CIS507: Design & Analysis of Algorithms
Homework 2, Spring 2014

Submissions are due as a zip or rar file emailed to **mjha@masdar.ac.ae** and **bhayk@masdar.ac.ae**; CC: **kelbassioni@masdar.ac.ae** by **Saturday, March 29th, 23:59:59 UAE time**. Acceptance of late submissions and penalties for the same are at the sole discretion of the course faculty.

Code is to be written in **python**, completed and submitted individually. Copying of code will invite severe penalties. Source code is to be submitted in a file <**last-name**>**-cis507hw2.src** and analyses/explanations are to be submitted in a LaTeX-generated PDF file <**last-name**>**-cis507hw2.pdf** (add your answers to the file <last-name>-cis507hw2.tex provided, then compile it using pdflatex).

Please ensure that the source files are tested thoroughly before committing to submission - post-submission bug fix requests will be entertained, again, at the sole discretion of the course faculty. Also, please ensure that your source file encoding is cross-platform readable (does not break when compiled on Linux/Windows/etc).

There is a total of 13 points in this assignment; for a perfect score, you need 10 points.

Q1. Problem 14-1 in text book (page 354) Suppose we wish to keep tack of a *point of maximum overlap (PMO)* in a set of intervals, that is, a point with the largest number of intervals in the set that overlap it (think of this as the busiest time for a set of events represented by the intervals). We would like to design a data structure to maintain the intervals, supporting the operations INTERVAL-INSERT and FIND-PMO. We would like all operations to take expected $O(\log n)$ time on n inserted *random* intervals.

1. **(0.5 point).** Show that there will always be a PMO that is an endpoint of one of the intervals.

2. **(0.5 point).** Suppose that we use a BST, which field should be used as the key, and what additional information, if any, needs to be stored at each node to support the required operations?

3. **(0.5 point).** Show how the operations FIND-PMO can be implemented to take expected $O(\log n)$ time on a random set of intervals.

4. **(2 points).** Implement the operations INTERVAL-INSERT and FIND-PMO. For testing purposes, your program should take a list of pairs, each of the form "interval.low interval.high", one per line. Suppose that the number of inserted intervals is n. The program then should output n lines, line i contains a PMO and the number of comparisons taken to find it (separated by a space), after the ith insertion.

Q2. Problem 17-2 in text book (page 473) Binary search of a sorted array takes logarithmic search time, but the time to insert a new element is linear in the size of the array. We can improve the time for insertion by keeping several sorted arrays.

Specifically, suppose that we wish to support SEARCH and INSERT on set of n elements. Let $k = \lceil \log_2(n+1) \rceil$, and let the binary representation of n be $\langle n_{k-1}, n_{k-2}, \ldots, n_0 \rangle$. We keep k sorted arrays $A_0, A_1, \ldots, A_{k-1}$, where for $i = 0, 1, \ldots, k-1$ the length of array A_i is 2^i. Each array is either full or empty, depending on whether $n_i = 1$ or $n_i = 0$, respectively. The total number of elements held in all k arrays is therefore $\sum_{i=0}^{k-1} n_i 2^i = n$. Although each array is sorted, elements in different arrays bear no particular relationships to each other.

1. **(0.5 point)** Describe (write a pseudo code) how to perform the SEARCH operation for this data structure in $O(\log^2 n)$-worst case running time.

2. **(1 point)** Describe how to perform the INSERT operation, and show it has $\Theta(n)$ worst case running time. *Hint:* Create a new array A'_0 of size 1 containing the new element to be inserted. If array A_0 (which has size 1) is empty, then replace A_0 with A'_0. Otherwise, merge the two arrays A_0 and A'_0 into another *sorted* array A'_1 of size 2, and make A_0 empty. If A_1 is empty, then replace A_1 with A'_1; otherwise, merge A_1 and A'_1 into a sorted array A'_2 of size 4, and so on.

3. Show that the INSERT operation has $O(\log n)$-amortized running time:

   (I) **(0.5 point)** use the aggregate method;

   (II) **(0.5 point)** use the accounting method;

   (III) **(1 point)** use the potential function method.

4. **(2 points)** Implement the data structure. For testing purposes, your program should accept as an input a file "test.in" containing a set of numbers (positive integers, 1 per line). It should search for each number in the data structure, and only inserts it if it is not found. The program outputs in another file "test.out", the following lines (in one line separated by a space): the number of successful searches, the amortized number of comparisons per successful search, the amortized number of comparisons per unsuccessful search, the amortized number of comparisons per insertion.

Q3. In a dynamic and limited memory environment, each running application is assigned a weight (or priority) and requires a certain amount of memory. An application might spawn other applications, but continues to function without all of them necessarily running (e.g., imagine opening a new tab in a browser). In case memory becomes scarce, the operating system has to choose a *minimum-weight* collection of applications to terminate such that the remaining ones fit into the available memory. When an application is terminated all the applications it generated are also terminated (and this applies recursively).

1. **(0.5 point)** Write a formal description of the problem.

2. **(2 points)** Design a dynamic program that decides which collection of applications to terminate. What are the time and space requirements of your DP?

3. **(1.5 points)** Implement the DP. For testing purposes, your program should accept as an input a file "test.in" containing in the first line the number of applications and the total avialable memory, followed by the set of applications (1 per line in the form "App-ID memory requirement weight <set of ID's of other applications-spawned by this application, separated by spaces>"). It should output in another file "test.out", the total weight of the minimum-weight set S of applications that has to be terminated, in one line, followed by the list of App-ID's for the applications in S (separated by spaces).