq c * g(n) \text{ , for all } n \geq n_0$$

Question 7

- what is big-theta - $\Theta(n)$

Definition:

- a function **f(n)** is in the set **$\Theta(g(n))$** [*denoted: $f(n) \in \Theta(g(n))$*] if there is some constants **c_1 and c_2**, and a positive integer **n_0** such that

$$c_2\, g(n) \le f(n) \le c_1\, g(n) \text{ , for all } n \ge n_0$$

- ie: f(n) is bounded both above and below by constant multiples of g(n)

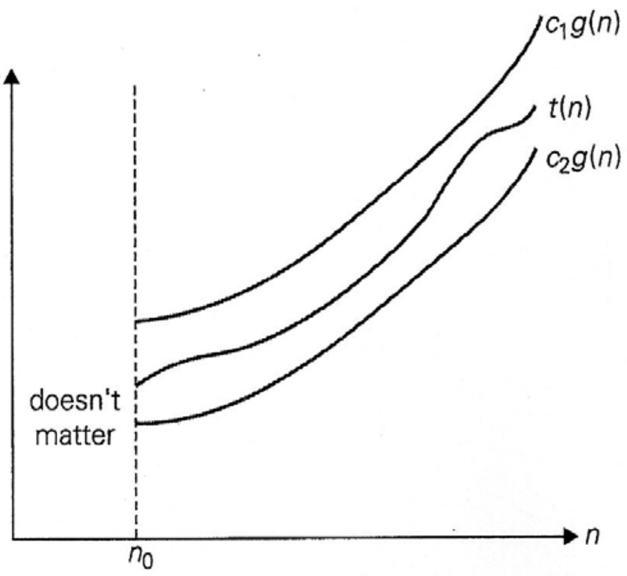

FIGURE 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Question 8

- solve $$\sum_{i=1}^{n} 3n$$

$$= 3 \cdot \sum_{i=1}^{n} n = 3 \cdot (n + n + \ldots + n) = 3n \cdot \underbrace{(1 + 1 + \ldots + 1)}_{n-times} = 3n \cdot n = 3n^2$$

Question 9

- solve

$$\sum_{i=0}^{n-1}(6n+3)$$

$$=\sum_{i=0}^{n-1}6n+\sum_{i=0}^{n-1}3=6n\sum_{i=0}^{n-1}1+3\sum_{i=0}^{n-1}1=6n(n)+3(n)=6n^2+3n=3n(2n+1)$$

Question 10

- what is the “basic operation” of an algorithm?
  - the fundamental operation in the algorithm that contributes the most to the overall running time of the algorithm

Question 11

- consider the following algorithm

(a) what is the total count of the number of operations (all operations)

```
x ← 0                        1 time
i ← 1                        1 time
while i ≤ 2*n do             2n+1 times
    x ← x + 1                2n times
    i ← i + 1                2n times

                             = 3(2n)+3 = 6n+3
```

(b) what is the basic operation in this algorithm

- it is going to be either addition or comparison – depending on which is the most important instruction in the algorithm

- since this algorithm is simple computing the sum from 1 to 2n of 1, addition is the basic op

Question 12

- assume an algorithm executes is basic instruction 16n+7 times, where n is the input size
- what is the worst case efficiency class for this algorithm
  - O(n)

Question 13

- algorithm efficiency depends on the size of the input as well as the order of the input
- give an example of an algorithm that will always run faster on a specific ordering of it's input
  - linear search for a value will always run faster if the value is near the beginning of the input

Question 14

- Assume an algorithm executes it's basic instruction $7n^2+12n+2^3$ times

- What big-oh class does this algorithm belong to?

  - $O(n^2)$

Question 15

- is 2n+17 $\in$ O(n^2)
  - yes, the function n^2 is an upper bound on the performance (but it is not the "best upper bound")
  - we can answer this question by applying the definition …

$$
\begin{aligned}
&2n + 17 \le cn^2 \\
&\frac{2}{n} + \frac{17}{n^2} \le c \\
&n = 1 \Rightarrow c = 19 \\
&n = 2 \Rightarrow c = 5.25 < 19 \\
&etc
\end{aligned}
$$

Question 16

- is $2n+17 \in \Omega(n^2)$
  - no!

$$2n + 17 \geq cn^2$$

$$\frac{2}{n} + \frac{17}{n^2} \geq c$$

  - C decreases as n increases, so there is always an n tht gives a value lower than any c we choose
  - $\Omega(n^2)$ says "it can not do better than n^2" – but we know it can run in O(n) ops – so the assertion is not true

Question 17

- is $8^{n+2} \in O(2^n)$
  - yes!

$$8^{n+2} = 2 \cdot 2 \cdot 2^{n+2} = (2 \cdot 2) \cdot (2 \cdot 2 \cdot 2^n) = 16 \cdot 2^n$$

Question 18

- order the following efficiency classes in order of relative efficiency (from best to worst)

  n log n
  log n
  n
  n!
  n^3
  2^n
  n^2

Answer:

logn
n
nlogn
n^2
n^3
2n
n!

Question 19

- does $f(n) \in O(\ g(n)\ )$ imply $g(n) \in O(\ f(n)\ )$
- give an example to support your claim

Answer: No

Example: assume $f(n) = n^2$, $g(n) = n^3$

$n^2 \in O(n^3)$ is true, however

$n^3 \in O(n^2)$ is false

Question 20

- use the informal definition of big-oh to show:

$3n^2 + 42 \in O(n^2)$

Answer:

by defn of big O, $3n^2 + 42 \leq c \cdot n^2$

solving for c: $3 + 42/n^2 \leq c$

setting n=1 gives $3+42 \leq c$

we notice that $42/n^2$ gets smaller as n increases, therefore:

$3n^2 + 42 \in O(n^2)$ must be true for $c=45$, $n \geq 1$

Question 21

- What is the best big-oh class for

  2 + 4 + 6 + ... + 2n

Answer:

$$2+4+6+...+2n = 2(1+2+3+..+n) =$$

$$2 \cdot \sum_{i=1}^{n} i = 2 \cdot \frac{n(n+1)}{2} = n^2 + n \in O(n^2)$$

Question 22

- What is the best big-oh class for

  $2 + 4 + 8 + 16 + \ldots + 2^n$

Answer:

$$2 + 4 + 8 + 16 + \ldots + 2^n = 2(1 + 2 + 4 + 8 + .. + 2^{n-1}) = 2 \cdot \sum_{i=0}^{n-1} 2^i$$

$$\text{we know that:} \quad \sum_{i=0}^{n} c^i = \frac{c^{n+1} - 1}{c - 1} \text{ which means } \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\text{therefore } 2 \cdot \sum_{i=0}^{n-1} 2^i = 2 \cdot (2^{(n-1)+1} - 1) = 2^{n+1} - 2 \in O(2^n)$$

Question 23

```
for i = 0 to n do
   for j = 1 to 10 do
         x ← x + 1
```

(a) what is the basic operation

addition

(b) how many times is the basic operation executed

$$\sum_{i=0}^{n}\sum_{j=1}^{10} 1 = \sum_{j=0}^{n} 10 = 10(n+1) = 10n + 10$$

(c) what is the efficiency class of this algo

O(n)

Question 24

```
for i = 1 to n do
    for j = 1 to i do
        for k = 1 to i do
            x[j,i] ← x[k,j]
```

(a) what is the basic operation

assignment

(b) how many times is the basic operation executed

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{i}1 = \sum_{i=1}^{n}\sum_{j=1}^{i} i = \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

(c) what is the efficiency class of this algo

$O(n^3)$

Question 25

- what is the worst case efficiency of this bit of code?

```
int[] v = new int[n];
for (int i = 0; i < n; i++)
      v[i] = i;
```

$$\sum_{i=0}^{n-1} 1 = n - 1$$

Question 26

(a) Write the pseudocode for an algorithm for checking whether two given words are anagrams
- i.e., whether one word can be obtained by permuting the letters of the other.
- For example, the words *tea* and *eat* are anagrams.

(b) What is the basic operation in this algorithm?

(c) How many times is the basic operation executed?

(d) What is the efficiency class of this algorithm?

Question 26: Answers

Technique #1: Brute Force

```
for each letter in word 1
   search word 2 for the letter
   if found, delete the letter from word 2
```

efficiency class = $O(n^2)$

Technique #2: Transform & Conquer

```
sort word 1
sort word 2
use a linear compare of the 2 sorted words
```

efficiency class = same as sorting algo used

Technique #3: Transform & Conquer

```
create letter vector for word 1
create letter vector for word 2
use a linear compare of the letter vectors
```

efficiency class = $O(n)$