HW#4
(EC 330)

Scott Hom
10/16/16

1.

a.) The worst case sequence for insertion sort is from bigger to smallest.

- ex:

1 40 35 30 25 20

2 35 40 30 25 20

3 30 35 40 25 20

4 25 30 35 40 20

The number of swaps would always be $\sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{n^2-n}{2}$

↳ by growth rates insertion sort is $\frac{n^2-n}{2} = \Theta(n^2)$. and to get $\frac{n^2-n}{2} = \Theta(n^2)$ it must be both $\Omega(n^2)$ and $O(n^2)$ so it is $\Omega(n^2)$.

b.) bucket sort worst case is $O(n^2)$ when all of the numbers filter into one bucket. The linear average case of bucket sort can be preserved by using merge sort instead of insertion sort to sort the buckets. Merge sort worst case is $O(n \log n)$, which overpowers the linear behavior of bucket sort from the sorting the # into buckets and force the worst case to $O(n \log n)$.

2.)

a.) The runtime on a sequence that is already sorted is $T(n) = T(\frac{n}{2}) + T(\frac{n}{2})$, because on very large inputs we can see that the input is initially n. However after the 1st cut the set is recursively sorted splitting into groups $\frac{1}{2}$. Even though for some case when n = odd input. the two sets would only differ by "1". At very large inputs a difference of "1" doesn't affect the execution of quick sort on the left and right groups formed.

8 5 3 2 1 5 3 4

b.) 8 6 4 2 1 3 5 7 would be both sequence that would be sort in $\Omega(n^2)$ time because, it would have to be sorted $T(\frac{n}{2}) + \frac{n}{2}$

$\sum_{i=1}^{n} i(i+1) + \Theta(1) = 2\sum_{i=1}^{n} i + \sum_{i=1}^{n} 1 = 2\left(\frac{n^2+n}{2}\right) + n + \Theta(1)$

$\Rightarrow n^2 + 2n + \Theta(1) = \Omega(n^2)$. For example while trying to get it so the pivot is always the largest/smallest #.

8 6 4 2 1 3 5 7

1 | 2 | 8 6 4 3 5 7 (sort 3 times)

1 2 | 3 | 4 | 8 6 5 7 (sort 3 times)

3 4 | 5 | 6 | 8 7 (sort

3.

a) Stable:
- insertion sort: stable because if let $j = i-1$ if $j > i$ swap. If not then keep # there.
- Merge sort: stable

Unstable:
- heap: either two are stable
- quick: because the sorting algorithm is executed in regular order. heap sort want to maintain max heap property and remove top and

for 2 $\overset{A}{1}$ $\overset{B}{1}$ 5, $\overset{A}{1}$ + $\overset{B}{1}$ 2 swap with each other.

quick sort split # based on relation to pivot

4) Unsorted Array = A[i] + mark blank array B[j] equal in length to A[j]. Use a sorting algorithm of O(n log n) such as merge sort. if two element compared are equal skip element

5) bool node_check(Node node, int k, int x, count, bool lessequal)
{
if (node.val > x){
{
count++
if(count == k){
lessequal = true;
return lessequal;
}
Check Node(node.left, k, x, bool lessequal)
Check Node(node.right, k, x, bool lessequal)
}