In-Class Midterm (Solution Ideas)

(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.
- 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.

Solution. We will first prove an $\mathcal{O}(n)$ upper bound on the time required to compute $\mathcal{C}(m)$. We construct two arrays $\mathcal{A}[0:n]$ and $\mathcal{B}[0:n]$, in $\Theta(n)$ time each, where for each $i \in [0, n]$,

$$\mathcal{A}[i] = \begin{cases} 1 & \text{if } i \in A, \\ 0 & \text{otherwise.} \end{cases} \qquad \mathcal{B}[i] = \begin{cases} 1 & \text{if } i \in B, \\ 0 & \text{otherwise.} \end{cases}$$

Then clearly, for any given $m \in [0, 2n]$, the following code fragment computes $\mathcal{C}(m)$ in c.

1. $c \leftarrow 0$
2. ***for*** $i \leftarrow \max\{\ 0,\ m - n\ \}$ ***to*** $\min\{\ m,\ n\ \}$ ***do***
3. ***if*** $\mathcal{A}[i] = 1$ ***and*** $\mathcal{B}[m - i] = 1$ ***then*** $c \leftarrow c + 1$

Observe that the ***for*** loop above iterates $\mathcal{O}(n)$ times with each iteration taking $\Theta(1)$ time. Hence, $\mathcal{C}(m)$ can be computed in $\mathcal{O}(n)$ time.

Now in order to prove a lower bound observe that the entries in the sets A and B are not sorted. Hence, for $m \geq n$, you must check each entry of A and B in order to make sure that you have not missed a pair that sum up to m. This give us the $\Omega(n)$ lower bound on the time required to compute $\mathcal{C}(m)$.

Thus 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.

Solution. We construct two polynomials $\mathcal{P}_{\mathcal{A}}(x) = \sum_{0 \leq i \leq n} a_i x^i$ and $\mathcal{P}_{\mathcal{B}}(x) = \sum_{0 \leq j \leq n} b_j x^j$ of degree n each in $\Theta(n)$ time, where, for $i, j \in [0, n]$

$$a_i = \begin{cases} 1 & \text{if } i \in A, \\ 0 & \text{otherwise.} \end{cases} \qquad b_j = \begin{cases} 1 & \text{if } j \in B, \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathcal{P}_C(x) = \sum_{0 \leq m \leq 2n} c_m x^m$ be the product of these two polynomials, i.e., $\mathcal{P}_C(x) = \mathcal{P}_A(x)\mathcal{P}_B(x)$. Then clearly, $c_m = \sum_{0 \leq i, m-i \leq n} a_i b_{m-i}$ which is nothing but $\mathcal{C}(m)$.

We know that $\mathcal{P}_C(x)$ can be computed in $\mathcal{O}(n \log n)$ time using FFT. Hence, all coefficients of $\mathcal{P}_C(x)$, and thus $\mathcal{C}(m)(= c_m)$ for all $m \in [0, 2n]$ can be computed in $\mathcal{O}(n \log n)$ time.

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.

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

Solution.

Given: $Q_0 = 0$.

Also: $Q_1 = 2R_0 + 1 = 2 \times 0 + 1 = 1$.

Finally, for $n > 1$, $Q_n = 2R_{n-1} + 1 = 2\left(Q_{n-1} + Q_{n-2} + 1\right) + 1 = 2\left(Q_{n-1} + Q_{n-2}\right) + 3$.

[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

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

Solution. Let's look at several ways of solving the given recurrences.

(*i*) **By Solving a Linear Homogeneous Recurrence with Constant Coefficients.**

Let $P_n = Q_n + 1$ for all n. Then from part 2(*a*):

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

Characteristic equation: $r^2 - 2r - 2 = 0 \Rightarrow r = 1 \pm \sqrt{3}$.

Then for some constants α_1 and α_2: $P_n = \alpha_1\left(1+\sqrt{3}\right)^n + \alpha_2\left(1-\sqrt{3}\right)^n$.

From initial conditions: $\alpha_1 + \alpha_2 = P_0 = 1$ and $\alpha_1\left(1+\sqrt{3}\right) + \alpha_2\left(1-\sqrt{3}\right) = P_1 = 2 \Rightarrow \alpha_1 - \alpha_2 = \frac{1}{\sqrt{3}}$.

Solving for α_1 and α_2 we get: $\alpha_1 = \frac{1}{2\sqrt{3}}\left(1+\sqrt{3}\right)$ and $\alpha_2 = -\frac{1}{2\sqrt{3}}\left(1-\sqrt{3}\right)$.

Hence, $P_n = \alpha_1\left(1+\sqrt{3}\right)^n + \alpha_2\left(1-\sqrt{3}\right)^n = \frac{1}{2\sqrt{3}}\left(1+\sqrt{3}\right)^{n+1} - \frac{1}{2\sqrt{3}}\left(1-\sqrt{3}\right)^{n+1}$.

Then $Q_n = P_n - 1 = \frac{1}{2\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+1} - \left(1-\sqrt{3}\right)^{n+1}\right] - 1$.

Given that for $n > 0$, $Q_n = 2R_{n-1} + 1 \Rightarrow R_{n-1} = \frac{1}{2}\left(Q_n - 1\right)$, we have for $n \geq 0$, $R_n = \frac{1}{2}\left(Q_{n+1} - 1\right) = \frac{1}{4\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+2} - \left(1-\sqrt{3}\right)^{n+2}\right] - 1$.

(*ii*) **By Solving a Linear Nonhomogeneous Recurrence with Constant Coefficients.**

From part 2(*a*):

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

Associated homogeneous characteristic equation: $r^2 - 2r - 2 = 0 \Rightarrow r = 1 \pm \sqrt{3}$.

Then for some constants α_1 and α_2, homogeneous solution: $Q_n^{(h)} = \alpha_1\left(1+\sqrt{3}\right)^n + \alpha_2\left(1-\sqrt{3}\right)^n$.

Particular solution of nonhomogeneous recurrence: $Q_n^{(p)} = p_0$.

Then substituting in the given recurrence we get $p_0 = 2(p_0 + p_0) + 3 \Rightarrow p_0 = -1$.

Hence, $Q_n = Q_n^{(h)} + Q_n^{(p)} = \alpha_1\left(1+\sqrt{3}\right)^n + \alpha_2\left(1-\sqrt{3}\right)^n - 1$.

From initial conditions: $\alpha_1 + \alpha_2 - 1 = Q_0 = 0 \Rightarrow \alpha_1 + \alpha_2 = 1$ and $\alpha_1\left(1+\sqrt{3}\right) + \alpha_2\left(1-\sqrt{3}\right) - 1 = Q_1 = 1 \Rightarrow \alpha_1 - \alpha_2 = \frac{1}{\sqrt{3}}$.

Solving for α_1 and α_2 we get: $\alpha_1 = \frac{1}{2\sqrt{3}}\left(1+\sqrt{3}\right)$ and $\alpha_2 = -\frac{1}{2\sqrt{3}}\left(1-\sqrt{3}\right)$.

Hence, $Q_n = \alpha_1\left(1+\sqrt{3}\right)^n + \alpha_2\left(1-\sqrt{3}\right)^n - 1 = \frac{1}{2\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+1} - \left(1-\sqrt{3}\right)^{n+1}\right] - 1$.

Given that for $n > 0$, $Q_n = 2R_{n-1} + 1 \Rightarrow R_{n-1} = \frac{1}{2}\left(Q_n - 1\right)$, we have for $n \geq 0$, $R_n = \frac{1}{2}\left(Q_{n+1} - 1\right) = \frac{1}{4\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+2} - \left(1-\sqrt{3}\right)^{n+2}\right] - 1$.

(*iii*) **Using a Generating Function.**

We can write the recurrence for Q_n from part $2(a)$ as a single recurrence equation as follows.

$$Q_n = [n=1] + \left(2Q_{n-1} + 2Q_{n-2} + 3\right)[n>1].$$

Now let the generating function for Q_n be: $Q(z) = \sum_{n=0}^{\infty} Q_n z^n$.

Then

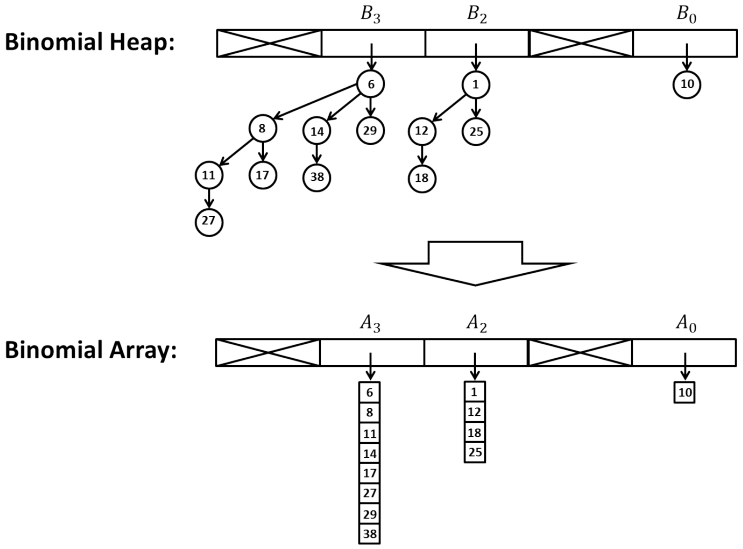
$$\begin{array}{rrcl}
 & Q(z) & = & \sum_{n=0}^{\infty}\left([n=1] + \left(2Q_{n-1} + 2Q_{n-2} + 3\right)[n>1]\right)z^n \\
 & & = & z + 2\sum_{n=2}^{\infty} Q_{n-1}z^n + 2\sum_{n=2}^{\infty} Q_{n-2}z^n + +3\sum_{n=2}^{\infty} z^n \\
 & & = & z + 2z\sum_{n=0}^{\infty} Q_n z^n + 2z^2\sum_{n=0}^{\infty} Q_n z^n + +3\sum_{n=0}^{\infty} z^n - 3 - 3z \\
 & & = & 2zQ(z) + 2z^2Q(z) + \frac{3}{1-z} - 3 - 2z \\
\Rightarrow & (1-2z-2z^2)Q(z) & = & \frac{z+2z^2}{1-z} \\
\Rightarrow & Q(z) & = & \frac{z+2z^2}{(1-2z-2z^2)(1-z)} \\
 & & = & \frac{1}{1-2z-2z^2} - \frac{1}{1-z} \\
 & & = & \frac{1}{(1-(1+\sqrt{3})z)(1-(1-\sqrt{3})z)} - \frac{1}{1-z} \\
 & & = & \frac{1}{2\sqrt{3}z}\left(\frac{1}{1-(1+\sqrt{3})z} - \frac{1}{1-(1-\sqrt{3})z}\right) - \frac{1}{1-z} \\
 & & = & \frac{1}{2\sqrt{3}}\left(\frac{1}{z}\cdot\left(1-\left(1+\sqrt{3}\right)z\right)^{-1} - \frac{1}{z}\cdot\left(1-\left(1-\sqrt{3}\right)z\right)^{-1}\right) - (1-z)^{-1} \\
 & & = & \frac{1}{2\sqrt{3}}\left(\frac{1}{z}\sum_{n=0}^{\infty}\left(1+\sqrt{3}\right)^n z^n - \frac{1}{z}\sum_{n=0}^{\infty}\left(1-\sqrt{3}\right)^n z^n\right) - \sum_{n=0}^{\infty} z^n \\
 & & = & \frac{1}{2\sqrt{3}}\left(\sum_{n=0}^{\infty}\left(1+\sqrt{3}\right)^{n+1} z^n - \sum_{n=0}^{\infty}\left(1-\sqrt{3}\right)^{n+1} z^n\right) - \sum_{n=0}^{\infty} z^n
\end{array}$$

Now equating the coefficients of z^n from both sides, $Q_n = \frac{1}{2\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+1} - \left(1-\sqrt{3}\right)^{n+1}\right] - 1$.

Given that for $n > 0$, $Q_n = 2R_{n-1} + 1 \Rightarrow R_{n-1} = \frac{1}{2}\left(Q_n - 1\right)$, we have for $n \geq 0$, $R_n = \frac{1}{2}\left(Q_{n+1} - 1\right) = \frac{1}{4\sqrt{3}}\left[\left(1+\sqrt{3}\right)^{n+2} - \left(1-\sqrt{3}\right)^{n+2}\right] - 1$.

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.

Solution. Consider inserting into a binomial array $\mathcal{A}$ containing $n = 2^{h+1} - 1$ numbers for some integer $h > 0$. Then for $0 \leq k \leq h$, the k^{th} location of $\mathcal{A}$ will contain an A_k with 2^k numbers before insertion. The initial creation of a binomial array containing only the new number will cost $\Theta(1)$. Then the insertion into $\mathcal{A}$ will complete in k stages of merge operations. At stage 0, the new number will be merged with the A_0 at location 0 producing an A_1 which will then be merged with the A_1 at location 1 at stage 1, and so on. In general, at stage $k \in [0, h]$ an A_k will be merged with the A_k at location k of $\mathcal{A}$ to produce an A_{k+1} which will be carried to stage k + 1. The total cost of all these merges is $\sum k = 0^h 2^{k+1} = 2(2^{h+1} - 1) = 2n = \Theta(n)$. Hence, INSERT will require $\Theta(n)$ time in this 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.

Solution. Consider the case when $n = 2^{h+1} - 1$ for some integer $h > 0$, and the item to searched is not in the data structure. For $0 \leq k \leq h$, the k^{th} location of $\mathcal{A}$ will contain an A_k with 2^k numbers, and we will have to search each of those A_k's. If we use binary search, total cost of search will be $\sum_{k=0}^{h}(k+1) = \frac{1}{2}(h+1)(h+2) = \Theta\left(h^2\right) = \Theta\left(\log^2 n\right)$.

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.

Solution. We will use the following potential function:

$$\Phi(D_i) = c \sum_{A_k \in D_i} (h - k) \cdot 2^k,$$

where $h = \lceil \log_2 (n + 1) \rceil$, and c is a constant. Observe that $A_k \in D_i \Rightarrow k < h \Rightarrow h - k > 0$.

Initially, D_i contains no numbers, and so it contains no A_k's, and consequently, $\Phi(D_i) = 0$.

Afterwards D_i contains at least one A_k, and since $h - k$ is always positive, we have $\Phi(D_i) > 0$.

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 $\mathcal{O}\left(\log^2 n\right)$.

Solution. Let's analyze the amortized cost of INSERT first.

The actual cost of creating the initial binomial array containing the new number is 1. Suppose INSERT scans $k > 0$ consecutive locations of D_{i-1} before it stops. At each location $j \in [0, k)$, it merges two A_j's costing 2^{j+1}. Total cost of these merges is $\sum_{j=0}^{k-1} 2^{j+1} = 2(2^k - 1)$.

Hence, actual cost of INSERT, $c_i = 1 + k + 2(2^k - 1) = 2^{k+1} + k - 1$.

Due to this INSERT operation all A_j's for $0 \leq j \leq k - 1$ vanish, and one new A_k emerges. Hence, potential drop, $\Delta_i = \Phi(D_i) - \Phi(D_{i-1}) = c\left((h - k) \cdot 2^k - \sum_{j=0}^{k-1} (h - j) \cdot 2^j\right) = c\left(-2^{k+1} + h + 2\right)$.

Hence, amortized cost, $\hat{c}_i = c_i + \Delta_i = (1 - c)2^{k+1} + (c \cdot h + k) + 2c$. Choosing $c = 1$, we get $\hat{c}_i = h + k + 2 = \mathcal{O}(\log n)$.

Now let us consider SEARCH. Actual cost, $c_i = \mathcal{O}\left(\log^2 n\right)$ (from part 3(b)), and potential drop, $\Delta_i = 0$. Hence, amortized cost, $\hat{c}_i = c_i + \Delta_i = \mathcal{O}\left(\log^2 n\right)$.