# Homework3 for EECS 340

Yu Mi,yxm319

February 26, 2018

## 1 Sorting Leftover Elements

*Question*: Suppose that you are given an array consisting of $n$ sorted elements followed by $f(n)$ elements in an arbitrary order, where $f(n) \in O(n^{1-\epsilon})$ for some $\epsilon \in (0, 1)$. Describe a method to sort the array in $O(n)$ time.
*Answer*: This question describes a scenario similar to the intermediate steps of a merge sort, so that we need to solve this problem similar to the approach of a merge sort. First, we need to sort the leftover elements with merge sort, which takes $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ time. After that, we will need to merge the two sequence (original sorted one and the left over part) into a whole sorted sequence, which takes $O(n)$ time. To show that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ takes less time than $O(n)$, we assign $g(n) = n^{1-\epsilon} \cdot \log n^{1-\epsilon}$, $h(n) = n$. Thus we have:

$$\lim_{n\to\infty} \frac{g(n)}{h(n)} = \lim_{n\to\infty} \frac{(1-\epsilon)\cdot n^{1-\epsilon}\cdot \ln n}{n \cdot \ln 2} = \lim_{n\to\infty} \frac{1-\epsilon}{\ln 2} \cdot (n^{-\epsilon} \cdot \ln n + n^{-\epsilon}) = \lim_{n\to\infty} \frac{1-\epsilon}{\ln 2} \cdot \frac{\ln n + 1}{n^{\epsilon}} = \lim_{n\to\infty} \frac{1-\epsilon}{\ln 2} \cdot \frac{1}{\epsilon n^{\epsilon}}$$

L'Hospital's rule is used in the second and forth step. The equation above will approach 0 when $n \to \infty$, so that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon})$ takes less time than $O(n)$, we can conclude that $O(n^{1-\epsilon} \cdot \log n^{1-\epsilon}) + O(n)$ is $O(n)$.

## 2 Theory: Sorting Algorithm Run-times

### 2.1 Give a tight asymptotic bound on $f(n)$

*Answer*: Since the total amount of permutation od $n$ elements is $n!$, and binary encoding will use $\log(n!)$ bits to encode such permutations, our tight asymptotic bound should be

$$\log(n!) = \sum_{i=1}^{n} \log i$$

Such time bound make sense because it is always smaller than $n \log n$, which is the time of fastest sorting algorithm (at least I know).

### 2.2 Why doesn't the derived lower bound, from the previous part hold for non-comparison-based sorting algorithms like radix sort.

*Answer*: Since non-comparison-based is not based on comparisons to make a sort, they cannot be viewed as a comparison-based approach where each comparison can be treated as searching for a bit in the permutation. The lower bound derived above only stands for the comparison based sorting algorithm where we use each comparison to compose a 'bit' for the ultimate answer.

## 3 Post Office Placement

*Question*: Is the bucket-sort algorithm in-place? Why or why not?
*Answer*: Bucket-sort is **not** an in-place algorithm. Since the number of buckets is only bounded by $n$, which is the length of the input sequence $S$, so that this algorithm will need linear additional space and consequently not in-place. In some worse implementations, the algorithm will allocate $O(n \cdot k)$ space where $k$ is the number of buckets.

## 4 Sorting Sequences

*Question*: Suppose we are given a sequence $S$ of $n$ elements, each of which is an integer in the range $[0, n^2 - 1]$. Describe a simple method for sorting $S$ in $O(n)$ time.
*Hint*: Think of alternate ways of viewing the elements. *Answer*: The approach is based on Radix sort:

**Algorithm** Radix-sort$(S, n)$
**Data**: A unsorted sequence $S$, and its length $n$
**Result**: The sorted sequence $S'$
$buckets \leftarrow n$ lists
**for** $i \leftarrow 0$ to $n$ **do**
  $digit \leftarrow$**int**$((S[n])/n)$
  $buckets[digit]$.append$(S[n])$
**end for**
**for** $i \leftarrow 0$ to $n$ **do**
  $result \leftarrow n$ lists
  **for** $j \leftarrow 0$ to $length(buckets[i])$ **do**
    $result[buckets[i][j]\%n].append(buckets[i][j])$
  **end for**
  $bucket \leftarrow$ empty list
  **for** $j \leftarrow 0$ to $n$ **do**
    $bucket.append(result[j])$
  **end for**
  $buckets[i] \leftarrow bucket$
**end for**
$result \leftarrow$ empty list
**for** $i \leftarrow 0$ to $n$ **do**
  **for** $number$ in $bucket[i]$ **do**
    $result.append(number)$
  **end for**
**end for**

## 5 Median From Two Lists

*Question*: Suppose you are given two sorted lists, $A$ and $B$, of $n$ elements each all of which are distinct. Describe a method that runs in $O(\log n)$ time for finding the median in the set defined by the union of $A$ and $B$. Note that merging or concatenating the arrays would take $O(n)$ time.

# 6 Warm-up: Graphs

## 6.1 R-7.1

*Question*: Suppose we have a social network with members $A, B, C, D, E, F$, and $G$, and the set of friendship ties,

$$\{(A, B), (B, C), (C, A), (D, E), (F, G)\}.$$

Where are the connected components?

## 6.2 R-13.2

*Question*: Let $G$ be a simple connected graph with $n$ vertices and $m$ edges. Explain why $O(\log m)$ is $O(\log n)$.

## 6.3 Given the graph

### 6.3.1 Draw the graph where all nodes are annotated with the order that they are visited in a depth-first search from A.

### 6.3.2 Draw a representation of the tree generated by the DFS annotated over the previous part.

# 7 Application: Chess

# 8 Application: A World of Voxels

# 9 Practice: Topological Sorting

# 10 EECS 454 only

## 10.1 C-17.4

*Question*: Consider the problem **DNF-SAT**, which takes a Boolean formula $S$ in disjunctive normal form (DNF) as input and asks whether $S$ is satisfiable. Describe a deterministic polynomial-time algorithm for **DNF-SAT**.

## 10.2 C-17.5

*Question*: Consider the problem **DNF-DISSAT**, which takes a Boolean formula S in disjunctive normal form (**DNF**) as input and asks whether S is dissatisfiable, that is, there is an assignment of Boolean values to the variables of S so that it evaluates to 0. Show that **DNF-DISSAT** is $NP$-complete.