CSE548, AMS542: Analysis of Algorithms, Fall 2012 **Date:** October 16

In-Class Midterm

(11:35 AM – 12:50 PM : 75 Minutes)

- This exam will account for either 15% or 30% of your overall grade depending on your relative performance in the midterm and the final. The higher of the two scores (midterm and final) will be worth 30% of your grade, and the lower one 15%.
- There are three (3) questions, worth 75 points in total. Please answer all of them in the spaces provided.
- There are 12 pages including four (4) blank pages. Please use the blank pages if you need additional space for your answers.
- The exam is *open slides*. So you can consult the lecture slides during the exam. No additional cheatsheets are allowed.

Good Luck!

Question	Score	Maximum
1. Pairing Numbers		20
2. Tower of Hanoi		25
3. Binomial Arrays		30
Total		75

Name: _____

Question 1. [20 Points] Pairing Numbers. Suppose you are given two sets of numbers, say, A and B. For some integer $n > 0$, each set contains $\Theta(n)$ integers between 0 and n (inclusive), and no number occurs more than once in the set. Let $\mathcal{C}(m)$ denote the number of distinct pairs $\langle a, b \rangle$ with $a \in A$ and $b \in B$ such that $a + b = m$. For example, if $A = \{1, 7, 3\}$ and $B = \{2, 1, 6\}$ then $\mathcal{C}(9) = 2$ as only the pairs $\langle 7, 2 \rangle$ and $\langle 3, 6 \rangle$ produce the sum 9, and $\mathcal{C}(6) = 0$ as no two $a \in A$ and $b \in B$ sum up to 6.

1(a) **[5 Points]** For any given integer $m \in [0, 2n]$, show that the computation of $\mathcal{C}(m)$ requires $\Theta(n)$ time.

1(b) [**15 Points**] Show that for $0 \leq m \leq 2n$, one can compute all $\mathcal{C}(m)$ values simultaneously in $\mathcal{O}(n \log n)$ time.

Use this page if you need additional space for your answers.

Question 2. [**25 Points**] **Tower of Hanoi.** The following recurrences arise while trying to compute the minimum number of moves needed to solve the *Tower of Hanoi* problem with n discs when all moves must be clockwise[1]. Let Q_n and R_n be the minimum number of moves required to transfer all n discs under the given restriction from the source pole to the target pole and from the target pole to the source pole, respectively. Then it can be shown that

$$Q_n = \begin{cases} 0 & \text{if } n = 0, \\ 2R_{n-1} + 1 & \text{otherwise;} \end{cases} \qquad R_n = \begin{cases} 0 & \text{if } n = 0, \\ Q_n + Q_{n-1} + 1 & \text{otherwise.} \end{cases}$$

$2(a)$ [**5 Points**] Show that one can rewrite the first recurrence as follows.

$$Q_n = \begin{cases} 0 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ 2\left(Q_{n-1} + Q_{n-2}\right) + 3 & \text{otherwise.} \end{cases}$$

[1] If A is the source pole, B is the target pole, and C is the intermediate pole, then each move must be either $A \to B$ or $B \to C$ or $C \to A$.

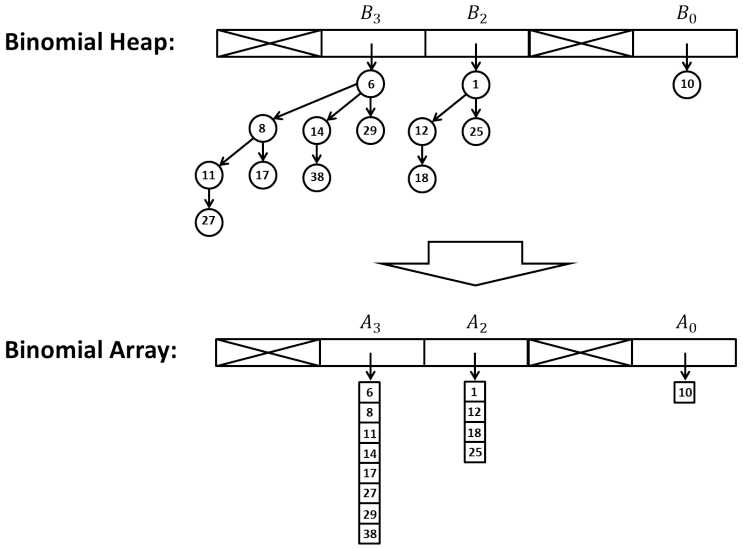
2(b) [**20 Points**] Solve the recurrences to show that

$$Q_n = \frac{1}{2\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+1} - \left(1-\sqrt{3}\right)^{n+1}\right] - 1 \text{ and } R_n = \frac{1}{4\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+2} - \left(1-\sqrt{3}\right)^{n+2}\right] - 1.$$

Use this page if you need additional space for your answers.

Question 3. [**30 Points**] **Binomial Arrays.** Searching for a given number in an array of n sorted numbers takes $\mathcal{O}(\log n)$ time using binary search, but inserting a new number into this array can take $\Theta(n)$ time in the worst case. On the other hand, a new number can be inserted into an array of n unsorted numbers in $\Theta(1)$ time[2], but searching for a number in this unsorted array can take up to $\Theta(n)$ time. In this problem we will analyze a very simple data structure that can support both insert and search operations very efficiently[3].

We will call the new data structure a *binomial array* which is nothing but a simple modification of the *binomial heap* data structure we saw in the class. If we replace each binomial tree B_k of a binomial heap with an array A_k of size 2^k containing all items of B_k in sorted order, we get a binomial array (see Figure below). Note that though each individual array is sorted there is no particular relationship between elements of two different arrays. While a Link operation on a binomial heap links two binomial trees of the same size (say, two B_k's) in $\Theta(1)$ time, the same operation on a binomial array merges two sorted arrays [4] of the same size, say two A_k's, in $\Theta\left(2^{k+1}\right)$ time into another sorted array A_{k+1} of size 2^{k+1}.



A binomial array supports only the following two operations:

(i) Insert(x) that inserts x into the data structure, and

(ii) Search(x) that returns True if x appears in the data structure, and False otherwise.

An Insert operation is implemented in exactly the same way as in a binomial heap except that each Link operation now merges two arrays instead of linking two binomial trees.

We start with an empty data structure, and perform Insert and Search operations on it in arbitrary order.

[2] assuming the array has at least one empty space that can be located in $\Theta(1)$ time

[3] not as efficiently as *balanced binary search trees* though, but they are very complicated

[4] two sorted arrays of size n_1 and n_2 can be merged into a sorted array of size $n_1 + n_2$ in $\Theta(n_1 + n_2)$ time

3(a) [**5 Points**] Show that an INSERT operation on a binomial array containing n numbers can take $\Theta(n)$ time in the worst case.

3(b) [**5 Points**] Show how to perform a SEARCH operation on a binomial array containing n numbers in $\Theta\left(\log^2 n\right)$ worst-case time.

3(c) [**10 Points**] Define a potential function Φ in order to analyze the amortized complexity of INSERT and SEARCH operations assuming that we insert n numbers into the data structure. Remember to show that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all $i > 0$, where D_i is the state of the data structure after the i^{th} operation.

3(d) [**10 Points**] Use your potential function from part 3(c) to argue that the amortized time complexity of INSERT is $\mathcal{O}(\log n)$ and that of SEARCH is $\mathcal{O}\left(\log^2 n\right)$.

Use this page if you need additional space for your answers.

Use this page if you need additional space for your answers.