CPSC 320 2023S2: Tutorial 2 Solutions

1 Asymptotic Notation: Pattern Matching

Algorithms that find occurrences of a pattern in a text are used in myriad applications, including text processing as well as detecting interesting motifs (patterns) in genomes (texts). Let $P[1..k]$ and $T[1..n]$ be strings, called the pattern and text, respectively. We say that P occurs at offset o in T if $P[1..k] = T[1 + o..k + o]$. The following algorithm finds the offsets of all occurrences of a pattern in a text:

```
Algorithm Find-Pattern ($P[1..k], T[1..n]$)
    // Output the list of all offsets of occurrences of pattern $P$ in text $T$
    Offset-List ← null              // initially the list of offsets is empty
    For $o$ from 0 to $n - k$
        // Does $P$ occur at offset $o$ of $T$?
        #Matches = 0
        For $j$ from 0 to $k - 1$
            If $T[1 + o + j] == P[1 + j]$
                #Matches++
        If #Matches = $k$
            Add $o$ to Offset-List
    Return Offset-List
```

To analyze the running time of this algorithm, we'll extend big-O notation to functions with two parameters. Specifically, let $T(n, k)$ be the worst-case running time of a given algorithm whose input size is described using a pair (n, k) of nonnegative integers. We say that $T(n, k)$ *is big-O of function* $f(n, k)$ and write $T(n, k) = O(f(n, k))$ if there are nonnegative integers c, n_0, and k_0 such that for all inputs of size (n, k) with $n \geq n_0$ and $k \geq k_0$, $T(n, k) \leq cf(n, k)$.

As a function of n and k, what terms below describe the worst-case running time of Algorithm Find-Pattern? Check all answers that apply. (Make sure to write down a brief justification for your choices, and don't forget to do that also in future problems and in the assignment.)

SOLUTION: $\Theta((n-k+1)k)$. The outer and inner For loops have $n-k+1$ iterations and k iterations, respectively, and the other lines of code take $\Theta(1)$ time. So the total time is $\Theta((n-k+1)k)$, which is the third option from the left. The function $(n-k+1)k = O(nk)$, but for $k = n-1$, $(n-k+1)k = 2k \leq 2n$, and so in this case $(n-k+1)k$ is not $\Omega(nk)$. So the rightmost option does not hold. For $k = 1$, the first option is $\Theta(1)$ and for $k = n$ the second option is $\Theta(1)$, and so neither of the leftmost two options are $\Omega((n-k)k)$. So these two options are not selected.

2 Faster or Slower

For this problem, consult the accompanying handout on little-o notation. Suppose that some algorithm A has running time $f(n)$ and that algorithm B has running time $g(n)$, on all inputs of size n. Assume that f and g are functions $\mathbb{N} \to \mathbb{N}^+$, and that $\lim_{n\to\infty} f(n)$ and $\lim_{n\to\infty} g(n)$ are both infinity. (Here, $\mathbb{N}$ is the set of nonnegative integers and $\mathbb{N}^+$ is the set of positive integers.)

We say that algorithm A *is eventually faster than* B if on all sufficiently large inputs, the running time of A is less than the running time of B. That is, $f(n) < g(n)$ for all sufficiently large n.

Explain whether each of the following statements is true or false.

1. For some choice of g such that $g(n) \in o(f(n) \log n)$, A is eventually faster than B.

   SOLUTION: True. Choosing $g(n) = 2f(n)$ satisfies the condition that $g(n) \in o(f(n) \log n)$. For this choice, $g(n) > f(n)$ for all n, and so A is in fact faster than B on all inputs.

2. For some choice of g such that $g(n) \in o(f(n) \log n)$, B is eventually faster than A.

   SOLUTION: True. Let $f(n) \geq 2$ for all $n \geq n_1$. Choosing $g(n) = \lfloor f(n)/2 \rfloor$ for $n \geq n_1$ satisfies the condition that $g(n) \in o(f(n) \log n)$. For this choice, $g(n) < f(n)$ for all $n \geq n_1$, and so B is faster than A on all inputs of size $n \geq n_1$.

3 Algorithms on Numbers

Let x and y be two nonnegative integers, represented in binary. Assume for concreteness that there are no trailing zero's in the binary representation; for example the number three is represented as 11 and not as 00011. Assume furthermore for this problem that the lengths of x and y are exactly the same. (If you prefer, you can assume that the numbers are represented in decimal and answer the problem for for decimal addition and division operations, it won't change the answer.) Here's an algorithm for computing $\max\{x, y\}$:

Algorithm Find-Max(x, y): Repeatedly compare two bits of x and y, from high-order bits to low-order bits, until one of the following occurs:

(a) The two compared bits differ. If the bit of x is 1 and that of y is 0, output x, otherwise output y.

(b) The end of both numbers is reached, and all compared pairs are the same. In this case, x must equal y, and so output either x or y.

1. Give a tight big-O bound on the running time of the Find-Max algorithm, as a function of x and y. Hint: What is the length of the binary representation of x? Make sure to distinguish between the inputs x and y and the input sizes, which in this case is the length of their binary representations.

   SOLUTION: The binary representations of x and y have length $\lfloor \log_2 x \rfloor + 1$ and $\lfloor \log_2 y \rfloor + 1$, respectively. So the Find-Max algorithm examines at most $\lfloor \log_2 x \rfloor + \lfloor \log_2 y \rfloor + 2$ bits, and examines exactly this many bits for example when $x = y$. When the lengths of x and y are the same, the time to compare a pair of bits and determine whether to stop and produce the output, or to continue, is $O(1)$.

   So a tight big-O bound on the running time of Find-Max is $O(\log x + \log y)$.

   (Note, when using big-O notation, we need not specify the base of the logarithm, since $\log x$ with respect to two different constant bases differs by a constant.)

2. Describe an infinite set of pairs (x, y) for which the running time of Find-Max is $O(1)$.

   SOLUTION: The running time of Find-Max is $O(1)$ on any pair in the infinite set

   $$\{(x, y) \mid x = 2^k - 1 \text{ and } y = 2^{k-1}\},$$

   since on the second iteration, the bits are different and the algorithm outputs x.

3. Which categories does the running time of the Find-Max algorithm belong to? (Choose as many as apply.)

   SOLUTION:

● $O(xy)$	● $O(x+y)$	● $O(\log x \log y)$	● $O(\log x + \log y)$
○ $\Omega(xy)$	○ $\Omega(x+y)$	○ $\Omega(\log x \log y)$	○ $\Omega(\log x + \log y)$
○ $\Theta(xy)$	○ $\Theta(x+y)$	○ $\Theta(\log x \log y)$	○ $\Theta(\log x + \log y)$
● $o(xy)$	● $o(x+y)$	● $o(\log x \log y)$	○ $o(\log x + \log y)$

   Brief justification: In part 1 of this problem we obtained a tight big-O bound of $O(\log x + \log y)$. All of the big-O bounds in the first row follow since $\log x + \log y$ is upper-bounded by the three other functions given.

In part 2 of this problem we saw that for infinitely many inputs, the running time is $O(1)$, so none of the big-Ω bounds in row 2 apply, and consequently none of the Θ bounds in row 3 apply.

Finally, $\log x + \log y$ grows asymptotically more slowly than the first three functions of row 4.