Basic Algorithms, Assignment 2, Solutions
Due, Tuesday, Sept 18

Prof. Spencer’s Office Hours: Wednesday 1-2:30
Course website:
www.cs.nyu.edu/cs/faculty/spencer/basicalg/index.html

1. If a computer does a billion operations per second how many does it do in a year. Round this to one significant digit and write it in exponential form, e.g., something like $3 \cdot 10^{43}$ (but thats not the answer!)

Solution: We have $60 \cdot 60 = 3600$ seconds in an hour, $3600 \cdot 24 \sim 80000$ seconds in a day and $\sim 80000 \cdot 365 \sim 3 \cdot 10^7$ seconds in a year so the computer will do $\sim 10^9 \cdot 3 \cdot 10^7 = 3 \cdot 10^{16}$ operations in a year.

2. Redo Table 2.1 on Page 34 with the processor performing a billion operations per second and with functions $n, n \log_2 n, n^3, 1.1^n$.

Solution: I give the time for input sizes resulting in a time greater than 1 second.

- $n$: < 1 sec, for each $n$
- $n \log_2 n$: < 1 sec for each $n$
- $n^3$: $n = 1,000$- 1 sec, $n = 10,000$-16min, $n = 100,000$-11 days, $n = 1,000,000$-32 years
- $1.1^n$: $n < 100$ - < 1sec, $n \geq 1000$ very long

3. Suppose an algorithm takes $n \log_2 n$ operations, and our computer does a billion operations per second. For how large an $n$ can our computer do the algorithm in a day. (I’m looking for a rough answer here, say to one significant digit.)

Solution: In a day the computer does $\sim 10^9 \cdot 80000 = 8 \cdot 10^{12}$ operations. Now the function $n \log_2 n$ does not have a nice exact inverse but the $\log_2 n$ term is nearly constant for long periods. As $\log_2 10^3 \sim 10$ (as $2^{10} = 1000$ except for roundoff error!) $\log_2 10^{12} \sim 40$. We want $n \log_2 n \sim 8 \cdot 10^{12}$ but $\log_2 n$ is around 40 so we want $n \sim 8 \cdot 10^{12}/40 = 2 \cdot 10^{11}$. Well, was our approximation OK? Actually $\log_2 n$ for this $n$ is less than 40 but only by 2 or 3 so the approximation is pretty good.

4. How many binary digits are there in a 100 decimal digit number? (Actually, there are a few possible answers depending on the exact number but we’re just looking for a rough answer.)

Solution: Each 3 decimal digits give 10 binary digits so we get $10 \cdot 33 = 330$ binary digits from $10^{99}$ plus another three or four (or five or six)
from the remaining factor (between 10 and 99) so 335 would be a pretty good estimate.

5. Let \( f(n) = \sqrt{2n \ln n} \) and let \( g(n) = 6\sqrt{n} \). Which function is larger as for \( n \) sufficiently large? When does the eventually larger function become larger? (These functions actually came up in my work some years ago.)

Solution: In comparing the \( \sqrt{n} \) factors cancel so we are comparing \( \sqrt{2 \ln n} \) with 6. As the first goes to infinity \( f(n) \) is eventually larger. This occurs when \( \sqrt{2 \ln n} = 6 \), when \( 2 \ln n = 36 \), when \( \ln n = 18 \), when \( n = e^{18} \). Another handy estimate is that \( e^3 \sim 20 \) (they are very close!) so that \( e^{18} \sim (20)^6 = 64000000 \).

6. John Wastenot decides to implement the Stable Marriage Gale-Shapley algorithm without using much space. His input consists of linked lists of size \( n \), \( \text{MANLIST}[i] \), \( 1 \leq i \leq n \), and \( \text{WOMANLIST}[j] \), \( 1 \leq j \leq n \), giving the preference list for each person of the opposite gender. He adds two array \( \text{MANMATE}[i] \), \( 1 \leq i \leq n \), and \( \text{WOMANMATE}[j] \), \( 1 \leq j \leq n \), giving the current mate \( \text{NIL} \) if there isn’t one, which is the original value) of each person. He adds an array \( \text{MANLASTPROP}[i] \) giving the last woman (\( \text{NIL} \) if there were none) that man \( i \) has proposed to.

Write a pseudocode program for implementing Gale-Shapley with this data structure. Do a worst case analysis of how many steps the algorithm will take. (Note: The answer will not be \( \Theta(n^2) \).)

Solution (Outline): In the WHILE loop you go through \( i \) until finding \( \text{MANMATE}[i] = \text{NIL} \). If none is found, exit and print out couples. (This takes \( \leq n \) steps.) Now go through \( \text{MANLIST}[i] \) until finding \( \text{MANLASTPROP}[i] \), let \( j \) be the next woman on \( \text{MANLIST}[i] \). (This takes \( \leq n \) steps.) (If \( \text{MANLASTPROP}[i] = \text{NIL} \) let \( j \) be the first woman on \( \text{MANLIST}[i] \).) Reset \( \text{MANLASTPROP}[i] = j \). Now \( i \) proposes to \( j \). The \( \text{WOMANMATE}[j] \) case is now easy and quick. But when \( \text{WOMANMATE}[j] = i' \neq \text{NIL} \) and we must see whether \( j \) prefers \( i \) or \( i' \). To do that go through \( \text{WOMANLIST}[j] \) until reaching either \( i \) or \( i' \), this taking \( \leq n \) steps. The updating in the two cases is quick. The total time for the WHILE loop is now \( \leq n + n = O(n) \). But the WHILE loop may be used \( n^2 \) time so this implementation has time \( O(n^3) \). Note that this can be considerably slower than the implementation with time \( O(n^2) \). This is a quite common occurance, that saving space by keeping auxilliary arrays to a minimum can lead to a great increase in time.

Throughout human history, mankind has been a lot better at
gathering data than at thinking about it.
From *Mirror Worlds* by David Gelernter