# CS4311 Design and Analysis of Algorithms

Homework 3

Due: 11:10 am, May 8, 2008 (before class)

1. (20%) There is a staircase with $n$ steps. Your friend, Jack, wants to count how many different ways he can walk up this staircase. Because John is quite tall, in one move, he can choose either to walk up 1 step, 2 steps, or 3 steps.

   Let $F_k$ denote the number of ways John can walk up a staircase with $k$ steps. So, $F_1 = 1$, $F_2 = 2$, $F_3 = 4$, and $F_4 = 7$.

   Derive a recurrence for $F_k$, and show that $F_n$ can be computed in $O(n)$ time.

2. Consider the following coin-taking game between Tom and Jerry. We start with two piles of coins, with $x$ coins in the first pile and $y$ coins in the second pile. They will take turn to remove coins from the pile. In each turn, they have three options: (i) remove any number of coins from the first pile, (ii) remove any number of coins from the second pile, or (iii) remove the *same* number of coins from both piles.

   In this game, the person loses if he cannot get any coin during his turn. For instance, suppose it is Tom's turn, and the coins in the current piles are $(x, y) = (5, 5)$. In this case, if Tom now removes coins using the third option, he can make the piles $(0, 0)$ so that Jerry loses by not getting any coin in his next turn.

   If both Tom and Jerry plays cleverly, some combinations of $(x, y)$ are always losing. For instance, $(0, 0)$ is a losing combination, and we can also show that $(1, 2)$ is a losing combination as follows:

   - Firstly, there are only 4 ways of removing the coins: (1) take 1 coin from pile 1; (2) take 1 coin from pile 2; (3) take 2 coins from pile 2; (4) take 1 coin from both piles;
   - If we remove using (1), the remaining pile is $(0, 2)$, so that the opponent player can force us to lose by taking 2 coins from pile 2.
   - If we remove using (2), the remaining pile is $(1, 1)$, so that the opponent player can force us to lose by taking 1 coin from both piles.
   - If we remove using (3), the remaining pile is $(1, 0)$, so that the opponent player can force us to lose by taking 1 coin from pile 1.
   - If we remove using (4), the remaining pile is $(0, 1)$, so that the opponent player can force us to lose by taking 1 coin from pile 2.

   Suppose $x$ and $y$ are both at most $n$. For any $i, j$, let $A(i, j) = L$ if $(i, j)$ is a losing combination, and $A(i, j) = W$ otherwise.

   (20%) Derive a recurrence for $A(i, j)$ and show that we can determine if $(x, y)$ is a losing combination in $O(n^3)$ time.

3. (20%) John wants to drive from San Francisco to Seattle using his car. His car's gas tank, when full, holds enough gas to travel exactly $n$ km. John has already made up his route, and his only concern now is to plan where to refill his gas tank along the way.

Suppose that there are $k$ gas stations $x_1, x_2, \ldots, x_k$ along the route, and the distance between any two of them are known. Also, for any two consecutive gas stations, the distance is at most $n$ km.[†]

To save time, John wants to stop for as few gas stations as possible. Give an efficient method to find out the gas stations for John to stop, and show that your method yields an optimal solution.

4. (20%) A min-heap with $n$ elements supports `Insert` and `Extract-Min` in $O(\log n)$ worst-case time. Give a potential function $\Phi$ such that the amortized cost of `Insert` is $O(\log n)$ and the amortized cost of `Extract-Min` is $O(1)$. Show that your $\Phi$ works.

5. A sorted array allows efficient searching in logarithmic time. However, if we want to insert a new element, it requires linear time in the worst case to keep the array sorted.

   In fact, we can improve the insertion time (in the amortized sense) by making the searching time a bit slower. Let $n$ be the number of elements in the current array. Let $k = \lceil \log(n+1) \rceil$, so that the binary representation of $n$ has $k$ bits.

   Our scheme is to partition the $n$ elements into $k$ arrays $A_0, A_1, \ldots, A_{k-1}$ based on $n$'s binary representation. Precisely, let $\langle b_{k-1}, b_{k-2}, \ldots, b_0 \rangle$ be the binary representation, with $b_{j-1}$ being the $j$th least-significant bit. The array $A_i$ will hold $2^i$ elements in *sorted* order if $b_i = 1$; otherwise, it will be *empty*.

   Based on this scheme, searching for an element will need to search all $k$ arrays in the worst case, and the total time will be $O(k \log n) = O(\log^2 n)$.

   To insert a new element, the number of elements, $n$, will be increased. Then, its binary representation will change, so we need to partition the elements in a different way.[¶]

   (20%) Describe how to insert a new element into this data structure so that the partitioning can be maintained correctly. Show that the amortized insertion time is $O(\log n)$.

6. (Bonus: 10%)[§] This question is an extension of Question 2. Our target is to show that we can determine if $(x, y)$ is a losing combination in $O(n)$ time. You may devise your own proof, or you may follow the steps suggested below.

   (a) Show that for each $k$, there is at most one $x$ such that $(x, x+k)$ is a losing combination.

   (b) Show that for each $x$, there is at most one $r$ such that $(x, r)$ is a losing combination.

   (c) We are going to describe a procedure to find losing combinations $L_0, L_1, \ldots$:

```
Set $L_0 = (0, 0)$;
**for** $k = 1, 2, \ldots,$ {
    Set $v$ = smallest positive integer not in $L_0, L_1, L_2, L_{k-1}$;
    Set $L_k = (v, v + k)$;
}
```

[†]Assume that the distance between San Francisco and $x_1$, and the distance between $x_k$ and Seattle, are both at most $n$ km. So, John must be able to arrive Seattle.

[¶]For example, when there are 5 elements, we will have 3 sorted arrays, such that one is holding 4 elements, one is holding 1 element, and one is empty. When we insert a new element, the number of elements becomes 6, and we want to have one sorted array with 4 elements, one with 2 elements, and one empty.

[§] Q6 is a bonus question. Total mark is calculated by: (Sum of Q1 to Q5) $\times$ (100%+ Q6).

E.g., the above procedure sets $L_1 = (1, 2)$, $L_2 = (3, 5)$, $L_3 = (4, 7)$, and $L_4 = (6, 10)$. Show that each $L_k$ from the above procedure is a losing combination. (Hint: Use induction and the results of (a) and (b).)

(d) Show that after $x$ iterations, there must be some $L_i$ containing $x$.

(e) Show how to implement the procedure in (c) so that the total running time for $x$ iterations is $O(n)$.

(f) Conclude that we can determine if $(x, y)$ is a losing combination in $O(n)$ time.