Adam Rich
EN.605.202.87.SP18 Data Structures
Homework Assignment 9
April 9, 2018

Assignment 9 – Sorting

Write pseudo-code not Java for problems requiring code. You are responsible for the appropriate level of detail.

1. Let's sort using a method not discussed in class. Suppose you have *n* data values in in array *A*. Declare an array called *Count*. Look at the value in *A[i]*. Count the number of items in *A* that are smaller than the value in *A[i]*. Assign that result to *count[i]*. Declare an output array *Output*. Assign *Output[count[i]] = A[i]*. Think about what the size of *Output* needs to be. Is it *n* or something else? Write a method to sort based on this strategy.

EXAMPLE 1

```
Assume that A looks something like this
A      = [1  0 -4 18 20  4  3  1 10  9  8 12]
Count  = [2  1  0 10 11  5  4  2  8  7  6  9]
i      = [0  1  2  3  4  5  6  7  8  9 10 11]
Output = [-4 0  1  *  3  4  8  9 10 12 18 20]
* represents NULL

Since there is no A[i] such that the count of elements less than A[i] = 3, the fourth
element of Output is blank.  But, Output still needs to have the same length as A.
```

EXAMPLE 2

```
Assume that A looks like this
A      = [1 1 1 1 1]
Count  = [0 0 0 0 0]
i      = [0 1 2 3 4]
Output = [1 * * * *]

In this case, Output need only have one element.

Output needs to have length equal to "count of A[i] < max[A] + 1"

The output of this sorting method will always look like a regular ascending sort of A,
but with duplicated skipped.
```

EXAMPLE 1 AGAIN

```
It would make sense to assume that the question really wants example 1 output to not
have a NULL in the middle of it, and that it should really look like this:

Output = [-4 0  1  3  4  8  9 10 12 18 20]

If this is correct, then output will always have length = "the number of unique
elements in A".
```

```
method QuickSortUnique(int[] A)
  S = quick sort A
  n = scan S and get count of unique elements
  Output = new int[n]
  scan S and put each unique element in Output
end-method
```

2. Analyze the cost of the sort you wrote in the previous problem. What is the impact of random, ordered, or reverse ordered data?

The cost will be the same as the underlying quicksort, which on average is O(N log N) plus two passes of size n, but those get swallowed up in the same O(N log N).

The quicksort can have worst case performance of O(n-squared) if the choice of pivot is the highest or lowest value in the array. This can happen when the first element is chosen as the pivot and the data are ordered or reverse ordered.

If the data are random, choosing the first element as the pivot for a quicksort should have O(N log N) complexity.

3. How many comparisons are necessary to find the largest and smallest of a set of n distinct elements?

N - 1 for largest and N - 1 for smallest if doing separately. If you want to check them at the same time then the number should be (N - 2) * 2 + 1.

To find the biggest, compare the A[0] to A[1], store the larger to L. Compare L to A[2], store the larger to L. Keep going to A[n-1], so comparisons where made to elements 1, 2, ..., N-1.

To do them both simultaneously, compare A[0] to A[1]. Store the smaller of the two in S, the larger in L (they have to be different). That is comaprison #1. Then compare S to A[2], store the smaller is S, and L to A[2], store the larger in L. That is comparisons #2 and #3. Elements 2 to N-1 get two comparisons, plus the first comparison. So the total is (N - 1 - 1) * 2 + 1 = (N - 2)*2 + 1 = **2N - 3**