CS 161 Homework 1 Solutions

Yusu Qian

06/27/2019

Exercise 1 *Set-up (2 pts)*

(a) Please do the following, if you haven't already, then answer "I did the things" for part (a).

- Read the syllabus (at least up to the "Resources" section).
- Join the course Piazza and Gradescope (hint: links are in the syllabus).

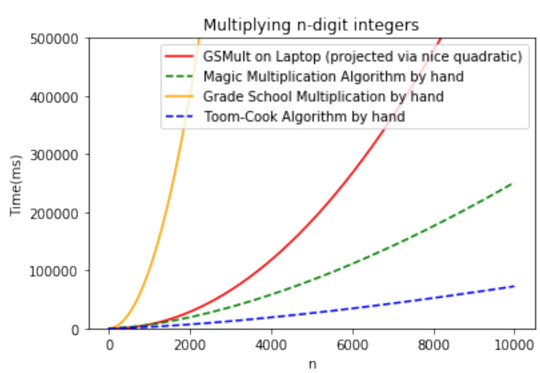
(b) Set up Jupyter notebooks and make sure you can get `lecture1_karatsuba.ipynb` up and running:

- Go to jupyter.org and choose either "Try it in your browser" or "Install the Notebook". Follow the instructions.
- Get the files `multHelpers.py` and `lecture1_karatsuba.ipynb` from Canvas ("Files" tab, in Lecture Materials>Lecture 01) and run `lecture1_karatsuba.ipynb` in the Jupyter Notebook.
  - If you are using the Notebook in your browser, you can upload the files by using File>Open... and then clicking Upload.
  - If you are installing the Notebook, you should install Python 3.3 or higher, and you may need to install matplotlib separately if you do not go through Anaconda to install Python.
- In lecture, we briefly mentioned that the Toom-Cook multiplication algorithm runs in time $O(n^{1.465})$. Find the cell where we compare the running time of grade-school multiplication to "Magic Multiplication" (Karatsuba). Add the following line, then re-run the cell to generate the new plot comparing Toom-Cook to the multiplication algorithms from class, and include a screenshot of it as your answer for part (b)!

```
plt.plot(nValsTmp, [ n**(1.465)/10 + 100 for n in nValsTmp], "--", color="blue",
                                          label="Toom-Cook Algorithm by hand")
```

Solution 1

(a) I did the things.



(b)

Exercise 2 *Basic Big-O (4 pts)*

Using the definitions of $O(\cdot), \Omega(\cdot)$, and $\Theta(\cdot)$, formally prove the following statements:

(a) $5\sqrt{n} + 3 = O(\sqrt{n})$.

(b) $n^{100} = \Omega(n)$.

(c) $2^{100} = \Theta(1)$.

(d) 4^n is **not** $O(2^n)$.

Solution 2

(a) By definition, we need to prove that there are positive constants n_0 and c such that at and to the right of n_0, the value of $5\sqrt{n} + 3$ always lies on or below $O(\sqrt{n})$.
When $n > 9$, $\sqrt{n} > 3$, $5\sqrt{n} + 3 < 5\sqrt{n} + \sqrt{n} = 6\sqrt{n}$
Thus we have $n_0 = 9$, $c = 6$, when $n > n_o$, $5\sqrt{n} + 3 < c\sqrt{n}$.

(b) By definition, $n^{100} = \Omega(n)$ holds as there are positive constants $n_0 = 1$ and $c = 1$ such that at and to the right of n_0, $n^{100} > 1 = c * n$. Thus $n^{100} = \Omega(n)$

(c) 2^{100} lies between $(2^{100}+1)*1$ and $(2^{100}-1)*1$, thus, there exists positive constraints $c_1 = 2^{100} - 1$ and $c_2 = 2^{100} + 1$, such that the value of 2^{100} lies between $c_1 * 1$ and $c_2 * 1$. Thus, $2^{100} = \Theta(1)$.

(d) By definition, 4^n is $O(2^n)$ if there are positive constants n_0 and c such that at and to the right of n_0, the value of 4^n always lies on or below $c * (2^n)$.
However, whatever value c is of, let n_0 be $log(c) + 1$.
$\frac{4^n}{c*2^n} = \frac{2^n}{c}$. When $n > n_0$, $\frac{2^n}{c} > 2$, thus $4^n > c * 2^n$.
Thus, 4^n is not $O(2^n)$.

Exercise 3 *More Big-O: True or False? (8 pts)*

In the following, suppose that $f(n)$ and $g(n)$ are strictly positive, strictly increasing functions. Formally prove or disprove the following statements:

(a) If $f(n) = O(g(n))$ then $100f^2(n) = O(g^2(n))$.

(b) There exists a constant $\alpha > 0$ such that $\log n = \Omega(n^\alpha)$. (You may assume, if it helps, that $\log n = O(n)$ but n is not $O(\log n)$.)

(c) If $f(n) = O(g(n))$ then $\log(f(n)) = O(\log(g(n)))$. (You may assume $\log(f(n)), \log(g(n)) > 0$.)

(d) If $f(n) = O(g(n))$ then $2^{f(n)} = O(2^{g(n)})$.

Solution 3

(a) True. By definition, if $f(n) = O(g(n))$, there exists a positive constant c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $O(g(n))$. $100f^2(n) = (10f(n))^2 < (10c * g(n))^2 = 100c^2 * g^2(n)$ when $n >= n_0$. To rephrase, there exists a positive constant $c_1 = 100c^2$ such that at and to the right of n_0, the value of $f(n)$ always lies on or below $O(g^2(n))$.

(b) True. For $\log n = \Omega(n^\alpha)$ to hold, there should be a constant c and a constant n_0 such that at and to the right of n_0, the value of $\log n$ always lies on or above $c * n^\alpha$. When α is smaller than 1 while larger than 0, n^α is strictly decreasing. $\log n$, on the other hand, is strictly increasing. Let $c = 1, alpha = 0.01$ and $n = 4$. $\log 4 = 2 = 4^{0.5} > 4^{0.01}$. For all $n > 4$, $\log n > n^{0.01}$. Thus, there exists a constant $\alpha > 0$ such that $\log n = \Omega(n^\alpha)$.

(c) True. As $f(n) = O(g(n))$, we can rewrite $f(n)$ as $c * g(n)$ + a lower order term. Thus, $\log(f(n)) =$

$\log(c * g(n) + \log \text{ (a lower order term)}) < \log(c * g(n) + g(n)) = \log((c+1) * g(n)) = \log(g(n)) + \log(c+1) = O(\log(g(n)))$. Thus $\log(f(n)) = O(\log(g(n)))$.

(d) False. By definition, if $f(n) = O(g(n))$, there exists a positive constant c such that at and to the right of n_0, the value of $f(n)$ always lies on or below $c * g(n)$. If $2^{f(n)} = O(2^{g(n)})$ then $2^{f(n)} < c_1 * 2^{g(n)} = 2^{logc_1} * 2^{g(n)} = 2^{g(n)+logc_1}$. Compare these two, we see that it only holds when $c < 1$.

Exercise 4 *Recurrence Relations (6 pts)*

Using either the Master Theorem or the "tree" method (show your work), give the best big-O bound possible on the following recurrences:

(a) $T(n) = 5T(\frac{n}{3}) + n$ with $T(n) = 1$ for $n < 3$.

(b) $T(n) = 2T(\frac{n}{2}) + n^2$ with $T(n) = 1$ for $n < 2$.

(c) $T(n) = T(n-2) + n$ with $T(n) = 1$ for $n < 2$.

(d) $T(n) = 4T(\sqrt[4]{n}) + \log n$ with $T(n) = 1$ for $n < 2$.

Solution 4

(a) $a = 5$, $b = 3$, $d = 1$, thus $a > b^d$, thus $O(n^{\log_3(5)})$

(b) $a = 2$, $b = 2$, $d = 2$, thus $a < b^d$, thus $O(n^2)$

(c) When $n = 2k+1$, $T(n) = 1 + 3 + ... + n = \frac{(1+n)n}{4}$, $T(n) = T(\frac{n}{2}) + \frac{3n^2}{16} + \frac{8}{n}$, when $n > 2$, $\frac{n^2}{16} > \frac{n}{8}$, so $T(n) < T(\frac{n}{2}) + \frac{n^2}{4}$. When $n = 2k$, $T(n) = 1 + 2 + ... + n = \frac{(2+n)n}{4} + 1$, $T(n) = T(\frac{n}{2}) + \frac{3}{16}n^2 + \frac{n}{4}$, when $n > 4$, $\frac{n^2}{16} > \frac{n}{4}$, so $T(n) < T(\frac{n}{2}) + \frac{n^2}{4}$. Thus, for every $n > 4$, $T(n) < T(\frac{n}{2}) + \frac{n^2}{4}$. $a = 1, b = 2, d = 2$, $a < b^d$, thus $O(n^2)$

(d) We can change variables. $T(n) = 4T(\sqrt[4]{n}) + \log n$ can be changed into $T(2^m) = 4T(2^{\frac{m}{4}}) + m$. Now rename $S(m) = T(2^m)$ to get the new recurrence $S(m) = 4S(\frac{m}{4}) + m$. This recurrence has a solution of $S(m) = O(m * logm)$. Thus, $T(n) = T(2^m) = S(m) = O(m * logm) = O(lognloglogn)$.

Exercise 5 *Generalized Karatsuba (6 pts)*

In class, we saw that Karatsuba's Algorithm breaks up the problem of integer multiplication of two n-digit numbers into 3 sub-problems of size $\frac{n}{2}$. We also mentioned that the Toom-Cook Algorithm breaks up integer multiplication into 5 sub-problems of size $\frac{n}{3}$.

Suppose that, for any integer $k \geq 2$, we can break up one integer multiplication xy into $2k-1$ sub-problems of size $\frac{n}{k}$ in time $O(n)$. Furthermore, suppose that by simple addition and subtraction of the correct sub-problems, we can calculate $\frac{2n}{k}$-digit numbers $q_0, \ldots, q_{2k-2}$ such that $xy = q_{2k-2}10^{(2k-2)n/k} + \cdots + q_2 10^{2n/k} + q_1 10^{n/k} + q_0 10^0$.

(a) Argue that, for any constant k, the number of one-digit operations required to reassemble the sub-problems into the overall product xy is $O(n)$.

(b) Give a recurrence relation representing the run-time of this generalized-Karatsuba algorithm for arbitrary k, and solve the recurrence relation to give the tightest-possible big-O bound.

(c) Your friend, Ms. Take, is very excited about this problem and tells you she's come up with an $O(n \log n)$ algorithm for integer multiplication—the fastest ever discovered! Here is her reasoning:

   i. We know that for any k, we can break up an n-digit multiplication into $2k-1$ sub-problems of size $\frac{n}{k}$.

ii. You showed in part (a) that recombining these sub-problems takes $O(n)$ time.

iii. Therefore, just let $k = \sqrt{n}$. This gives us the recurrence relation $T(n) = (2\sqrt{n} - 1)T(\sqrt{n}) + O(n)$.

iv. This recurrence solves to $O(n \log n)$. First, notice that there are $\log \log n$ levels of the recursion tree, because that's how many times you have to take the square root of n to get down to $O(1)$. At the 0th level, there is 1 sub-problem of size n. At the 1st level, there are fewer than $2\sqrt{n}$ sub-problems of size $\sqrt{n}$. At the 2nd level, there are fewer than $2\sqrt{n} \times 2n^{1/4} = 4n^{3/4}$ sub-problems of size $n^{1/4}$. In general, at the t'th level, there are fewer than $2^t n^{1-\frac{1}{2^t}}$ sub-problems of size $n^{1/2^t}$, for a total of $2^t O(n)$ work at level t. Finally, observe that $\sum_{t=0}^{\log \log n} 2^t O(n) = O(2^{\log \log n} n) = O(n \log n)$, as claimed!

What big conceptual error has Ms. Take made?

Solution 5

(a) There are $2k - 1$ subproblems, each with $\frac{2n}{k}$ digits, $(2k - 1) * \frac{2n}{k} < 4n$, time needed to reassemble is $O(n)$.

(b) We divide n times, and eventually have $(2k - 1)^{\log_k n} = n^{\log_k (2k-1)}$ subproblems. $T(n) = (2k - 1)T(\frac{n}{k}) + O(n)$. $T(n) = O(n^{\log_k (2k-1)})$.

(c) n is a fixed number, and k should be too. However, it seems Ms. Take changes the value of k in every step and uses square root of the new digit length. Thus the calculation of levels is incorrect.

Exercise 6 *Preview of Selection (6 pts) answer on the next page*

In lecture 03 we will discuss the selection problem: given an unsorted list of n elements and an integer k, return the k'th smallest element—i.e., when $k = 1$ return the min, when $k = n$ return the max, etc.

In this problem, we are instead given two **sorted** lists, each of length n; assume that all the elements are distinct. Give a divide-and-conquer algorithm that returns the k'th smallest element in the union of the two lists, in time $O(\log n)$. Provide an informal argument that your algorithm has the correct runtime, and a concise proof by induction that your algorithm returns the correct answer.

Solution 6

n = len(input)
sequence$_1$ =list$_1$[$first\frac{k}{2}elements$]
sequence$_2$ =list$_2$[$first\frac{k}{2}elements$]
for $k >= 1$
    if $sequence_1[\frac{k}{2}] < sequence_2[\frac{k}{2}]$
        $max = sequence_1[\frac{k}{2} - th]$
        list$_1$ $delete \quad first \quad \frac{k}{2} \quad elements$
        $k = \frac{k}{2}$
    else if $sequence_1[\frac{k}{2}] > sequence_2[\frac{k}{2}]$
        $max = sequence_2[\frac{k}{2} - th]$
        list$_2$ $delete \quad first \quad \frac{k}{2} \quad elements$
        $k = \frac{k}{2}$
return max

Explanation:
I divide this problem to first finding the $\frac{k}{2}$ smallest elements in the union of the two lists, and then finding the $\frac{k}{4}$ smallest elements, till the last step when $k = 1$ and we find the only one we need. This element is the k-th smallest element in the union of the two lists.
Let's take a closer look at the first step. Take out the smallest $\frac{k}{2}$ elements in both lists, and compare the $\frac{k}{2} - th$ elements. If the $\frac{k}{2} - th$ element in $list_1$ is smaller than than in $list_2$, it means that the first $\frac{k}{2}$ elements in $list_1$ are among the first k smallest elements in the union of the two lists. The $\frac{k}{2} - th$ element in $list_1$ is so far the largest element we have collected, and we give its value to a variable we call max. Repeat this step and reduce k by 2 all the way down to when $k = 1$. The last element we collect and assign to max is the k-th smallest element we are finding.
Running time: There are $log_2 k$ steps. In each step, we compare the largest elements in the two sub-lists, assign a value to the variable max, and use $\frac{k}{2}$ to replace the original k. Thus, $T(k) = T(\frac{k}{2}) + \Theta(1)$ when $k > 1$ and $T(k) = \Theta(1)$ when $k = 1$. The total cost therefore is $c * \log k + ck$. As $k < 2n$, $(c * \log k + ck) < (c * \log 2n + 2cn) = c * \log n + 2cn + c * \log 2 = O(\log n)$