Assignment 2

Paul Jones and Matthew Klein
Professor Professor Kostas Bekris
Design and Analysis of Computer Algorithms (01.198.344)

March 5, 2014

Divide-and-Conquer Algorithms, Sorting Algorithms, Greedy Algorithms

Part A (20 points)

Problem 1

The more general version of the Master Theorem is the following. Given a recurrence of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where $a \geq 1$ and $b > 1$ are constants and $f(n)$ is an asymptotically positive function, there are 3 cases:

1. If $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{log_b a})$.
2. If $f(n) = \Theta(n^{log_b a} log^k n)$ with $k \geq 0$, then $T(n) = \Theta(n^{log_b a} log^{k+1} n)$. In most cases, $k = 0$.
3. If $f(n) = \Omega(n^{log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$. The regularity condition specifies that $af(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Give asymptotic bounds for the following recurrences. Assume $T(n)$ is constant for $n = 1$. Make your bounds as tight as possible, and justify your answers.

A. $T(n) = 2T(\frac{n}{4}) + n^{0.51}$

- $a = 2$, $b = 4$
- $f(n) = \Omega(n^c)$ where $c = 0.51$
- $\log_b a = \log_4 2 = \frac{1}{2}$
- $c > \log_b a$, that is $0.51 > \frac{1}{2}$
- $\frac{n^{0.51}}{2} \leq cn^{0.51}$ for any constant $c < 1$ and $c > 0.5$
- $T(n) = \Theta(f(n)) = \Theta(n^{0.51})$

B. $T(n) = 16T(\frac{n}{4}) + n!$

- $a = 16$, $b = 4$
- $f(n) = n!$

C. $T(n) = \sqrt{2}T(\frac{n}{2}) + \log n$

- $a = \sqrt{2}$, $b = 2$
- $f(n) = \log n$

D. $T(n) = T(n-1) + \log n$

- $a = 1$, $b = 1$
- $f(n) = \log n$

E. $T(n) = 5T(\frac{n}{5}) + \frac{n}{\lg n}$

- $a = 5$, $b = 5$
- $f(n) = \log n$

Part B (25 points)

Problem 2

You are in the HR department of a technology firm, and here is a job for you. There are n different projects, and n different programmers.

Every project has its unique payoff when completed and level of difficulty (which are uniform, regardless which programmer will work on the project). Every programmer has a unique skill set as well as expectations for compensation (which are uniform, regardless the project the programmer will work on). You cannot directly collect information that allows you to compare the payoff or difficulty level of two projects, or the capability or expectations for compensation of two programmers.

Instead, you can arrange a meeting between each project manager and programmer. In each meeting, the project manager will give the programmer an interview to see whether the programmer can do the project; the programmer can ask the project manager about the compensation to see whether it meets her expectations. After the meeting, you can get a result based on the feedback of the project manager and the programmer. The result can be:

1. The programmer can't do the project.
2. The programmer can do the project, but the compensation of the project doesn't meet her expectation.
3. The programmer can do the project, and the compensation for the project matches her expectations. At this time, we say the project and the programmer *match* with each other.

Assume that the projects and programmers match one to one. Your goal is to match each programmer to a project.

A. Show that any algorithm for this problem must need $\Omega(n \log n)$ meetings in the worst case.

B. Design a randomized algorithm for this problem that runs in expected time $O(n \log n)$.

Part C (40 points)

Problem 3

A nation-wide programming contest is held at k universities in North America. The i^{th} university has m_i participants. The total number of participants is n, i.e., $n = \sum_{i=1}^{k} m_i$. In the contest, participants have to write programs to solve 6 problems. Each problem contains 10 test cases, each test case is worth 10 points. Participants aim to maximize their collected points.

After the contest, each university sorts the scores of participants belonging to it and submits the grades to the organizer. Then the organizer has to collect the sorted scores of participants and provide a final sorted list for all participants.

1. For each university, how do they sort the scores of participants belonging to it? Please briefly describe a comparative sorting algorithm that is appropriate for this purpose and a non-comparative sorting algorithm that works in this setup.

2. How does the organizer sort the scores of participants given k files, where each file includes the sorted scores of participants from a specific university? Please describe an algorithm with a $O(n \log k)$ running time and justify its time complexity.

3. Suppose the organizer wants to figure out the participants of ranking r among all participants. Given the k sorted files, how does the organizer find the r^{th} largest scores without sorting the scores of all participants? Please describe an algorithm with $O(k(\sum_{i=1}^{k} \log m_i))$ running time and justify its time complexity.

   Can you do this in $O(\log k(\sum_{i=1}^{k} \log m_i))$ time?

   [Hint: If r is larger than $\frac{n}{2}$, the elements that have at least $\frac{n}{2}$ elements larger than them should not be considered. The problem is how to identify these elements within each sorted file.]

Problem 4

You have a collection of n New York Times crossword puzzles from 01/01/1943 until 12/31/2012 stored in a database. The only operations that you can perform to the database are the following:

- crossword_puzzle $x \leftarrow$ getPuzzle(int index); where the index is between 1 and n; the puzzles are *not* sorted in the database in terms of the date they appeared.
- getDay(crossword_puzzle x); which returns a number between 1 to 31.
- getMonth(crossword_puzzle x); which returns a number between 1 to 12.
- getYear(crossword_puzzle x); which returns a number between 1943 to 2012.

All of the above queries can be performed in constant time. You have found out that the number of puzzles is less than the number of days in the above period (from 01/01/1943 until 12/31/2012) by one, i.e., one crossword puzzle was not included in the database. We need to identify the date of the missing crossword puzzle.

Design a linear-time algorithm that minimizes the amount of space that it is using to find the missing date. Ignore the effect of leap years.

Part D (25 points)

Problem 5

You are running a promotional event for a company during which the plan is to distribute n gifts to the participants. Consider that each gift i is worth an integer number of dollars a_i. There are m people participating in the event, where $m < n$. The j-th person is satisfied if he receives gifts that are worth at least s_j dollars each. The task is to satisfy as many people as possible given that you have a knowledge of the gift amounts a_i and the satisfaction requirements of each person s_j. Give an approximation algorithm for assigning rewards to people with a running time of $O(m \log m + n)$. What is the approximation ratio of your algorithm and why?