#1. Peer Credit Assignment

1 Point Extra Credit for replying

Please list the names of the other members of your peer group for class 2/17 and 2/24 and the number of points you think they deserve for their participation in group work on the two days combined.

- You have a total of 6 points to allocate across all of your peers.
- You can distribute the points equally, give them all to one person, or do something in between.
- You need not allocate all the points available to you.
- You cannot allocate any points to yourself! Points allocated to yourself will not be recorded.

#2. Analysis of d-ary heaps (11 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. (3) How would you represent a d-ary heap in an array? Answer this question by

- Giving an expression for \texttt{Jth-Child(i,j)}: the index of the jth child as a function of j and the index i of a given node, and
- Giving an expression for \texttt{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 (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. (3) 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 \(\Theta\) expression.)

d. (3) Give an efficient implementation of INSERT in a \(d\)-ary max-heap. Analyze its running time in terms of \(n\) and \(d\).

---

**#3. Tracing Quicksort (12 pts)**

Show the operation of Partition (not randomized) on this 1-based array:

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

and the two sub-partitions that result as directed below. In other words, you will trace the three calls to Partition that are highest in the recursion tree. (They are not the first three calls: part a is for the first call and part b is for the second call, but part c is for the call that takes place after all the recursive calls breaking down the first partition.)

In order to make the desired response format clear and to make it easy for the TA to grade, we are providing a template for your response. You are to fill in wherever the underscore character "_" appears. Use a plain text editor with fixed-width font. Be sure to fill in all fields marked with underscore: use search to make sure you get them all. We start you off with the first few lines: continue in the same pattern.

**a. (4) Call to Partition (A, 1, 9) made in Line 2 of the initial call to Quicksort:**

Initially:

\[ A = [1, 6, 2, 8, 3, 9, 4, 7, 5], i=0, j=1, pivot = A[r] = A[9] = 5 \]

Trace at the conclusion of each pass through the loop lines 3-6

\[ A = [1, 6, 2, 8, 3, 9, 4, 7, 5], i=1, j=1, exchanged A[1] with A[1] \]
\[ A = [1, 6, 2, 8, 3, 9, 4, 7, 5], i=1, j=2, no exchange \]

... you fill in the rest until the loop exits ...

\[ A = [_, _, _, _, _, _, _, _], i=_, j=4, _ _ _ _ _ _ _ _ _ \\
A = [_, _, _, _, _, _, _, _], i=_, j=5, _ _ _ _ _ _ _ _ _ \]
A = [ _, _, _, _, _, _, _, _ ], i=_, j=6, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=7, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=8, __________

After the swap in line 7:
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, exchanged A[_] with A[_]

What does Partition(A, 1, 9) return? __

Continuing execution of the top level call to Quicksort, identify the two partitions that will be handled by the recursive calls to Quicksort at this level:
(#2) On what subarray will Quicksort in line 3 be called? A[_, _]
(#3) On what subarray will Quicksort in line 4 be called? A[_, _]

Now in parts b and c we trace these two calls in a manner similar to above.

b. (3) Call to Partition handled in the first call to Quicksort line 3:

Initially:

Trace at the conclusion of each pass through the loop lines 3-6
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________

After the swap in line 7:
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, exchanged A[_] with A[_]

What does this second call to Partition return? __

c. (3) Call to Partition handled in the first call to Quicksort line 4:

Initially:

Trace at the conclusion of each pass through the loop lines 3-6
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________
A = [ _, _, _, _, _, _, _, _ ], i=_, j=_, __________
After the swap in line 7:
\[ A = [\_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ , \_ ] , \ i = \_ , \ j = \_ , \ \text{exchanged } A[\_] \text{ with } A[\_] \]

What does this third call to Partition return? __

d. (2) What patterns do you see in the second and third calls to Partition? How do you expect that this behavior will affect the runtime of Quicksort on data with these patterns? (You may answer without doing a formal analysis.)

#4. 3-way Quicksort (10 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, we are going to design a new partition algorithm that partitions the array into three parts, those that are strictly less than the pivot, equal to the pivot, and strictly greater than the pivot.

a. (5) Develop a new algorithm \( 3\text{WayPartition}(A, p, r) \) that takes as input array \( A \) and two indices \( p \) and \( r \) and returns a pair of indices \( (e, g) \). \( 3\text{WayPartition} \) 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 \( q \) and every element of \( A[g..r] \) is strictly greater than \( q \).

Hint: modify \( \text{Partition}(A,p,r) \) 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. (3) Develop a new algorithm \( 3\text{WayQuicksort} \) which uses \( 3\text{WayPartition} \) to sort a sequence of \( n \) items. Again, \( 3\text{WayPartition} \) returns a pair of indices \( (e, g) \).

c. (2) What is the runtime of \( 3\text{WayQuicksort} \) on a sequence of \( n \) random items? What is the runtime of \( 3\text{WayQuicksort} \) on a sequence of \( n \) identical items? Justify your answers.