EC 504
Spring, 2021
Homework 1

Due Saturday, Feb. 6, 2021

Note: This part of the HW is to be turned in to Gradescope as a PDF submission by 8 PM Boston time.

1. (30 pts) Here is a classical problem that is often part of interviews. You are given n nuts and n bolts, such that one and only one nut fits each bolt. Your only means of comparing these nuts and bolts is with a function TEST(x, y), where x is a nut and y is a bolt. The function returns +1 if the nut is too big, 0 if the nut fits, and -1 if the nut is too small. Note that you cannot compare nuts to nuts and bolts to bolts.

   The usual question is to design and analyze an algorithm for sorting the nuts and bolts from smallest to largest using the TEST function. While there are many approaches that work, the smart answer is to combine a pivot operation from quicksort to solve this as follows: Pick a bolt, and partition the nut array into nuts that are smaller, nut that fits, and nuts that are larger. Then, pick the matching nut, and partition the nut array similarly. Below is pseudo-code for this algorithm.

```
Procedure quicknutboltsort(nutarray,boltarray,m,n):
if m >=n, return;
new_nutarray[0:n-m] = -1; new_boltarray[0:n-m]=-1;
testbolt = boltarray[n];
low = 0; best_fit = -1; high = n-m;
For k in m to n,
     if Test(k,testbolt) < 0,
          new_nutarray[low++]=nutarray[k];
     else if Test(k,testbolt) == 0,
          best_fit = nutarray[k];
     else
          new_nutarray[high--] = nutarray[k];
new_nutarray[low] = best_fit;
// Now pivot the bolts against the best_fit nut.
low = 0; testnut = best_fit; best_fit = -1; high = m-n;
For j in m to n,
     if Test(testnut,j) < 0,
          new_boltarray[low++]=boltarray[k];
     else if Test(testnut,i) < 0,
          new_boltarray[high--]=boltarray[k];
     else
         continue;
new_boltarray[low] = testbolt;
// now recur:
quicknutboltsort(new_nutarray, new_boltarray, 0, low-1);
quicknutboltsort(new_nutarray,new_boltarray,low+1,n);
// copy back into original array
for k in m to n:
    nutarray[k] = newnut_array[k-m];
    boltarray[k] = newbolt_array[k-m];
```

   (a) What is the tightest worst-case asymptotic growth of the above function as $n \to \infty$?

   (b) To analyze the average case, assume that the value of `low` in the above algorithm is equally likely to be any number from 0 to m-n. Write a recursion for the average asymptotic run time for `quicknutboltsort(nutarray,boltarray,0,n)` as a function of n.

(c) Solve that recursion, using the following fact:

$$T(n) = cn + \frac{d}{n}\sum_{k=0}^{n-1} T(n-1) \iff T(n) \in \Theta(n\ln(n))$$

(d) The above algorithm uses lots of extra storage, copying arrays back and forth. Modify the above algorithm so that you use only the original nutarray and boltarray. You may want to look at implementations of quicksort to see how to sort in place.

2. (30 pts) Suppose you are given two sorted arrays A, B of integer values, in increasing order, sizes n and m respectively, and $m+n$ is an odd value (so we can define the median value uniquely). Assume $n < m$.

   (a) Assume arrays are indexed from 0, and that integer division rounds down. Show that, if $A[(n-1)/2] \leq B[(m-1)/2]$, the median value of the combined arrays is in the interval $[A[(n-1)/2], B[(m-1)/2]]$. Otherwise, show that the median value is in the interval $[B[(m-1)/2], A[(n-1)/2]]$.

   (b) Assume that, if $n \leq 2$, we can find the median in $O(\log(m))$ using binary insertion of the elements of A into B. Develop an algorithm for finding the median of the combined sorted arrays, with worst case complexity $O(\log(m))$, and write your algorithm in pseudocode as an answer.

   (c) Write a recursion for the run time of the algorithm as a function of m, the length of the longer array. Solve the recursion to show that your algorithm is $O(\log(m))$.

   (d) Demonstrate your algorithm by finding the median of the following two cases:

$$A = [0, 1, 2, 3]; \quad B = [4, 5, 6, 7, 8, 9, 10, 11, 12]$$

$$A = [7, 8, 9, 10]; \quad B = [0, 1, 2, 3, 4, 5, 6, 11, 12]$$