Data Structures Set 2

Sherri Shulman
Data Structures Set 2

Oct 10, 2013

2.1

$2/N, 37, \sqrt{N}, N, N \log\log N, N \log N, N \log N^2, N \log^2 N, N^{1.5}, N^2, N^2 \log N, N^3, 2^{N/2}, 2^N$.

$N \log N$ and $N \log(N^2)$ grow at the same rate since $N \log N^2) = 2 \log N$ and we "drop" the 2.

2.2 **a** This is true. Since adding the two functions will double the time, but that's a constant factor.

b This is false. If $T_1(N) = 2N, T_2(N) = N, and f(N) = N$ little "o" is strictly less than, and N is not stritly less than N. (see p 44)

c This is false. A counterexample is $T_1(N) = N^2, T_2(N) = N, f(N) = N^2$. So this does not have constant growth.

d False: the same counterexample from part (c).

2.3 $N \log N$ is slower growing. Suppose otherwise. Then $N^\epsilon/\sqrt{\log N}$ would grow slower than $\log N$. Take logs of both sides, and (under this assumption) $\epsilon/\sqrt{\log N} \log N$ grows slower than $\log\log N$. But the first expression simplifies to $\epsilon\sqrt{N}$ (since $\log N/\sqrt{\log N} = \sqrt{\log N}$). If we let $L = \log N$, this means that $\epsilon\sqrt{L} < \log L$. But $\log N = O(sqrt(N))$! A contradiction. (Note that $\log(x) < x$ for $x > 0$. So $\log(x) = \log(\sqrt{x})^2 = 2 \log \sqrt{x} < 2\sqrt{x}$).

2.7 **1** This is $O(N)$ since we run through the loop once.

2 This is $O(N^2)$ since we run throughh the inner loop $N * N$ times.

3 This inner loop is done $N * N$ for each N so is $O(N^3)$

4 This is $O(N^2)$ since we go through the inner loop $1, 2, 3, 4, \cdots, n$ times and this sum is $n(n + 1)/2$ which is $ON^2)$.

5 This does the outer loop n times. It does the next loop n^2 times. And then the inner loop n^2 times. So that gives us $O(n^5)$.

6 This is $n * n^2$, (outer loop times inner loop) so the if statement is executed N^3 times. But we only do the innermost loop when $j\%i == 0$ so we only execute the innermost loop i times for each i. So that gives us $1 + 2 + 3 + \cdots n = O(N^2)$.

2.14 This is a very cool algorithm. If the polynomial is kept as an array of coefficients, we minimize the number of multiplications by multiplying the highest order coefficient first (innermost) and then successively multiplying by x to build up the right degree. Eg for the example polynomial:

$$\begin{aligned}
2 + (x * (1 + (x * (0 + x(8))))) &= \\
2 + (x * (1 + (x * (8x)))) &= \\
2 + (x * (1 + (8x^2))) &= \\
2 + (x + (8x^3))) &= \\
2 + x + 8x^3 &=
\end{aligned}$$

Actually in thgis implementation:

$$\begin{aligned}
[2, 1, 0, 8] &= \\
p = x * 0 + 8 &= \\
p = x(x * 0 + 8) + 0 &= \\
p = x * (x(x * 0 + 8) + 0) + 1 &= \\
p = x(x(x(x * 0 + 8) + 0) + 1) + 2 &= \\
p = x(x(x(8)) + 1) + 2 &= \\
p = 8x^3 + x + 2 &
\end{aligned}$$

The running time (the number of multiplications) is the degree of the polynomial.

2.15 Assuming that we don't have to account for reading in the array, we can just do a binary search. I.e. if we want to check if $A_i = i$, start with the $i'th$ element of the array. If the value is lower, go half way down, higher go halfway up. This leads to a binary tree, which has a height of $\log N$. So $O(\log N)$

2.26 **a** The recursion is unnecessary if there ar etwo or fewer elements, so the size of the array terminates recursion.

b One way to do the odd array is to note that if the first $n - 1$ elements have a majority, then the last element cannot change this. Otherwise, the last element could be a majority. So if N is odd, ignore the last element. Run the algorithm as before. If no majority element emerges, return the nth element as a candidate.

c Running time is $O(N)$ and satisfies $(N) = T(N/2) + O(N)$.

d One copy of the original needs to be saved. After this the B array can be used. The original strategy implies that you use $O(\log N)$ arrays. But this modification only uses two copies.

I haven't written the code yet.

2.28 **a** Find the two largest numbers (one pass)

b The max difference is at least 0, so that can be the initial answer to beat. At any point, we have the current value j and the current low point i. If $a[j] - a[i]$ is larger than the current best, update the best. If $a[j] < a[i]$ reset the current low point to i. Start with i at 0 and j at 0. One pass, so $O(N)$.

c This is just like [a].

d This is just like [b].

2.31 No ... if $low = 1, high = 2$ then $mid = 1$ and the recursive all does not make any progress.