CS 344 Assignment 2

Craig Perkins, Alex Tang, Steve Grzenda

Due March 14, 2014

Problem 1

A. $n^{.51} = \Omega(n^{\log_4 2}) \implies O(n^{.51})$

B. $n! = \Omega(n^{\log_4 16}) \implies O(n!)$

C. $\log n = O(n^{\log_2 \sqrt{2}}) \implies O(n^{\log_2 \sqrt{2}})$ or $O(n^{\frac{1}{2}})$

D. Since at each step the size of the input is $n - 1$ this means that the algorithm is performed n times. Since each step take $\lg n$ time the total running time is $O(n \lg n)$

E. $\frac{n}{\lg n} = O(n^{\log_5 5}) \implies O(n^{\log_5 5})$ or $O(n)$

Problem 2

A.

B. Assume there are n programmers and n project managers needing to be matched. On the first iteration of the algorithm the n programmers will be assigned a meeting to a random project manager. If matched the programmer and project manager will be taken out of contention. If a match did not occur then the programmer will then hold a meeting with another project manager. The project manager will be determined based on the outcome of the first meeting, so if the first meeting did not match because the programmer was not skilled enough then he will meet with a project manager who met with a qualified programmer that wanted too much compensation and vice versa if the programmer wanted too much compensation.

Problem 3

1. To sort each university using a linear time comparison algorithm we can use either radix sort or bucket sort if the results are fairly uniform. Bucket sort would be appropriate because we know that there could be exactly 60 bins (if no partial credit is given). As for a non-comparative sorting algorithm counting sort could be appropriate because the elements to be sorted will be in the range from 0 to 600.

2. The algorithm to merge all of the sorted files will start by merging pairwise files. The algorithm starts by merging 2 sorted lists. The merging works by comparing the heads of both respective lists and and putting the smaller of the 2 elements as the head of the new list. The merging of the lists will occur in O(n) time. The algorithm will do this list sorting pairwise and work up in a binary tree sort of fashion. Since there are k files and n elements in total, the total running time of the algorithm will be $O(n \log k)$.

3.

Problem 4

```
function getMissingDate():
  boolean puzzles[n] // puzzles is a n-bit array
  for i in [1..n]:
    puzzles[i] = 0

  for i in [1..n]:
    crossword_puzzle puzzle = getPuzzle(i)
    day = getDay(puzzle)
    month = getMonth(puzzle)
    year = getYear(puzzle)
    puzzles[hash(month, day, year)] = 1

  for i in [1..n]:
    if puzzles[i] == 0:
      day = getDay(puzzle)
      month = getMonth(puzzle)
      year = getYear(puzzle)
      return month, day, year
```

This algorithm will run in $O(n)$ time and take n-bits of space, i.e. $O(n)$ space. Note that `hash()` can be any function that maps a 1-to-1 correspondence between the dates and the indices in the boolean array `puzzles`.

Problem 5

To achieve an approximation algorithm that runs in time $O(m \log m + n)$ we used a greedy approach. To start the algorithm will sort the people using quicksort from least to most greedy. The algorithm then distributes the gifts to each person starting with the person with lowest satisfaction level. If a person is not satisfied with a gift, the algorithm will repeatedly give gifts until the person is satisfied.

To find the approximation ratio for this greedy approach first assume that k people are satisfied in the optimal allocation of the gifts.