

COS 226 – Data Structures and Algorithms
Fall 2014 – Flipped Lecture Section
Individual or small group worksheet week 4 – 10.02.14
25 minutes

Instructions: Answer these solo or in small groups (2-3).

1. Apply 2-way partitioning to the set: S S S O B E R R Show the array after each exchange in one partition run.

ANS: S S S O B E R R → E R R O B S S S

2. Apply 3-way partitioning to the set: S S S O B E R R Show the array after each exchange in one partition run.

ANS: S S S O B E R R → O S S S B E R R → O B S S S E R R → O B E S S S S R R → O B E R S S S R → O B E R R S S S

3. Suppose that a mysterious data set (not seen by human eye) is randomized and yet the runtime of the quicksort ended up quadratic. What could be a possible reason for this?

ANS: Most likely the data set had many duplicates and 2-way quicksort with no stop at equal elements was applied

4. The quick-select algorithm for finding median was applied to a non-randomized data set. What is the worst case runtime of the algorithm?

ANS: It is proven that (theoretically) that the worst case run time for quick-select is still linear. But the construction to make that happen is not practical. So there is still an opportunity to find a “practical” linear time quick-select algorithm

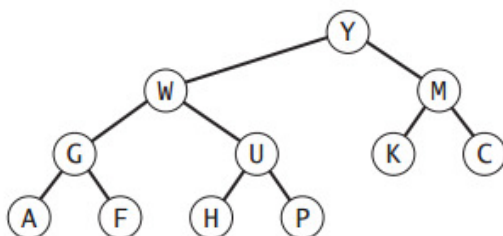
5. Assuming at each partition step, we were lucky to get median as the pivot. Write down a recurrence relation that represents the quicksort and argue that it is linearithmic.

ANS: $T(N) = 2 T(N/2) + N \rightarrow T(N) \sim N \lg N$

6. Assuming at each partition step, we get 1/3 and 2/3 split of the array. Write down a recurrence relation that represents the runtime of the quicksort.

ANS: $T(N) = T(N/3) + T(2N/3) + N$

7. Consider the following binary max heap. Show the original array that represents this heap and the array after Max is deleted



ANS: Y W M G U K C A F H P (original) W U M G P K C A F H (after max is deleted)

8. Why is it impossible to build a PQ that supports insert() and delMax() in constant time?

ANS: it is impossible to get constant performance w/o an array implementation. But if we keep insert() constant time, then delMax would be linear and vice versa. So at least we try to keep both $\lg N$ by using a binary heap

9. Design an implementation of a PQ that supports delMax() in constant time.

ANS: Suppose we keep max at the end of the array and so delete would be constant time. However, we will pay a linear price to insert an element since we need to make sure current max is always at the end of the array

10. Suppose we use a linked list instead of an array to implement a binary heap. Do we expect the running time to be better or worse? Memory usage?

ANS: probably not a good idea. First of all for each node, we have to keep two pointers of extra memory. So we will be using at least $16N$ more memory for a heap of size N . Actual runtime may be more as arrays are well cached and linked lists are not.

11. We can use binary search to find out where a new item should be inserted in the heap (instead of sinking one-by-one) in $\log(\log(N))$. Why don't we do this?

ANS: Even if you find a place to insert, you will still need to do $\lg N$ work to move the element into the right place.

12. Suppose we used a linked data structure, which allows constant time insertion. Why can't we use the binary-search trick from #4?

ANS: You cannot do binary search on a linked list. Linked lists are not random access

13. Suppose we use bottom-up SWIMMING (i.e. every node at the lowest level is swum, then every node at the next level up is swum, etc.). Is this procedure guaranteed to result in a heap?

ANS: No. we can give a counter example. Try the heap represented by array: x 1 2 3 4 5 6 7 to make a maxHeap by swimming up

14. For each of the following sorts, write down an invariant that is true during the execution of the algorithm after i -steps

(a) selection sort → All elements in $A[0..i]$ is Less than $A[i+1..n-1]$

(b) insertion sort → is $A[0..i-1]$ is sorted

(c) top-down merge sort Subarrays of size power of 2 can be sorted at any given time

(d) heap sort → $A[i+1..n-1]$ is the largest $n-i$ elements

15. Binary heap is not cache friendly. Explain why?

ANS: The fact the children of $A[i]$ are $A[2i]$ and $A[2i+1]$ makes it difficult to keep elements together in cache ($A[i]$ and $A[2i]$ may be in different blocks)