CS:3330 Exam 1, Fall 2015

Thursday, Sept 24 2015, 6:30 pm to 8:30 pm

---

1. Here are two problems on understanding the growth rate of functions that represent running times of algorithms and the use of asymptotic notation.

   (a) Take the following list of functions (from nonnegative integers to nonnegative integers) and arrange them in ascending order of growth. Thus, if a function g immediately follows f in the list, then $f = O(g)$. Some of these functions are described in words and some as summations. For every function described in words or as a summation, write it in standard form first before placing it in the sorted list of functions. Show your work in order to receive partial credit.

   (i) $2^{5 \log_2 n}$

   (ii) The running time of the `binarySearch` algorithm.

   (iii) $100n^2 + 1000000$

   (iv) $\sqrt{2^n}$

   (v) $(\log_2 n)^2 \cdot \sum_{i=1}^{n} \Theta(1)$

   (vi) $n^3/(\log_2 n)^4$

   (vii) $2^{\sqrt{\log_2 n}}$

   (viii) 100

   Solution: The correct ordering of functions by asymptotic growth rate is:

   (viii) 100

   (ii) The running time of the `binarySearch` algorithm. This is $\Theta(\log n)$ in the worst case.

   (vii) $2^{\sqrt{\log_2 n}}$. Note that this function is sublinear (i.e., grows more slowly than the linear function) because $2^{\log_2 n} = n$.

   (v) $(\log_2 n)^2 \cdot \sum_{i=1}^{n} \Theta(1)$. The sum is $\Theta(n)$ and so this function is $\Theta(n \log^2 n)$.

   (iii) $100n^2 + 1000000$

   (vi) $n^3/(\log_2 n)^4$

   (i) $2^{5 \log_2 n}$. This can be rewritten as n^5.

   (iv) $\sqrt{2^n}$. This can be rewritten as $2^{n/2}$.

   (b) For each statement below, write down if it is `True` or `False`. Provide a 1-2 sentence justification for your answer.

   (i) $100n^2 + 10n + 15 = \Theta(n^3)$.
   Solution: False. It is true that $100n^2 + 10n + 15 = O(n^3)$, but it is not true that $100n^2 + 10n + 15 = \Omega(n^3)$. This is because no matter how small a constant we multiply n^3 with, it will eventually become larger than $100n^2 + 10n + 15$.

   (ii) I prefer an algorithm running in $\Theta(\sqrt{2^n})$ time relative to an algorithm running in $\Theta(2^{\log_2 n})$ because the first algorithm is more efficient.
   Solution: False. $\sqrt{2^n}$ can be rewritten as $2^{n/2} = (2^{1/2})^n \approx (1.44)^n$. This is an exponential function, whereas $\Theta(2^{\log_2 n})$ can be simplified to $\Theta(n)$.

(iii) Every algorithm known to us, for finding the median of a list of n numbers takes time $\Omega(n^2)$.
Solution: False. We can sort a list with n numbers in $\Theta(n \log n)$ time and then in an additional $O(1)$ time, find the median.

(iv) $\log_2 n = \Theta(\log_3 n)$.
Solution: True. By the change of base formula, $\log_2 n = \log_3 n / \log_3 2$. Since $\log_3 2$ is just a constant, we get $\log_2 n = \Theta(\log_3 n)$.

(v) $n^{\log_2 n} = \Theta(2^{(\log_2 n)^2})$.
Solution: True. $2^{(\log_2 n)^2}$ can be rewritten as $(2^{\log_2 n})^{\log_2 n} = n^{\log_2 n}$.

2. Write down the worst case running time of each of the following code fragments as a function of n. Use the Θ notation to express your answers. Show your work to receive partial credit.

(a) The arguments to this function are an element k and a list L of length n.

```
function scan(k, list L)
    i ← 1
    while k ≠ L[i] do
        i ← i + 1
    return i
```

Solution: $\Theta(n)$. In the worst case, the entire list will be scanned.

(b) The arguments to this function are two $n \times n$ matrices A and B. The function computes the matrix product $A \cdot B$ and returns the resulting $n \times n$ matrix C.

```
function matrixMult(A, B)
    for i ← 1 to n do
        for j ← 1 to n do
            C[i, j] ← A[i, 1] · B[1, j] + A[i, 2] · B[2, j] + ··· + A[i, n] · B[n, j]
    return C
```

Solution: $\Theta(n^3)$. The statement that assigns a value to $C[i, j]$ takes $\Theta(n)$ time to execute. This statement is within nested loops that causes this statement to execute n^2 times.

(c) The arguments to this function are an element k and a list L of length n.

```
function fastScan(k, list L)
    left ← 1; right ← n
    while left ≤ right do
        mid ← (left + right)/2
        if L[mid] = k then
            return mid
        else if L[mid] < k then
            right ← mid − 1
        else if L[mid] > k then
            left ← mid + 1

    return 0
```

Solution: $\Theta(\log n)$. This is essentially a "binary search" type algorithm with the indices $left$ and $right$ start off at distance n from each other and come closer by a factor of 2 in each iteration (which take $O(1)$ time).

(d)
```
for i ← 1 to n do
    for j ← n downto i do
        print("hello")
```

Solution: $\Theta(n^2)$. For every execution of the outer loop, the inner loop executes at most n times. Thus the running time is $O(n^2)$. Now consider values of i from 1 to $n/2$. For these values of i, the inner loop executes at least $n/2$ for every execution of the outer loop. Thus the number of executions of the print-statement is at least $n^2/4 = \Omega(n^2)$. Thus the running time is also $\Omega(n^2)$. Since the running time is $O(n^2)$ and $\Omega(n^2)$, it is $\Theta(n^2)$.

3. Given a list L of length n ($n \geq 3$), an element $L[i]$, $1 < i < n$, is a *local minimum* if $L[i-1] \geq L[i]$ and $L[i] \leq L[i+1]$. For example, if $L = [3, 1, 7, 7, 7, 2, 11]$ then the elements 1, 7 (the middle one), and 2 are all local minima.

   (a) A length-n list L is called *good* if $L[1] \geq L[2]$ and $L[n-1] \leq L[n]$. Argue (in 2-3 sentences) that any good list has a local minimum.
   Solution: If $n = 3$, then $L[2]$ is a local minima since L is good. Proceeding inductively, consider a length-n list L with $n > 3$. If $L[2]$ is a local minimum, we are done. Otherwise, $L[2] > L[3]$ and therefore the sublist $L[2, \ldots, n]$ is good and applying the inductive hypothesis to this smaller list, we see that it has a local minimum.

   (b) Describe an algorithm (using clear pseudocode) that takes as input a good list L and returns the index of a local minimum in L. If L has several local minima, your algorithm can return the index of any of these. It is required that your algorithm run in time that is asymptotically *faster* than $\Theta(n)$.
   Hint: Start by looking at the middle element of L and its neighbors on either side. Depending on how the middle element of L compares to its neighbors, your algorithm can determine if (i) the middle element is a local minimum or (ii) which of the two halves of L is guaranteed to contain a local minimum.
   Solution: Pseudocode is given below. Since the input L is good, we know that it will contain a local minimum and so the only way of exiting from the function is via the **return** $L[mid]$ statement.

```
function localMinimum(list L)
    left ← 1; right ← n
    while True do
        mid ← (left + right)/2
        if (L[mid] ≤ L[mid − 1]) and (L[mid] ≤ L[mid + 1]) then
            return L[mid]
        else if L[mid] > L[mid − 1] then
            right ← mid
        else if L[mid] > L[mid + 1] then
            left ← mid
```

   (c) Express the worst case running time of your algorithm as a function of n (in Θ notation). Here n is the size of the input list L.
   Solution: The worst case running time if $\Theta(\log n)$ since the algorithm is essentially binary search.

4. We want to determine if a given list L with n elements has a *majority* element and if so return it. (Recall that a *majority* element in a length-n list L is one that occurs *more* than $n/2$ times.)

   Here is a simple randomized algorithm for this problem that runs in $\Theta(n)$ time.

```
function majority(L)
    n ← length(L)
    Comment: pick a random index i between 1 and n
    i ← random(1, n)

    Comment: Count the number of time L[i] occurs in the list
    count ← 0 Comment: Fixed a typo here; see more on this below.
    for j ← 1 to n do
        if L[i] = L[j] then
            count ← count + 1

    Comment: Check if count is more than n/2
    if count > n/2 then
        return L[i]
    else
        return "no majority"
```

   (a) This algorithm does not always produce the correct answer. Explain the circumstances under which it produces an incorrect answer (1-2 sentences). Specifically, discuss whether the algorithm can produce an incorrect answer for each of the two cases: (i) L has a majority element and (ii) L does not have a majority element.

   Solution: If L does have a majority, but contains some elements distinct from the majority, then there is a chance that the random index i picked by the algorithm points to a non-majority element. In this case, the algorithm would output "no majority" despite their being one. If L does not have a majority, then no matter what random value is assigned to i, the algorithm will correctly output "no majority."

   (b) What is the maximum probability that the algorithm produces an incorrect answer?

   Solution: Suppose that L has a majority and this occurs t times in the list. (Note: $t > n/2$.) The probability that the algorithm will pick a non-majority element is $n - t/n$ and since $t > n/2$, we see that

$$\frac{n-t}{n} < \frac{n-n/2}{n} = \frac{1}{2}.$$

   (c) Let us suppose that we want to reduces the probability of this algorithm producing an incorrect output to at most $1/10$. What changes would you make to the algorithm? You can describe your changes in words, no need for pseudocode.

   Solution: We call the `majority` function 4 times and if the `majority` function returns an element in any one of the calls, we return that element (Note: if more than one call returns an element, then the elements returned by different calls will be identical). Otherwise, if "no majority" is returned by every call, then that is what is returned.

   As we have seen, the probability that L contains a majority, but the call to `majority` outputs "no majority" is less than $1/2$. The probability that this happens for all 4 independent attempts is $1/(2^4) = 1/16 < 1/10$.

Comments: Most students understood the source of possible incorrect output (part (a)). Most students also understood that the error probability (in part (b)) is bounded above by 1/2. However, a fairly large fraction of students presented complicated (and incorrect) solutions to (c) forgetting that one can drive the error probability down by simply repeating the randomized algorithm a few times. In general, this is the first idea one should try when faced with the problem of reducing error probability or a randomized algorithm. As mentioned in class, this technique is called *probability amplification*.

A small number of students assumed that my typo (see pseudocode above) was a deliberate attempt at trickery. You may remember that in the exam, *count* was (incorrectly) initialized to 1. I was quite lenient in my grading when I read solutions that took this at face value. It is worth noting that the question does make sense even with this typo. Now, even when L has no majority, the algorithm could behave erroneously. As a particularly bad example, consider a situation in which L has an even number of elements, half of which are 1 and the other half are 2. L has no majority, but no matter whan random choice is made, the algorithm outputs an element (1 or 2). Thus in these circumstances, the error probability is 1.

5. Here are two problems on the *stable marriage* problem.

   (a) Show the execution of the Gale-Shapley algorithm on an input with 3 men and 3 women having the following preferences:

   $$m_1 : w_2 > w_1 > w_3 \qquad m_2 : w_2 > w_1 > w_3 \qquad m_3 : w_1 > w_2 > w_3$$

   $$w_1 : m_2 > m_1 > m_3 \qquad w_2 : m_3 > m_1 > m_2 \qquad w_3 : m_3 > m_2 > m_1$$

   Let us suppose that whenever your algorithm has choice of "free" men to pick for making a proposal, it picks the man with lowest index.

   Solution: Here is the execution of the algorithm shown as a sequence of proposals by men and responses by women.

   (a) m_1 proposes to w_2; w_2 is free and accepts
   (b) m_2 proposes to w_2; w_2 prefers current partner m_1 and rejects
   (c) m_2 proposes to w_1; w_1 is free and accepts
   (d) m_3 proposes to w_1; w_1 prefers current partner m_2 and rejects
   (e) m_3 proposes to w_2; w_2 accepts m_3 and releases m_1
   (f) m_1 proposes to w_1; w_1 prefers current partner m_2 and rejects
   (g) m_1 proposes to w_3; w_3 is free and accepts

   The resulting marriage is $\{(m_1, w_3), (m_2, w_1), (m_3, w_2)\}$.

   Comment: Most students got this problem completely correct. A few students lost a small number of points for ignoring my instruction on the order in which "free" men are processed by the algorithm.

   (b) Consider the *stable roommate* problem in which we are given n individuals (for even n) and each individual has preferences over the remaining $n - 1$ individuals. (So in this problem, the individuals are not divided into two groups, as in the *stable marriage* problem.) Unlike the stable marriage problem, there are instances of the stable roommate problem for which no solution exists. Consider the following example with 4 individuals A, B, C, and D with the following preferences:

   $$A : B > C > D, \qquad B : C > A > D, \qquad C : A > B > D, \qquad D : A > B > C.$$

   Prove that for this input, there is no solution.

   Solution: There are three matchings to consider: $\{(A, B), (C, D)\}$, $\{(A, C), (B, D)\}$, and $\{(A, D), (B, C)\}$. For each of these matchings, there is an unstable pair. Below, I identify the unstable pair for each matching and provide an explanation for the instability.

$\{(A,B),(C,D)\}$**:** (B,C) is unstable because B prefers C to A and C prefers B to D.
$\{(A,C),(B,D)\}$**:** (A,B) is unstable because A prefers B to C and B prefers A to D.
$\{(A,D),(B,C)\}$**:** (A,C) is unstable because C prefers A to B and A prefers C to D.

Since every possible matching has an instability, there is no solution to this instance of the stable roommate problem.

Comment: A lot of students lost points for this problem because they were trying to show that the Gale-Shapley algorithm would not work for this instance. The question was not asking about the Gale-Shapley algorithm or whether *any* algorithm would be able to find a stable matching for this input. The question was more basic: is there even a stable matching for this input?