

CS502 Algorithms, Spring 2004
Solutions to Homework 2

Solution 1:

- (a) Let $f(n) = 2n^3 - 3n^2 + 10$. To show that $f(n) \in \Theta(n^3)$ we need to establish the lower and upper bounds.

Lower bound: We show that there exist positive constants c_1 and n_0 such that

$f(n) \geq c_1 n^3$, for all $n \geq n_0$. For this we will assume that $n_0 \geq 3$, implying that $n^3 - 3n^2 \geq 0$, and we will let $c_1 = 1$. For all $n \geq n_0$, we have

$$f(n) = 2n^3 - 3n^2 + 10 \geq n^3 + (n^3 - 3n^2) + 10 \geq n^3 = c_1 n^3.$$

Upper bound: We show that there exist positive constants c_2 and n_0 such that $f(n) \leq c_2 n^3$ for all $n \geq n_0$. For this we will assume that $n_0 \geq \sqrt[3]{10}$, implying that $10 \leq n^3$, and we will let $c_2 = 3$. For all $n \geq n_0$, we have

$$f(n) = 2n^3 - 3n^2 + 10 \leq 2n^3 + 10 \leq 2n^3 + n^3 = 3n^3 = c_2 n^3.$$

Thus, to satisfy both bounds, we select $n_0 = \max(3, \sqrt[3]{10}) = 3$, and set $c_1 = 1$ and $c_2 = 3$, from which it follows that

$$0 \leq c_1 n^3 \leq f(n) \leq c_2 n^3,$$

implying that $f(n) \in \Theta(n^3)$.

- (b) Let $f(n) = n^2 + 10n \lg^2 n \in O(n^2)$ (\lg is \log_2). To show that $f(n) \in O(n^2)$, we need to show that there exist positive constants c and n_0 such that $0 \leq f(n) \leq cn^2$ for all $n \geq n_0$. First off, we select n_0 large enough so that for all $n \geq n_0$, $\lg^2 n \leq n$. By a little trial and error, we find that for $n = 16$, $\lg^2 n \leq n$. (This is easy to verify that they are equal for $n = 16$. You can prove that the inequality holds for larger n , by showing that the derivative of $\lg^2 n$ grows more slowly than the derivative for n for $n \geq 16$.) Now, we let $c = 11$, and let $n_0 = 16$, from which we conclude that for all $n \geq n_0$,

$$f(n) = n^2 + 10n \lg^2 n \leq n^2 + 10n^2 \leq 11n^2 = cn^2,$$

as desired.

Solution 2:

- (a) Let $f(n) = 2n^3 - 3n^2 + 10$. To show that $f(n) \in \Theta(n^3)$ by the Limit Rule, we need to show that

$$\lim_{n \rightarrow \infty} \frac{f(n)}{n^3} = c,$$

for some strictly positive constant c . Plugging in $f(n)$ we have

$$\lim_{n \rightarrow \infty} \frac{2n^3 - 3n^2 + 10}{n^3} = \lim_{n \rightarrow \infty} \left(2 - \frac{3}{n} + \frac{10}{n^2} \right) = 2 - 0 + 0 = 2.$$

Thus by the Limit Rule, $f(n) \in \Theta(n^3)$.

- (b) Let $f(n) = n^2 + 10n \lg^2 n$. To show that $f(n) \in O(n^2)$, by the Limit Rule, we need to show that

$$\lim_{n \rightarrow \infty} \frac{f(n)}{n^2} = c,$$

where c is any nonnegative constant. Plugging in $f(n)$ we have

$$\lim_{n \rightarrow \infty} \frac{n^2 + 10n \lg^2 n}{n^2} = \lim_{n \rightarrow \infty} \left(1 + \frac{10 \lg^2 n}{n} \right) = 1 + 0 = 1.$$

Here we made use the fact that for any positive constants b and d

$$\lim_{n \rightarrow \infty} \frac{\lg^b n}{n^d} = 0.$$

In this case, we applied this with $b = 2$ and $d = 1$. Thus, by the Limit Rule, $f(n) \in O(n^2)$.

Solution 3: The ranking is shown below. Functions f_9 and f_{10} are asymptotically equivalent, as are f_1 and f_7 . (The function f_7 is tricky because it appears to be an exponential function, but after simplifying it is equivalent to n^3 .)

$f_9(n) = 20 \log(n^2)$	$\in \Theta(\lg n)$	(Note 1)
$f_{10}(n) = 4 \log_3 n$	$\in \Theta(\lg n)$	(Note 2)
$f_8(n) = 5 \log^2 n + 10 \log n$	$\in \Theta(\lg^2 n)$	(Note 3)
$f_5(n) = 4\sqrt{n} + 3 \log(n^2)$	$\in \Theta(\sqrt{n})$	(Note 4)
$f_2(n) = 17n + (2/n^2)$	$\in \Theta(n)$	
$f_3(n) = 7n \log \log n$	$\in \Theta(n \lg \lg n)$	
$f_4(n) = 20n \log^3 n + 5n^2$	$\in \Theta(n^2)$	(Note 5)
$f_1(n) = 500n^3$	$\in \Theta(n^3)$	
$f_7(n) = 2^{(3 \log n)}$	$\in \Theta(n^3)$	(Note 6)
$f_6(n) = 2^{\sqrt{n}}$	$\in \Theta(2^{\sqrt{n}})$	

Notes:

- (1) We use the fact that $\lg(n^2) = 2 \lg n \in \Theta(\lg n)$.
- (2) We use the fact that $\log_3 n = (\lg n)/(\lg 3) \in \Theta(\lg n)$.
- (3) To show that $\lg^2 n$ grows asymptotically faster than $\lg n$, we observe that the ratio $\lg^2 n / \lg n = \lg n$ tends to infinity as n tends to infinity.
- (4) To show that \sqrt{n} grows asymptotically faster than either $\lg n$ (the other term) or $\lg^2 n$ (from $f_8()$) we use the fact that $\lg^b n / n^c = 0$ in the limit as n tends to infinity. In this case $b = 1, 2$ and $c = 1/2$.
- (5) We are using the fact that $\lg^3 n$ is asymptotically smaller than n .
- (6) We use the result of the last homework, that $2^{(3 \lg n)} = (2^{\lg n})^3 = (n^{\lg 2})^3 = n^3$.

Solution 4: The recurrence to solve is:

$$T(n) = \begin{cases} 2/7 & \text{if } n = 1, \\ 2T(n/3) + n^2 & \text{otherwise.} \end{cases}$$

We begin by repeatedly expanding the definition, until seeing a trend.

$$\begin{aligned} T(n) &= 2T(n/3) + n^2 \\ &= 2(2T(n/9) + (n/3)^2) + n^2 = 4T(n/9) + 2(n/3)^2 + n^2 \\ &= 4(2T(n/27) + (n/9)^2) + 2(n/3)^2 + n^2 = 8T(n/27) + 4(n/9)^2 + 2(n/3)^2 + n^2 \\ &= \dots \\ &= 2^k T(n/3^k) + \sum_{i=0}^{k-1} 2^i (n/3^i)^2 \end{aligned}$$

First we must get rid of the $T(n/3^k)$, which we do by setting k such that we get to the basis case of $T(1)$. Thus, we want $n/3^k = 1$, from which it follows that $k = \log_3 n$. By definition we have $T(1) = 2/7$. We would also like to simplify the summation. Observe that the expression inside the summation can be written as $n^2(2^i/(3^i)^2) = n^2(2/9)^i$. (We use the fact that $(3^i)^2 = 3^{(2i)} = (3^2)^i$.) Since the n^2 does not change throughout the summation, we can pull it out in front. This gives us

$$T(n) = 2^{\log_3 n} T(1) + n^2 \sum_{i=0}^{\log_3 n - 1} (2/9)^i = n^{\log_3 2} (2/7) + n^2 \sum_{i=0}^{\log_3 n - 1} (2/9)^i$$

Observe that the summation is a geometric series. We may apply the standard formula to get

$$\begin{aligned} \sum_{i=0}^{\log_3 n - 1} (2/9)^i &= \frac{(2/9)^{\log_3 n} - 1}{(2/9) - 1} = -\frac{9}{7}((2/9)^{\log_3 n} - 1) \\ &= -\frac{9}{7}(n^{\log_3(2/9)} - 1) = -\frac{9}{7}(n^{\log_3 2 - \log_3 9} - 1) \\ &= -\frac{9}{7}(n^{(\log_3 2) - 2} - 1) = -\frac{9}{7}\left(\frac{n^{\log_3 2}}{n^2} - 1\right) \end{aligned}$$

Putting this all together we have

$$T(n) = \frac{2}{7}n^{\log_3 2} - n^2 \frac{9}{7} \left(\frac{n^{\log_3 2}}{n^2} - 1 \right) = \frac{2}{7}n^{\log_3 2} - \frac{9}{7}(n^{\log_3 2} - n^2) = \frac{9}{7}n^2 - n^{\log_3 2}.$$

Thus, the final answer is

$$T(n) = \frac{9}{7}n^2 - n^{\log_3 2} \quad (\text{which is } \in \Theta(n^2))$$

Solution 5: There are a couple of ways to solve the 2d maxima problem by divide and conquer. The one that we will show is modeled after MergeSort. The algorithm takes a sub array of P as its input, the call `Maxima(P, p, r)` returns a pair (A, m) , where A is an array containing the maxima of $P[p..r]$, sorted in increasing order of x -coordinate, and $m \geq 1$ is the number of elements in A . The initial call is `Maxima(P, 1, n)`, to compute the maxima of all the points. If

the input array has a single element, then the algorithm returns this element as the single maxima. Otherwise, the algorithm splits the array into two sub arrays of roughly equal size. It then computes the maxima of each sub array, storing the results in arrays $A[1..ma]$ and $B[1..mb]$. Then we invoke the procedure $\text{MaxMerge}(A, B, ma, mb)$ which merges these two arrays of sorted maxima into a single array of sorted maxima. The recursion continues until the arrays have P down to a single element, then this single point is the maxima of the sub array.

```
Maxima(array P, int p, int r)
{
    if (p < r)                // we have at least 2 items
    {
        q = (p + r)/2
        (A, ma) = Maxima(P, p, q) // get maxima of P[p..q]
        (B, mb) = Maxima(P, q+1, r) // get maxima of P[q+1..r]
        return MaxMerge(A, B, ma, mb) // merge the two maxima sets
    }
    else return (P[p], 1) // just one item left
}
```

The procedure for merging two maxima assumes that it has been given two arrays A and B of maxima sorted by increasing x -coordinate. It merges them together into a single array C of maxima, also sorted by x -coordinate, which is returned. By the nature of maxima, since the arrays are sorted by increasing x , they are also sorted by decreasing y . We maintain indices i, j and k , which point to the current elements of arrays A, B , and C , respectively. In each case we consider the current elements A_i and B_j .

Suppose that A_i has the smaller x -coordinate of the two. As with MergeSort, we assume inductively that all elements with smaller x -coordinates from either list have already been processed. If A_i 's y -coordinate is larger than B_j 's y -coordinate, then we know that A_i is not dominated by B_j and hence is not dominated by any subsequent element of B (because they will have even smaller y -coordinates). Hence A_i can be copied to the list of maxima. Otherwise, A_i is dominated by B_j , and it is skipped. The case when B_j has a smaller x -coordinate is symmetric. As with MergeSort, when we have exhausted all the elements of one list, then the elements of the other list are simply copied to the final list of maxima. To simplify things, we will assume that there are no duplicate x -coordinates or y -coordinates.

```
// merges A[1..ma] and B[1..mb]
MaxMerge(array A, array B, int ma, int mb)
{
    array C[1..ma+mb]                // temp storage
    i = j = k = 1                    // initialize indices
    while (i <= ma and j <= mb)      // while both arrays nonempty
    {
        if (A[i].x < B[j].x)        // A[i] is left of B[j]
        {
            if (A[i].y > B[j].y) C[k++] = A[i++] // A[i] is a maxima
            else i++ // A[i] is dominated
        }
        else                        // B[j] is left of A[i]
        {
            if (B[j].y > A[i].y) C[k++] = B[j++]
            else j++
        }
    }
    while (i <= ma) C[k++] = A[i++]
    while (j <= mb) C[k++] = B[j++]
    return C
}
```

```

    {
        if (B[j].y > A[i].y) C[k++] = B[j++] // B[j] is a maxima
        else j++ // B[j] is dominated
    }
}
while (i <= ma) C[k++] = A[i++] // copy leftovers to C
while (j <= mb) C[k++] = B[j++]
return (C[1..k1], k1) // return array of maxima
}

```

Structurally, the algorithm is essentially identical to MergeSort, and hence the $\Theta(n \log n)$ analysis for MergeSort can be applied to this algorithm as well.