CS:3330 Algorithms
Midterm Exam (100 points)

Closed books and notes (except one sheet of notes)
None of the digital devices is allowed.
3/26/2019

1. (30 points) We insert the following numbers in the given order into an empty binary search tree. (**a)** If the tree is an AVL tree, for each rotation in an insertion, please display the tree before and after rotation. If no rotation happens during an insertion, simply draw the tree after the insertion. Please repeat the above task when the tree is (**b)** a Red-Black tree, or (**c)** a splay tree:

   7, 1, 2, 8, 6, 4.

2. (30 points) We like to sort the array A = [7, 1, 6, 3], and the only operation to change the array A is by swap(i, j), which swaps the elements of A at positions i and j.
   (**a)** If heapsort is used, please illustrate how the heap is constructed and changed by the swap operation (show the content of the array A after each swap operation).
   (**b)** If quicksort is used (and the first element of the subarray is the pivot), please show the content of the array A after each swap operation.

3. (40 points) Given two lists A and B, append(A, B) will create a new list C consisting of elements from A and B with cost O(|A| + |B|), where |X| is the size of X, i.e., the number of elements in list X.
   (**a)** Please design an efficient algorithm (as fast as you can) which appends n lists of the same size m, into a single list by calling append(A, B). Please analyze the complexity of your algorithm in terms of n and m.
   (**b**) If the sizes of these n listings are 1, 2, …, n, respectively, please repeat the question in (**a)**.

Answer:

(a) We may use the idea of merge sort to append all lists into one. Let **S** = { S_1, S_2, …, S_n }, each S_i is a list of m elements. The pseudo code of the algorithm is given as follows:

```
appendAll(S)
  let S = { S1, S2, …, Sn }
  if  n = 1 return S1
  d = (n+1)/2
  A = appendAll({S1, S2, …, Sd})
  B = appendAll({Sd+1, ..., Sn})
  return append(A, B)
```

Let T(n, m) be the complexity of appendAll(**S**). Then T(n, m) = 2T(n/2, m) + cmn, where cmn is the cost of append(A, B) and c is a constant. The solution is O(mn*log*(n)).

(b) Without loss of generality, we assume n is even (if not, we add an empty list). At first, we pair the list of size 1 to the list of size n, the list of size 2 to the list of size n-1, and so on. We then call append(A, B) to append each pair of lists into one list. The whole time takes O(n/2(n+1)), or O(n^2). The result **R** is a set of n/2 lists of size n+1. Then we call appendAll(**R**). From (a), we know the complexity is O(mn/2*log*(n/2)), where m = n+1. So the total complexity is O((n+1)n/2*log*(n/2)+ n^2), or O(n^2*log*(n)). The pseudo code looks as follows:

```
appendAll2(S)
    let S = { S1, S2, …, Sn }
    R = {}
    for k from 1 to n/2
        R = R U { append(Sk, Sn-k+1) }
    return appendAll(R)
```