Data Structures and Algorithms – (COMP SCI 2C03)
Winter 2021
Tutorial-3

January Feb 8, 2021

**Question 1.** Draw the recursion tree for the recursive function $T(n) = T(n/10)+T(9n/10)+\Theta(n)$. Based on the recursion tree explain that $T(n) \in \Theta(n \log_2 n)$.

**Answer.** Figure 1 shows the recursion tree for this recurrence. Notice that every level of the tree has cost $cn$ (in order of $\Theta(n)$), until the recursion reaches a boundary condition at depth $\log_{10} n = \Theta(\log n)$, and then the levels have cost at most $cn$. After this level, the cost of each level is going to be less than $c(n)$ since one branch is ended and when we go further, other branches will end as well, until the level $\log_{10/9} n$. The recursion terminates at depth $\log_{10/9} n = \Theta(\log n)$. The total is therefore $O(n \log n)$.

**Question 2.** Show, in the style of the quicksort trace given in class, how quicksort sorts the array E A S Y Q U E S T I O N (for the purposes of this exercise, ignore the initial shuffle).

**Answer.**
Input: E A S Y Q U E S T I O N
Pivot (index 0): E
$\{E\ A\ (E)\ Y\ Q\ U\ S\ S\ T\ I\ O\ N\}$
Pivot (index 0): E
$\{\ A\ (E)\ \}\ E\ Y\ Q\ U\ S\ S\ T\ I\ O\ N$
Pivot (index 3): Y
$A\ E\ E\{N\ Q\ U\ S\ S\ T\ I\ O\ (Y)\ \}$
Pivot (index 3): N
$A\ E\ E\ \{\ I\ (N)\ U\ S\ S\ T\ Q\ O\ \}\ Y$
Pivot (index 5): U
$A\ E\ E\ I\ N\ \{\ O\ S\ S\ T\ Q\ (U)\ \}\ Y$

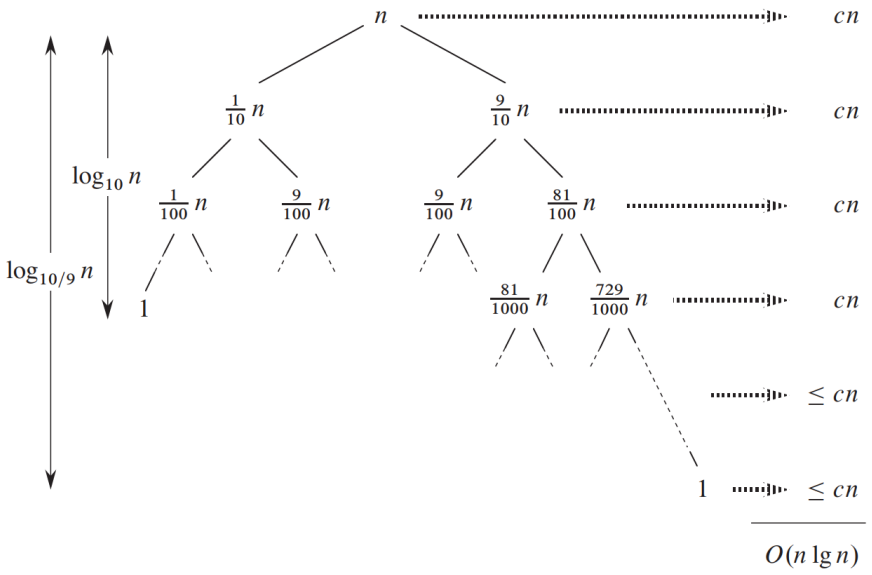


Figure 1: Recursion tree.

Pivot (index 5): O
*A E E I N* { (*O*) *S S T Q* } *U Y*
Pivot (index 6): S
*A E E I N O* { *Q* (*S*) *T S* } *U Y*
Pivot (index 8): T
*A E E I N O Q S* { *S* (*T*) } *U Y*

**** Result ***
*A E E I N O Q S S T U Y*

**Question 3.** Explain what happens when Quick.sort() is run on an array having items with just two distinct keys

**Amswer.** On first call of the function *partition*$(a)$, the array will be halved into two sub arrays, each contains a distinct value. After that, quick sort will perform the worst case scenario on each sub array since the pivot will halve each sub array of size $s$ into an empty sub array and a sub array of size $s - 1$.

**Question 4.** About how many compares will Quick.sort() make when sorting an array of N items that are all equal? **Answer.** If we have N items that are all equal, then the partition function will compare all the elements

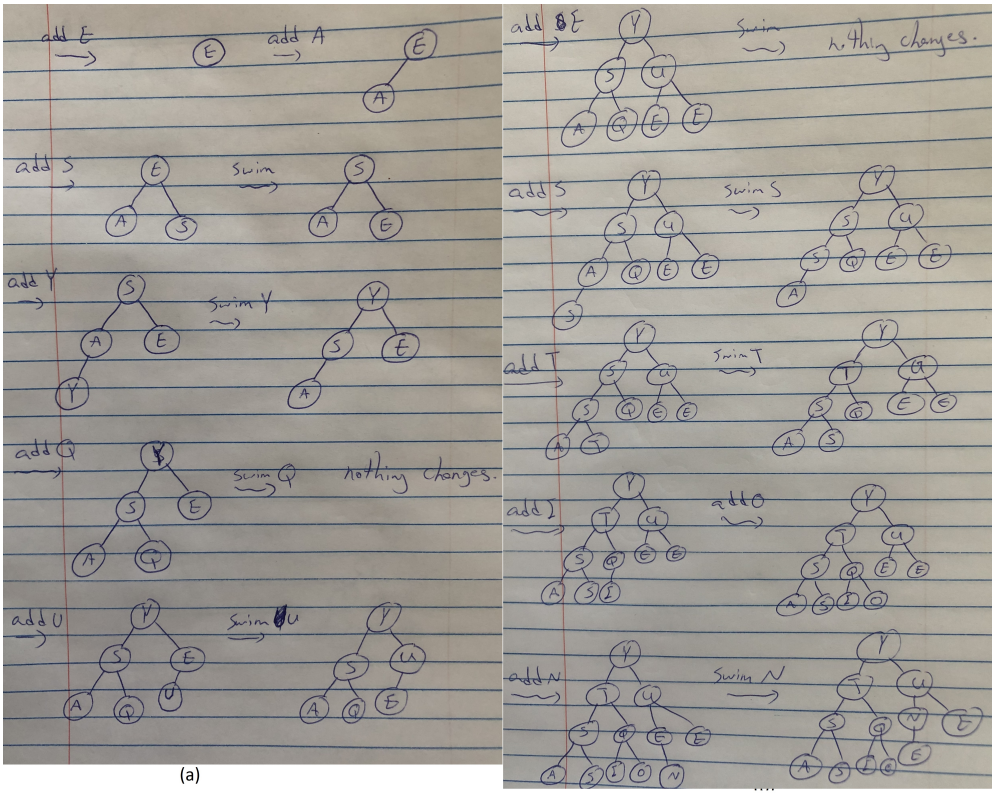


Figure 2: Max Heap step by step.

with the pivot and at the end swap the pivot with the last element. The result of the partition is an empty sub array and a sub array of size $N-1$. This means the number of the number of compares would be $n-1+n-2+...+ = O(n^2)$ and the number of swap is in order of $O(n)$.

**Question5.** Give the heap that results when the keys E A S Y Q U E S T I O N are inserted in that order into an initially empty max-oriented heap.

**Answer.** The answer is provided in Figure 2. The final max heap array would be [Y, T, U, S, Q, N, E, A, S, I, O, E].

**Question6.** Suppose that your application will have a huge number of insert operations, but only a few remove the maximum operations. Which priority-queue implementation do you think would be most effective: heap, unordered array, or ordered array?

**Answer.** Unordered Array. Because we can insert in constant time and find the maximum on $O(n)$ for just a few times (if less than $\log n$ times).

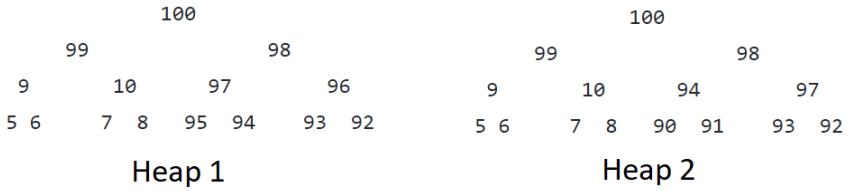


Figure 3: Sample Heaps.

**Question7.** What is the minimum number of items that must be exchanged during a remove the maximum operation in a heap of size $N$ with no duplicate keys? Give a heap of size 15 for which the minimum is achieved. Answer the same questions for two and three successive remove the maximum operations.

**Answer.** The minimum number of items that needs to be exchanged during a remove the maximum operation in a heap of size $N$ with no duplicate keys is 2. This happens when the last node to be exchanged with the root is greater than all the nodes in distance of 2 of the root in left (or right) sub tree of the root. For example, consider Heap1 of Figure 3. For 2 and 3 successive remove the maximum operation, we need minimum 5 and 8 comparisons respectively. The sample Heap2 of Figure 3 illustrates this case.

**Question8.**The largest item in a heap must appear in position 1, and the second largest must be in position 2 or position 3. Give the list of positions in a heap of size 31 where the $k$-th largest (i) can appear, and (ii) cannot appear, for $k = 2, 3, 4$ (assuming the values to be distinct).

**Answer.**

The height of the heap equals to $h = \log n$ ($h(root) = 0$). Assuming the size of the heap is 31, the $2^{nd}$ largest item can only appear at height 1 (positions $\{2, 3\}$) and cannot appear at $h = 0$ or $h > 1$. So, the positions that it cannot be appeared are $\{1, 4, 5, ..., 31\}$.

The $3^{rd}$ largest item can only appear at $h = 1$ or $h = 2$ (positions $\{2, 3, ..., 7\}$) and cannot appear at $h = 0$ or $h > 2$. So, the positions that it cannot be appeared are $\{1, 8, 9, ..., 31\}$.

The $4^{th}$ largest item can only appear at $h = 1$, $h = 2$ or $h = 3$ (positions $\{2, 3, 4, ..., 15\}$) and cannot appear at $h = 0$, $h = 1$ and $h > 3$. So, the positions that it cannot be appeared are $\{1, 16, 17, ..., 31\}$.