# CSCI E-124 Data Structures and Algorithms — Spring 2015

## Problem Set 4

Due: 11:59pm, Wednesday, March 11th

See homework submission instructions at
`http://sites.fas.harvard.edu/~cs124/e124/problem_sets.html`

**Problem 5 is worth 40% of this problem set, and problems 1-4 constitute the remaining 60%.**

## 1 Problem 1

There are $n$ young wizards standing in a row at Hogwart's, sorted alphabetically by last name. The sorting hat would like to assign a house to each one: Hufflepuff, Gryffindor, Ravenclaw, or Slytherin. The hat's job is complicated by the fact that some children's parents absolutely hate certain houses, and if the hat assigned their child to such a house then the parents would complain to the headmaster, who in turn would burn the sorting hat to ashes. Furthermore the sorting hat does not like to put two alphabetically adjacent students into the same house. Given the sorted order of students, and constraints on certain students not being assigned to certain houses, is an assignment even possible?

Consider the following divide-and-conquer algorithm to answer the above question, where $S[i]$ is student $i$, and houses$[i]$ is the list of houses student $i$ may be assigned to:

```
algorithm is_possible(houses):
  n = length of houses

  if n == 0:
    return true

  possibilities = houses[middle]

  # split array (excluding middle element)
  left = houses[0...middle-1]
  right = houses[middle+1...n-1]

  if length of possibilities >= 3:
    return is_possible(left) and is_possible(right)
  else:
    for each possibility in possibilities:
      # try assigning middle student to house ‘possibility’
      a_left = ‘left’ with ‘possibility’ removed from houses[middle-1], if present
      a_right = 'right' with 'possibility' removed from houses[middle+1], if present
      if is_possible(a_left) and is_possible(a_right)
        return true

  return false
```

What's the running time of the above algorithm?

## 2 Problem 2

You are given an $n$-digit positive integer $x$ written in base-2. Give an efficient algorithm to return its representation in base-10 and analyze its running time. Assume you have black-box access to an integer multiplication algorithm which can multiply two $n$-digit numbers in time $M(n)$ for some $M(n)$ which is $\Omega(n)$ and $O(n^2)$.

## 3 Problem 3

Recall the following text search problem from class. There is an alphabet $\Sigma$ and two strings $P \in \Sigma^m$, $T \in \Sigma^n$, $n \geq m$ (i.e. strings of length $m$, $n$, respectively, made up of characters in $\Sigma$). We would like to output a list of all indices $i$ such that $T[i, i+1, \ldots, i+m-1] = P$, i.e. $t_{i+j-1} = p_j$ for $j = 1, \ldots, m$. In class, when $\Sigma = \{0, 1\}$ we showed how to use FFT to solve this problem in $O(n \log n)$ time (assuming a computer supporting infinite precision complex arithmetic). Making the same precision assumptions:

(a) Improve this time to $O(n \log m)$.

(a) Deal with the case when $P$ (but not $T$) has "don't care" symbols *. A * symbol can match either a 0 or a 1.

(a) Show how to deal with the case $|\Sigma| > 2$, without don't care symbols. You can assume $\Sigma = \{1, 2, \ldots, |\Sigma|\}$.

(a) Give a solution for $|\Sigma| > 2$ **with** don't care symbols, which can match any character in $\Sigma = \{1, 2, \ldots, |\Sigma|\}$. An $O(n \log m)$ solution (which is possible) naturally implies you don't have to separately do parts (a) through (c).

## 4 Problem 4

We construct an infinite sequence of arrays $A_1$, $A_2$, $A_3$, ... in the following recursive fashion. First, we specify that $A_1 = [1]$. For $k > 1$, we recursively define $A_k$ to be two copies of $A_{k-1}$ put together, with the number $k$ inserted between the two lists.

To illustrate the above algorithm:

$$A_1 = [1]$$

$$A_2 = [1, 2, 1]$$

$$A_3 = [1, 2, 1, 3, 1, 2, 1]$$

$$A_4 = [1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1]$$

and so on.

Devise an efficient algorithm that accepts some $k$ and two intervals $[a, b)$ and $[c, d)$, and determines the length of the longest subarray contained in both the arrays $A_k[a : b]$ and $A_k[c : d]$. An array $C$ is a subarray of an array $B$ if there exist $i$ and $j$ such that $C = B[i : j]$. As always, you must prove the algorithm's correctness and analyze its runtime.

## 5 Problem 5

Solve "Problem A - Zoo" on the programming server; see the "Problem Sets" part of the course web page for the link.