CS 430 – FALL 2023
INTRODUCTION TO ALGORITHMS
HOMEWORK #2

1. **(5 points) We have a problem that can be solved by an iterative (non-recursive) algorithm that operates in N^2 time. We also have a recursive algorithm for this problem that takes $N \lg N$ operations to divide the input into two equal pieces, two recursive calls to conquer, and $\lg N$ operations to combine the two solutions together. Show whether the iterative or recursive version is more efficient.**

2. **(6 points) During the running of the procedure RANDOMIZED-QUICKSORT, how many calls are made to the random-number generator RANDOM in the worst case? How about in the best case? Give your answer in terms of Θ-notation.**

Algorithm 1 Randomized-Partition pulled from the book.

```
1: function RANDOMIZED-PARTITION(A, p, r)
2:     i ← RANDOM(p, r)
3:     exchange A[r] with A[i]
4:     return PARTITION(A, p, r)
5: end function
```

Algorithm 2 Randomized-Quicksort pulled from the book.

```
1: function RANDOMIZED-QUICKSORT(A, p, r)
2:     if p < r then
3:         q ← RANDOMIZED-PARTITION(A, p, r)
4:         RANDOMIZED-QUICKSORT(A, p, q − 1)
5:         RANDOMIZED-QUICKSORT(A, q + 1, r)
6:     end if
7: end function
```

3. **(5 points) Give pseudocode, code or a description of an algorithm to rearrange the objects in array so that the objects with odd number keys come first (in any order, not necessarily sorted), followed by the objects with even numbers (also any order), returning the position of the first even number object. What is the big-O runtime of your algorithm?**
   So, for example, $\{1, 0, 8, 7, 5, 3, 9, 6, 2, 7, 4\}$ might become $\{1, 7, 9, 7, 5, 3, 8, 6, 2, 0, 4\}$ and the index 7 (the new position of value 8) is returned.

4. **Suppose you are given k n-element sorted sequences A_i. Each sequence has no duplicate entries, but there may be duplicates between sequences. Describe an $O(nk \log k)$-time method that uses a minheap for computing a sorted sequence representing the union of all A_i sequences, with no duplicates. Discuss why your algorithm is $O(nk \log k)$.**

5. **This problem requires you to use a heap to store items, but instead of using a key of an item to determine where to store the item in the heap, you need to**

assign a priority (key) to each item that will determine where it is inserted in the heap

5a) (4 points) Show how to assign priorities to items to implement a first-in, first-out queue with a heap.

5b) (4 points) Show how to assign priorities to items to implement a first-in, last-out stack with a heap.

6. (6 points) Show that the second smallest of n elements can be found with $(n - 1) + (\lceil \lg n \rceil - 1)$ comparisons in the worst case. Use a tournament (upside down binary tree) approach to also find the smallest element and think about where the second-smallest element might be in the tournament. Hint: The number of comparisons shown above is shown as two terms because there are two steps to the algorithm.

7. (4 points) The following comparison tree sorts the four values a, b, c, and d.

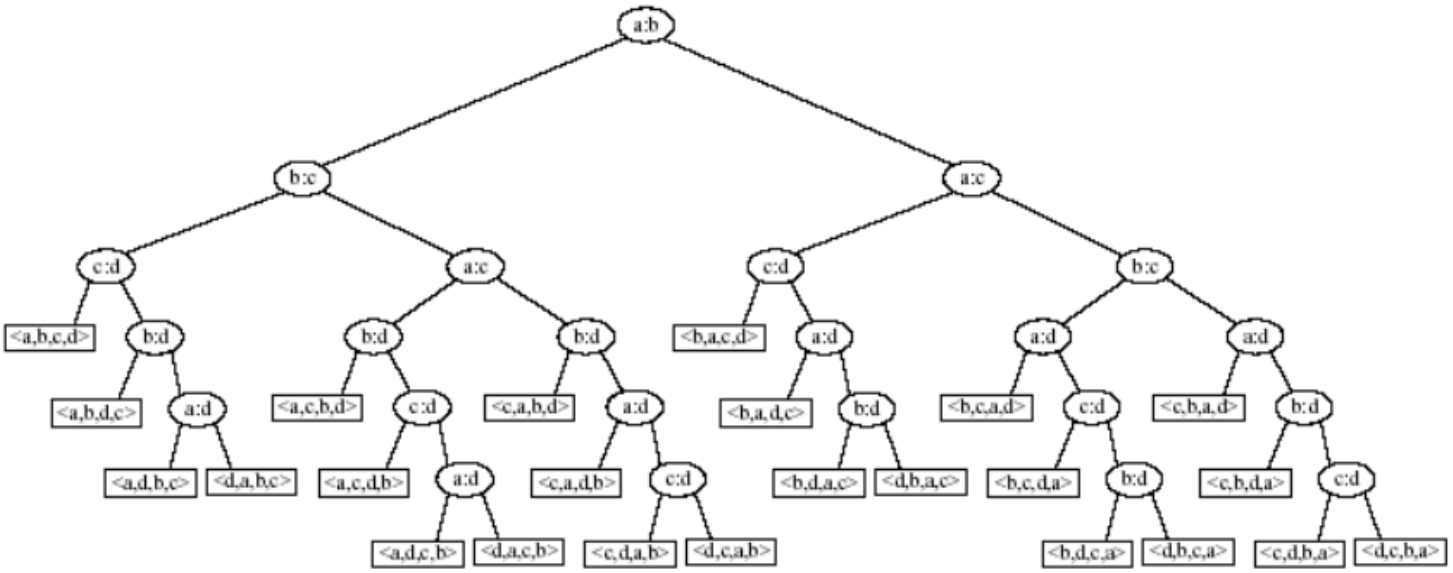

Figure 2.1:

7a) What is the worst-case number of comparisons performed by the comparison tree?

7b) What is the best-case number of comparisons performed by the comparison tree?

7c) What is the average-case number of comparisons performed by the comparison tree, assuming that any permutation of the four inputs is equally likely?