ICS 311, Fall 2020, Problem Set 05, Topics 9 & 10A

Firstname Lastname <uhumail@hawaii.edu> (Section 1)

Due by midnight Tuesday October 13. 40 points total.

#1. Analysis of d-ary heaps (15 pts)

In class you did preliminary analysis of ternary heaps. Here we generalize to *d*-ary heaps: heaps in which non-leaf nodes (except possibly one) have *d* children.

a. (5) How would you represent a *d*-ary heap in an array with 1-based indexing? Answer this question by:

- Giving an expression for **Jth-Child(i,j)**: the index of the *j*th child as a function of *j* and the index *i* of a given node, and
- Giving an expression for **D-Ary-Parent(i)**: the index of the parent of a node as a function of its index *i*.
- Checking that your solution works by showing that D-Ary-Parent(Jth-Child(i,j)) = i (Show that if you start at node i, apply your formula to go to a child, and then your other formula to go back to the parent, you end up back at i).

b. (2) What is the height of a *d*-ary heap of *n* elements as a function of *n* and *d*? By what factor does this height differ from that of a binary heap of n elements?

c. (4) Give an efficient implementation of EXTRACT-MAX in a *d*-ary max-heap. (Hint: consider how you would modify existing code.) Analyze its running time in terms of *n* and *d*. (*Note that d must be part of your Θ expression.*)

d. (4) Give an efficient implementation of INSERT in a *d*-ary max-heap. Analyze its running time in terms of *n* and *d*.

#2. Quicksort Pathology (7 pts)

The point of this question is to show that data patterns other than strictly sorted data can be problematic in non-randomized Quicksort.

a. (3) Trace the operation of a single call to Partition (A, 1, 9) (not randomized) on this 1-based indexing array:

A = [1, 6, 2, 8, 3, 9, 4, 7, 5], p=1, q=9

Show the state of A after the call and the value Partition returns.

b. (2) On what subarray will Quicksort in line 3 be called?

On what subarray will Quicksort in line 4 be called?

c. (2) How are the keys organized in the two partitions that result? How do you expect that this behavior will affect the runtime of Quicksort on data with these patterns?

#3. 3-way Quicksort (18 pts)

In class we saw that the runtime of Quicksort on a sequence of *n* identical items (i.e. all entries of the input array being the same) is $O(n^2)$. All items will be equal to the pivot, so n-1 items will be placed to the left. Therefore, the runtime of QuickSort will be determined by the recurrence $T(n) = T(n-1) + T(0) + O(n) = O(n^2)$ To avoid this case, and to handle duplicate keys in general, we are going to design a new partition algorithm that partitions the array into three partitions, those that are strictly less than the pivot, those equal to the pivot, and those strictly greater than the pivot.

a. (10) Develop a new algorithm *3WayPartition(A, p, r)* that takes as input array *A* and two indices *p* and *r* and returns a pair of indices (*e, g*). *3WayPartition* should partition the array *A* around the pivot $q = A[r]$ such that every element of *A[p..(e-1)]* is strictly smaller than *q*, every element of *A[e..g-1]* is equal to *q* (e indicates the start of "equal" keys), and every element of *A[g..r]* is strictly greater than *q* (g indicates the start of "greater" keys). Explain why your code is correct.

Hint: modify Partition(A,p,r) presented in the lecture notes/book, such that it adds the items that are greater than q from the right end of the array and all items that are equal to q to the right of all items that are smaller than q. You will need to keep additional indices that will track the locations in A where the next item should be written.

b. (4) Develop a new algorithm *3WayQuicksort* that uses *3WayPartition* to sort a sequence of *n* items, keeping in mind that *3WayPartition* returns a pair of indices (e, g).

c. (4) What is the runtime of *3WayQuicksort* on a sequence of *n* random items? What is the runtime of *3WayQuicksort* on a sequence of *n* identical items? Justify your answers.