H.W. 1

Patrick Chen

June 9, 2017

Contents

1 1-1

Group 1

$f_1(n)$ is the slowest growing function since $log(n)$ grows slower than any power function where the exponent is less than 1. This means that $O(n^{0.99999}logn) < O(n^{0.99999+c}) < O(n) = O(f_2(n))$ as $c \to 0$. Exponential functions grow faster than power functions, so $O(f_4(n)) < O(f_3(n))$. So we have $O(f_1) < O(f_2) < O(f_4) < O(f_3)$.

Group 2

f_1 is $O(1)$ since it is a constant. $f_4(n) = n^{3/2}$. $f_2(n) = (2^{100000})^n$. $f_3(n) = \frac{n!}{2!(n-2)!} = n(n-1)/2! = O(n^2)$. Thus we have $O(f_1) < O(f_4) < O(f_3) < O(f_2)$, since exponentials grow faster than power functions.

Group 3

2 1-2

a

$T(n,n) = \Theta(2n)+T(n/2,y/2) \implies \Theta(2n)+\Theta(n/2)+\Theta(n/8)+.... = \Theta(8/3n) = \Theta(n)$

b

$$T(n,n) = O(n) + T(n,n/2) = log(n)O(n) = O(nlog(n)) \tag{1}$$

c

$$T(n,n) = O(n) + S(n,n/2) = O(n) + O(n) + T(n/2,n/2) \tag{2}$$
$$= O(2n) + O(n) + O(n/2) + ... = O(4n) = O(n) \tag{3}$$

3 Peak-Finding

1-3

a) Yes, the algorithm finds a peak by doing essentially a binary search on the matrices' columns.
b) Yes, the brute force algorithm will eventually terminate and find a peak.
c)
d) Yes, the algorithm finds a peak much in the same manner as algorithm1, except by alternating between columns and rows, it halves the time it takes to find the maximum for the middle row/column.

4 1-4

a) $nlog(n)$, we must search through n entries to find the column maximum. Since we are doing a binary search, this occurs $log(n)$ times.
b) The worst case complexity is n^2, since the algorithm can potentially search through every entry in the matrix.
c) The worst-case runtime is $O(2n) + O(n) + O(n/2) + ... = O(4n) = O(n)$.
d) $O(n)$, since each step only involves finding the maximum of the dividing column, which halves in length at each step.

5 1-5

1. If the peak problem is not empty, then algorithm1 will always return a location:
Say that we start with a problem of size $m \times n$. The recursive subproblem examined by algorithm1 will have dimensions $m \times \lfloor n/2 \rfloor$ or $m \times (n - \lfloor n/2 \rfloor - 1)$ if we are on an odd iteration of the algorithm. If we are on an even iteration of the algorithm, the recursive subproblem will have dimensions $\lfloor m/2 \rfloor \times n$ or $(m - \lfloor m/2 \rfloor - 1) \times n$. The number of columns or rows strictly decrease with each recursive call (alternatively, the number of entries in each recursive subproblem strictly decreases) as long as n,m¿0. So algorithm1 either returns a location at some point, or eventually examines a subproblem with a non-positive number of columns or rows. The only way for the number of columns or rows to become strictly negative according to the formulas is to have n or m equal 0 at some point. So if algorithm4 doesn't return a location, it must eventually examine an empty subproblem.
We wish to show that there is no way this can occur. Assume to the contrary,

that algorithm4 does examine an empty subproblem. Just prior tot his , it must examine a subproblem of one of the following sizes, $a \times 1, a \times 2, 1 \times b, 2 \times b$. If the problem has a dimension that is of size 1, then calculating the maximum of the central column is the same as calculating the m maximum of the entire problem. Hence that maximum must be a peak and our algorithm will halt and return that location. If the problem has 2 rows or 2 columns, it will either reduce to a subproblem with 1 row or 1 column within 2 steps, or we will find a maximum in the row or column. Thus algorithm4 can never recurse to an empty subproblem and must eventually return a location.
2. If algorithm4 returns a location, it will be a peak in the original problem.
If algorithm4 returns a location (r_1, c_1), then that location must have the best value in either r_1 or c_1 and must have been a peak in some recursive subproblem. Assume for the sake of contradiction that (r_1, c_1) is not also a peak in the original problem. Then as the location is passed up the chain of recursive calls, it must eventually reach a level where it stops being a peak. At that level, the location (r_1, c_1) must be adjacent to the dividing column or row where the value in the dividing column or row is greater than or equal to our maximum(or our value would continue to be a peak). The maximum value in the row or column found by our algorithm has to be greater than the value adjacent to (r_1, c_2). However, since the algorithm chose to recurse on the subproblem containing (r_1, c_1), the location in the same column as (r_1, c_1) adjacent to the maximum must be larger than the maximum in the dividing column. But this is a contradiction, because the value at (r_1, c_1) is the maximum in the column and cannot be both smaller than the value in the dividing column adjacent to it and larger than the maximum value in the dividing column.

6 Problem 1-6