CS 61B Discussion 13 Spring 2016

1 Quicksort

(a) Sort the following unordered list using in-place quicksort. Assume that the pivot you use is always the first element and that we use the 3-way merge partitioning process described in lecture and lab last week. Show the steps taken at each partitioning step.

```
18, 7, 22, 34, 99, 18, 11, 4
```

```
-18-, 7, 22, 34, 99, 18, 11, 4
-7-, 11, 4 | 18, 18 | 22, 34, 99
4, 7, 11, 18, 18 | -22-, 34, 99
4, 7, 11, 18, 18, 22 | -34-, 99
4, 7, 11, 18, 18, 22, 34, 99
```

(b) What is the worst case running time of quicksort? Give an example of a list that meets this worst case running time.

$\Theta(n^2)$. Running quicksort on a sorted list will take $\Theta(n^2)$ if the pivot chosen is always the first or last in the subarray. In general, the worst case is such that the partitioning scheme repeatedly partitionins an array into one element and the rest. At each level of recursion, you will need to do $\Theta(n)$ work, and there will be $\Theta(n)$ levels of recursion. This sums up to $1+2+...+n$.

(c) What is the best case running time of quicksort? Briefly justify why you can't do any better than this best case running time.

$\Theta(n \log n)$. The optimal case for quicksort occurs if you can choose a pivot such that the left partition and right partition are of equal sizes. At each level of recursion, you will need to do $\Theta(n)$ work, and there will be $\Omega(\log n)$ levels of recursion.

(d) What are two techniques that can be used to reduce the probability of quicksort taking the worst case running time?

1. Randomly choose pivots. 2. Shuffle the list before running quicksort.

2 Comparing Sorting Algorithms

When choosing an appropriate algorithm, there are often several tradeoffs that we have to consider. For the following sorting algorithms, give the expected space complexity, time complexity, and whether or not each sort is stable.

	Time Complexity	Space Complexity	Stable?
Insertion Sort	$\Theta(n^2)$	1	Yes
Heapsort	$\Theta(n \log n)$	1	No
Mergesort	$\Theta(n \log n)$	$\Theta(n)$	Yes
Quicksort	$\Theta(n \log n)$	$\Theta(\log n)$	No

(a) For each unstable sort, give an example of a list where the order of equivalent items is not preserved.

```
Heapsort: 1a, 1b, 1c
Quicksort: 1, 5a, 2, 5b, 3
```

(b) In general, what are some other tradeoffs we might want to consider when designing an algorithm?

```
1. Readability when other engineers are using your algorithm.

2. Constant factors in runtime, especially when working with small
   inputs.
```

3 Bounding Practice

Given an array, the heapification operation permutes the elements of the array into a heap. There are many solutions to the heapification problem. One approach is bottom-up heapification, which treats the existing array as a heap and rearranges all nodes from the bottom up to satisfy the heap invariant. Another is top-down heapification, which starts with an empty heap and inserts all elements into it.

(a) Why can we say that any solution for heapification requires $\Omega(n)$ time?

In order to check that an array satisfies the heap invariant, we have to at least look at every element, which takes linear time.

(b) Give the worst-case runtime for top-down heapification in $\Theta(\cdot)$ notation. Why does this mean that the optimal solution for heapification takes $O(n \log n)$ time?

Worst-**case** runtime **for** top-down heapification is $\Theta(n \log n)$. This means that the optimal solution **for** heapification takes $O(n \log n)$ time since at least one solution **for** heapification takes $O(n \log n)$ time.

(c) Extra: Show that the running time of bottom-up heapify is $\Theta(n)$. Not extra: Is bottom-up heapification asymptotically optimal?

Some useful facts:

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}$$

Taking the derivative of both sides:

$$\sum_{i=0}^{\infty} ix^i = \frac{x}{(1-x)^2}$$

Running time of heapify is:

$$\begin{aligned}
\sum_{i=0}^{\log n} i \frac{n}{2^{i+1}} &= \frac{n}{2}\left(\sum_{i=0}^{\log n} i \left(\frac{1}{2}\right)^i\right) \\
&\leq \frac{n}{2}\left(\sum_{i=0}^{\infty} i \left(\frac{1}{2}\right)^i\right) \\
&= \frac{n}{2}\frac{\frac{1}{2}}{(\frac{1}{2})^2} \\
&= \Theta(n)
\end{aligned}$$

Since the running time of bottom-up heapify is $\Theta(n)$ and any solution for heapification requires $\Omega(n)$, bottom-up heapification is asymptotically optimal.