CMPUT 204 (A1)
Midterm 2: Version A
October 29, 2004

Department of Computing Science
University of Alberta

Instructor: M. Bowling

Time: 50 Minutes

Last Name:_____

First Name:_____

- Put your name above (no ID). Print clearly.
- There are 3 problems, on 3 pages (50 marks in total).
- Closed Book: Only Pen, Pencil & Eraser allowed.
- Use space below the questions to write your answers.

Do NOT write on this page, for instructor's use only.

total marks	question	your marks
15	1	
20	2	
15	3	
50		

Last Name:_____ **ID#:**_____

1. **(15 Marks)** (a) Consider a max-heap with n elements stored as an array. Describe a procedure that takes a max-heap and modifies it into another max-heap with the *smallest* element removed. You may make use of any of the standard max-heap or min-heap procedures (e.g., MAX-HEAPIFY, BUILD-MAX-HEAP, EXTRACT-MAX). Your procedure must be efficient, i.e., it must have the fastest possible asymptotic running time in terms of n.

Solution: The simplest solution is:

BUILD-MIN-HEAP(A)
EXTRACT-MIN(A)
BUILD-MAX-HEAP(A)

This has a running time in $\Theta(n)$, which is efficient since any algorithm must look at all of the at least $\lfloor \frac{n}{2} \rfloor$ leaves.

A solution with a better constant factor in running time is to find the minimum element in the range $A[\lfloor \frac{n}{2} \rfloor \ldots n]$, say it's $A[i]$. Swap $A[i]$ with $A[n]$ and reduce the heap size by 1. Now look at $A[i]$ and compare its value with its parent. If larger, swap, and repeat with the new parent until it is less than or equal to its parent or it is the root. This will take $\Theta(n)$ time to find the minimum and $\Theta(\log n)$ to make a max-heap again. In total, $\Theta(n)$.

(b) What is the asymptotic running time of *your* algorithm?

Solution: $\Theta(n)$

Last Name:_____ **ID#:**_____

---

2. **(20 Marks)** Using divide-and-conquer, write pseudocode that finds the median element in an unsorted list of n elements. Recall, the median element is the $\lfloor \frac{n}{2} \rfloor$th element in the array after it is sorted. Your algorithm must be efficient for the average case, i.e., it must have the fastest possible asymptotic running time in terms of n. You do *not* have to analyze the running time.

Hint: An efficient running time is in $o(n \log n)$, so any algorithm that sorts the array is not efficient.

Hint: Quicksort's PARTITION procedure might be helpful.

Solution:

MEDIAN(A, s, e)
  $m \leftarrow$ PARTITION(A, s, e)
  if $\left(m = \lfloor \frac{\text{length}(A)}{2} \rfloor\right)$ then return $A[m]$
  if $\left(m < \lfloor \frac{\text{length}(A)}{2} \rfloor\right)$ then return MEDIAN$(A, m+1, e)$
  if $\left(m > \lfloor \frac{\text{length}(A)}{2} \rfloor\right)$ then return MEDIAN$(A, s, m-1)$

The running time is much like quicksort, and the analysis can proceed in the same way to arrive at $T(n) \in \Theta(n)$. No analysis was required, though.

Last Name:_____ **ID#:**_____

3. **(15 Marks)** Dr. Strange has decided to invent his own sequence of numbers, which he calls the Multinacci sequence. The sequence is defined by the following recurrence.

$$M(n) = \begin{cases} 1 & \text{if } n = 1 \\ 1 & \text{if } n = 2 \\ M(n-1)M(n-2) + 1 & \text{otherwise} \end{cases}$$

Use dynamic programming to write the procedure MULT(n) in pseudocode to compute the nth Multinacci number. The procedure must run in $O(n)$ time.

Solution:

```
MULT(n)
    m[1] ← 1
    m[2] ← 1
    for i ← 3 to n do
        m[i] ← m[i − 1] ∗ m[i − 2] + 1
    return m[n]
```