Algorithms* Lab 3

Review

Topics covered this week:

- heaps and heapsort
- quicksort

In lab exercises (COLLABORATION LEVEL: 0)

The in-lab problems are meant to be be solved during the lab and to generate discussion and sharing of ideas. Work with your team at your own pace and make sure to ask lots of questions. You need to turn in your in-class work, but will not be graded.

Heaps

1. (CLRS 6.1-1, 6.1-2)

   Consider a heap of height h, where the height is defined as the number of edges on the longest root-to-leaf path.

   (a) What is the minimum number of elements in the heap, as a function of h?

   (b) What is the maximum number of elements in the heap, as a function of h?

*csci2200, Laura Toma, Bowdoin College

(c) Use these to derive an $O()$ and $\Omega()$ bound for h as function of n, and argue that an n-element heap has height $\Theta(\lg n)$.

2. (CLRS 6.1-3) Where in a min-heap might the largest element reside, assuming that all elements are distinct?

3. (CLRS 6.1-5) Is an array that is in sorted order a min-heap?

4. (CLRS 6.1-7) Argue that the leaves in a heap of n elements are the nodes indexed by $\lfloor n/2 \rfloor + 1, ..., \lfloor n/2 \rfloor + 2, ..., n$.

5. What is the effect of calling HEAPIFY(A, i) for $i > size[A]/2$?. Recall that i is the index of the node where HEAPIFY is called.

6. (CLRS 6.5-2) Illustrate the operation of HEAP-INSERT (A, 7) on the heap (note: this is a min-heap):

$$A = \{2, 5, 10, 6, 8, 100, 11, 9, 15, 9, 10, 200, 101\}$$

7. (CLRS 6.2-1) Illustrate the operation of HEAPIFY(A, 1) on

$$A = \{(20, 5, 10, 6, 8, 100, 11, 9, 15, 9, 10, 200, 101, 12\}$$

8. (CLRS 6.3-1) Illustrate the operation of BUILD-MAX-HEAP on the array

$$A = \{5, 3, 17, 10, 84, 19, 6, 22, 9\}$$

9. (CLRS 6.4-1) Illustrate the operation of Heapsort on the array

$$A = \{5, 13, 2, 25, 7, 17, 20, 8, 4\}$$

10. How would you implement a function that searches for a given element in a heap, and how long would it take in the worst case?

Quicksort

1. Below is the pseudocode for Quicksort that we talked about in class. As usual with recursive functions on arrays, we see the array indices p and r as arguments. Quicksort(a, p, r) sorts the part of the array between p and r inclusively. The initial call (that is, to sort the entire array) is Quicksort$(A, 0, n-1)$.

```
QUICKSORT(A, p, r)
IF p < r THEN
    q=PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
FI
```

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
FOR j = p TO r − 1 DO
    IF A[j] ≤ x THEN
        i = i + 1
        Exchange A[i] and A[j]
    FI
OD
Exchange A[i + 1] and A[r]
RETURN i + 1
```

Let $A = \{3, 6, 1, 5, 8, 2, 4, 1, 3\}$, and assume we call Quicksort$(A, 0, 8)$. Show what happens during the first invocation of Partition. What is the value of q returned, and what are the two recursive calls made?

2. What is the running time of QUICKSORT when all elements of arrary A have the same value?

Homework COLLABORATION LEVEL : 1

1. Come up with an algorithm that finds the kth smallest element in a a set of n distinct integers in $O(n + k \lg n)$ time.

2. (C-4.9) Suppose we are given a sequence S of n elements, each of which is colored red or blue. Assuming S is represented as an array, give an $O(n)$ and in-place method for ordering S so that all blue elements are listed before all the red elements.

3. (CLRS 6.5-9) Assume you have k sorted lists containing a total of n elements, and you want to merge them together in a single (sorted) list containing all n elements. For simplicity you may assume that the k lists contain the same number of elements.

   (a) Approach 1: merge list 1 with list 2, then merge the result with list 3, then merge the result with list 4, and so on. What is the worst-case running time ?

   (b) Approach 2: split the k lists into two halves, merge each one recursively, then use the standard 2-way merge procedure (from mergesort) to combine the two halves. What is the worst-case running time ?

   (c) Give another approach (to merge the k lists) that uses a heap, and runs in $O(n \lg k)$-time.

4. (CLRS 7-3) Professors Dewey, Cheatham, and Howe have proposed the following "elegant" sorting algorithm:

   STOOGE-SORT(A, i, j)
   if $A[i] > A[j]$: swap $A[i] \leftrightarrow A[j]$
   if $i + 1 \geq j$: return
   $k \leftarrow \lfloor (j - i + 1)/3 \rfloor$
   STOOGE-SORT$(A, i, j - k)$
   STOOGE-SORT$(A, i + k, j)$
   STOOGE-SORT$(A, i, j - k)$

   (a) Correctness:

      i. Argue that STOOGE-SORT$(A, 1, length[A])$ correctly sorts any array of one element.

      ii. Argue that STOOGE-SORT$(A, 1, length[A])$ correctly sorts any array of two elements.

      iii. Would the algorithm work if the first line (that swaps elements $A[i]$ and $A[j]$) was missing?

      iv. Then argue that STOOGE-SORT$(A, 1, length[A])$ correctly sorts any array of three elements.

      v. Assume that STOOGE-SORT sorts sorrectly any array of $2n/3$ elements, and argue that this implies that it sorts correctly any array of n elements (What is true after the first recursive call? After the second?)

   (b) Running time: Give a recurrence for the worst-case running time of STOOGE-SORT and a tight asymptotic (Θ-notation) bound on the worst-case running time.

Notes on grading

You need to write each problem on a separate sheet of paper (do not forget to write your name). Each problem will be graded by a different TA.

Your assignment will be evaluated based not only on the final answer, but also on clarity, neatness and attention to details.

When you describe an algorithm, use high-level pseudocode. Focus on clarity, and don't forget to argue why the algorithm computes what it's supposed to compute (correctness), and to analyse its running time. You need to do this even if the problem does not specifically ask for it.

Generally speaking, it's not about getting the right answer, but about communicating it in a style that makes it easy to understand and convincing.