CS161 – Fall 2016 — Homework3

Jenny Zeng, SID 52082740, `zhaohuaz@uci.edu`

R-8.5, C-8.10, R-9.5, C-9.1

R-8.5

Consider again the modification of the deterministic version of the quick-sort algorithm so that, instead of selecting the last element in an n-element sequence as the pivot, we choose the element at index $\lfloor n/2 \rfloor$. Describe the kind of sequence that would cause this version of quick-sort to run in $\Theta(n^2)$ time.
Answer:
To achieve this goal, in every depth, the pivot should be the largest number or the smallest number in the current sequence. One example for this kind of sequence could be: $[0, 2, 4, 6, 8, 9, 7, 5, 3, 1]$, $n = 10$.

- depth 0: 9 is the pivot, which is the largest number in the current sequence. We will do quick-sort in $[0, 2, 4, 6, 8, 7, 5, 3, 1]$
- depth 1: 8 is the pivot, do quick-sort in $[0, 2, 3, 6, 7, 5, 3, 1]$
- depth 2: 7 is the pivot, do quick-sort in $[0, 2, 3, 6, 5, 3, 1]$
- ...
- depth 9: 0 is the pivot, and it is the only element in the sequence. So this is the base case. return [0]

Thus, in every depth, the size of subsequence is always n-1. The complexity is $T(n) = T(n-1) + O(n) = \Theta(n^2)$

C-8.10

Give an example of a sequence of n integers with $\Omega(n^2)$ inversions. (Recall the definition of inversion from Exercise C-8.9.)

Answer:
it can be a decreasing sequence, in which every element is larger than the next one.
e.g.
$[9, 8, 7, 6, 5, 4, 3, 2, 1]$

- for the first element in the list, there are $n - 1$ inversion.
- for the second element, $n - 2$ inversion.
- ...
- last element, 0 inversion.

# of inversion $= (n - 1) + (n - 2) + ... + 1 = \Omega(n^2)$

R-9.5

Show that the worst-case running time of quick-select on an n-element sequence is $\Omega(n^2)$.

Answer:
It happens if the size of recursive subproblem is always $n-1$
Quick-select is similar to quick-sort, it randomly chooses a pivot and put it into three lists: L(less than pivot), E(equal to pivot), G(greater than pivot)
Here is a worst case:
Assume we will get a sorted list in increasing order and we want to get the last element in this sorted list. Every time this partition happens, pivot is always the smallest element in the sub-list, size of list G is $n-1$. Because we want to get the last element in the list, this partition will continue to happen until $G = [lastElement]$ (only the last element is in the list).
Therefore, we can get: $T(n) = T(n-1) + O(n) = \Omega(n^2)$

C-9.1

Show that any comparison-based sorting algorithm can be made to be stable, without affecting the asymptotic running time of this algorithm.
Hint: Change the way elements are compared with each other.

Answer:
We can tag those elements if they are the same to determine their relative position. After sorting the sequence according to the value, we compare elements' relative positions and sort the sequence again according to their position. The worst case could be all the elements in the sequence have the same value. To sort it according to the position, we sort again, which will double the sorting time but will not affect the asymptotic running time of this algorithm.