Homework1 for EECS 340

Yu Mi

January 25, 2018

1 Warm-up: Big-Oh and Counting Primitive Operations

Show your work on the following questions. Use the limit-based definitions of asymptotic notation on the "Big-Oh Cheat Sheet" on Canvas wherever applicable.

1.1 Solve R-1.20, R-1.22, and R-1.23 in the text

1.1.1 R-1.20

Show that $(n+1)^5$ is $O(n^5)$.

Proof: Let $f(n) = (n+1)^5$ and $g(n) = n^5$ so that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{(n+1)^5}{n^5} = \lim_{n\to\infty} \frac{n^5 + 5n^4 + 10n^3 + 10n^2 + 5n + 1}{n^5}$$
$$= 1 + \lim_{n\to\infty} \frac{5}{n} + \frac{10}{n^2} + \frac{10}{n^3} + \frac{5}{n^4} + \frac{1}{n^5} = 1$$

Since $0 \le 1 < \infty$, $(n+1)^5$ is $O(n^5)$.

1.1.2 R-1.22

Show that n is $o(n \log n)$.

Proof: Let $f(n) = n$ and $g(n) = n \log n$ so that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{n \log n} = \lim_{n\to\infty} \frac{1}{\log n} = 0$$

Since $0 = 0$, n is $o(n \log n)$.

1.1.3 R-1.23

Show that n^2 is $\omega(n)$.

Proof: Let $f(n) = n^2$ and $g(n) = n$ so that

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^2}{n} = \lim_{n\to\infty} n = \infty$$

Since $\infty = \infty$, n^2 is $\omega(n)$.

1.2 Intuitively, $2^x \in O(3^x)$, since 3^x grows faster. Is $3^x \in O(2^x)$?

Answer: No, proof as follows:

Let $f(x) = 3^x$ and $g(x) = 2^x$ so that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{3^x}{2^x} = \lim_{x\to\infty} (\frac{3}{2})^x = \infty$$

Since $\infty = \infty$, so that 3^x is $\omega(2^x)$, $3^x \notin O(2^x)$

1.3 Intuitively, $\log_3(x) \in O(\log_2(x))$. Is $\log_2(x) \in O(\log_3(x))$?

Answer: Yes, proof as follows:

Let $f(x) = \log_2(x)$ and $g(x) = \log_3(x)$ so that

$$\lim_{x\to\infty} \frac{\log_2(x)}{\log_3(x)} = \lim_{x\to\infty} \frac{\frac{\ln x}{\ln 2}}{\frac{\ln x}{\ln 3}} = \lim_{x\to\infty} \frac{\ln 3}{\ln 2} = \log_2(3)$$

Since $0 \le \log_2(3) < \infty$, $\log_2(x)$ is $O(\log_3(x))$.

1.4 Use summations to derive tight asymptotic bounds ($\Theta(-)$) on the runtime of each algorithm

1.4.1 R-1.12

Answer: First, we need to rewrite this algorithm into *while* loop:

Algorithm Loop2(n):

$p \leftarrow 1$ ▷ 1 unit of time
$i \leftarrow 1$ ▷ 1 unit of time
while $i \le 2n$ **do** ▷ $2n + 1$ units of time
  $p \leftarrow p \times i$ ▷ $2 \times 2n$ units of time
  $i \leftarrow i + 1$ ▷ $2 \times 2n$ units of time
end while

As is described in the comments of the algorithm, the run time of this algorithm should be $\Theta(10n + 3)$ units of time. So that this algorithm is $\Theta(n)$

1.4.2 R-1.14

Answer: First, we need to rewrite this algorithm into *while* loop:

Algorithm Loop4(n):

$s \leftarrow 0$ ▷ 1 unit of time
$i \leftarrow 1$ ▷ 1 unit of time
while $i \le 2n$ **do** ▷ $2n + 1$ units of time
  $j \leftarrow 1$ ▷ $2n$ units of time
  while $j \le i$ **do** ▷ $(i + 1) \times 2n$ units of time
    $s \leftarrow s + i$ ▷ $2 \times i \times 2n$ units of time
    $j \leftarrow j + 1$ ▷ $2 \times i \times 2n$ units of time
  end while
  $i \leftarrow i + 1$ ▷ $2n$ units of time
end while

To calculate the time cost of the inner loop, we need to focus on the value of i which changes with the outer loop. To make calculate easy to understand, we define C_1 as the actual units of time the outer loop will cost and C_2 as the actual units of time the inner loop will cost. So that:

$$C_1 = 1 + 1 + 2n + 1 + 2n + 2n = 6n + 3$$

$$C_2 = 2n \sum_{i=1}^{2n} (i + 1 + 2i + 2i) = 10n^2 + 7n$$

To sum up, the total units of time this algorithm will cost should be $time_{total} = C_1 + C_2 = 10n^2 + 13n + 3$. So that this algorithm is $\Theta(n^2)$

1.4.3 R-1.15

Answer: First, we need to rewrite this algorithm into *while* loop:

Algorithm Loop5(n):

$s \leftarrow 0$	$\triangleright$ 1 unit of time
$i \leftarrow 1$	$\triangleright$ 1 unit of time
while $i \leq n^2$ **do**	$\triangleright$ $n^2 + 1$ units of time
$j \leftarrow 1$	$\triangleright$ n^2 units of time
while $j \leq i$ **do**	$\triangleright$ $(i+1) \times n^2$ units of time
$s \leftarrow s + i$	$\triangleright$ $2 \times i \times n^2$ units of time
$j \leftarrow j + 1$	$\triangleright$ $2 \times i \times n^2$ units of time
end while	
$i \leftarrow i + 1$	$\triangleright$ n^2 units of time
end while	

To calculate the time cost of the inner loop, we need to focus on the value of i which changes with the outer loop. To make calculate easy to understand, we define C_1 as the actual units of time the outer loop will cost and C_2 as the actual units of time the inner loop will cost. So that:

$$C_1 = 1 + 1 + n^2 + 1 + n^2 + n^2 = 3n^2 + 3$$

$$C_2 = 2n \sum_{i=1}^{n^2} (i + 1 + 2i + 2i) = \frac{5}{2}n^4 + \frac{7}{2}n^2$$

To sum up, the total units of time this algorithm will cost should be $time_{total} = C_1 + C_2 = \frac{5}{2}n^4 + \frac{13}{2}n^2 + 3$. So that this algorithm is $\Theta(n^4)$

1.5 Explain why it is reasonable to ignore the overhead of a ranged *for* loop when you derived the tight asymptotic runtime bounds in the previous question.

Answer: When calculating the runtime bounds of a loop, the overhead of such loop will cost a constant time of computation, while the loop body will cost n times more than the overhead. As n grows big enough, the overhead is always significantly smaller than the loop body. Thus, when we are deriving the asymptotic runtime bounds, we can ignore the overhead of a ranged loop.