CS3230 Tutorial 3

AY 25/26 Sem 1 — github/omgeta

Q1). (a) At start of iteration i, array $A[0..i-1]$ is sorted

(b) Initialization: At start of iteration $i = 1$, array $A[0..0]$ is trivially sorted.
Maintenance: Assuming $A[0..i-1]$ is sorted, the inner loops looks for the correct position to insert $A[i]$, leaving $A[0..i]$ sorted for the next iteration.
Termination: Loop ends after $i = N-1$, leaving $A[0..N-1]$ sorted.
(Optional) Inner Loop Invariant: At start of iteration j, array $A[j+1..i]$ consists of the original elements $\geq X$ shifted right by one for X to be inserted.

Q2). (a)

1. Base Case: if $n \leq 2$, swap if necessary and resultant array is trivially sorted
2. Inductive Step: Suppose StoogeSort correct sorts arrays of size $< n$.
   Let X, Y, Z denote the 3 thirds of the array.
   - 2.1. Sorting first $\lceil 2n/3 \rceil$ sorts X, Y by IH, so all elements in Y are $\geq$ elements in X.
   - 2.2. Sorting last $\lceil 2n/3 \rceil$ sorts Y, Z by IH, so all elements in Z are $\geq$ all elements in X, Y. Z is now finished.
   - 2.3. Sorting first $\lceil 2n/3 \rceil$ sorts remaining elements in X, Y by IH, and we're done.

(b) $T(n) = 3T(\lceil 2n/3 \rceil) + \Theta(1) \in O(n^2)$;
since $d = \log_{3/2} 3 \approx 2.71$ and $f(n) \in O(n^{\log_{3/2}+\epsilon})$ for $\epsilon = 0.000001$,
$\therefore$ by case 1 of Master Theorem, $T(n) \in \Theta(n^{\log_{1.5} 3}) = \Theta(n^{2.71})$

Q3). Take the maximum of any 2D array. By definition, the maximum is $\geq$ its neighbours, therefore there is always a peak which is the maximum.

Q4). $T(n) \in O(n^2)$;
since $d = \log_4 16 = 2$ and $f(n) = 32n \log^{128} n \in O(n^{2+\epsilon})$ for $\epsilon = 0.000001$,
$\therefore$ by case 1 of Master Theorem, $T(n) \in \Theta(n^2) \implies T(n) \in O(n^2)$

Q5). $T(m, n) = 2T(m, \lfloor n/2 \rfloor) + \Theta(m)$. There are $\log n$ levels, 2^k subproblems at each level and $\Theta(m)$ work for each subproblem. So overall we have total $\sum_{i=1}^{\log n} 2^i = n$ subproblems so $\Theta(mn)$ time.

Q6).

1. Base Case: if $n = 1$, maximal of column is trivially a peak because there are no left-right neighbours, and its $\geq$ up-down neighbours
2. Inductive Step: Assume there is a peak for all widths $< n$.
   - 2.1. If maximum of C_m is a peak, we're done
   - 2.2. Else, there must be a left-right neighbour greater. WLOG, if the left neighbour is greater, recurse on the left half. Then by IH, there must be a peak in the left half.

To optimize, we only recurse on the half with the larger neighbour of the current maximal element, giving $T(n) = T(m, \lfloor n/2 \rfloor) + \Theta(m) \in \Theta(m \log n)$