CS 240: Data Structures and Data Management **Fall 2020**

Midterm Practice Problems

Reminder: Midterm begins on Thursday, October 29 2020, 5 pm.

Note: This is a sample of problems designed to help prepare for the midterm exam. These problems do *not* encompass the entire coverage of the exam, and should not be used as a reference for its content.

1 True/False

Indicate "True" or "False" for each of the statements below.

a) If $T_1(n) \in \Omega(f(n))$ and $T_2 \in O(g(n))$, then $\frac{T_1(n)}{T_2(n)} \in \Omega\left(\frac{f(n)}{g(n)}\right)$.

b) If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = e^{42}$, then $f(n) \in \Theta(g(n))$.

c) If $f(n) \in o(n \log n)$, then $f(n) \in O(n)$.

d) For MSD-RadixSort, we can use HeapSort as the digit-sorting algorithm instead of CountSort.

e) All heaps satisfy the AVL height-balance requirement.

f) A binary search tree with n leaves must have height in $O(n)$.

g) If at least one rotation was performed during AVL-delete, then the height of the AVL Tree after deletion is strictly less than the height of the AVL Tree before deletion.

h) A skip-list with n keys and height h must have a total of $O(nh)$ nodes.

i) For an array of integers representing a min-heap, we can use interpolation search to find a target key.

j) The height of a compressed trie storing n keys is always less or equal to the height of an AVL Tree storing the same keys.

2 Order Notation

a) Show that $3n^2 - 8n + 2 \in \Theta(n^2)$ from first principles.

b) Complete the statement $n! \in \sqcup(n^n)$ by filling in $\sqcup$ with either Θ, o, or ω, and prove the corresponding relationship.

c) Prove or disprove: if $f(n) \in \Theta(g(n))$, then $\log f(n) \in \Theta(\log g(n))$. Assume that $f(n)$ and $g(n)$ are positive functions. You should prove the statement from first principles or provide a counter example.

3 Runtime Cases

After midterm grading is complete, Frank wants to make sure all the grades are okay, and have them listed in sorted order. He assigns either Jason or Sajed to perform this task. Jason and Sajed have many sorting algorithms to choose from, so their decision is based on their mood. Their moods fluctuate while they check assignment grades, and eventually depends on the final student's grade.

Assignment grades in array A are integers ranging from 0 to 100. There is exactly one student with a grade of 0, and exactly one student with a grade of 100. The function `verify(A)` runs in $\Theta(n)$ time and returns:

- +1, if the final student of A scored 100.
- -1, if the final student of A scored 0.
- 0, otherwise.

```
JasonSort (A):
    mood := verify(A)
    if mood = +1
        HeapSort(A)
    else if mood = -1
        SelectionSort(A)
    else MSDRadixSort(A, R = 10, m = 3)
```

```
SajedSort (A):
    mood := verify(A)
    if mood = +1
        SelectionSort(A)
    else if mood = -1
        LSDRadixSort(A, R = 10, m = 3)
    else MergeSort(A)
```

a) What is the best-case, worst-case, and average-case runtime for `JasonSort`?

b) What is the best-case, worst-case, and average-case runtime for `SajedSort`?

c) Frank's decision, which we refer to as `FrankSort`, is to flip a coin. If it flips heads, then he assigns the task to Jason (`JasonSort`). Otherwise, if it flips tails, then he assigns the task to Sajed (`SajedSort`). What is the best-case expected runtime, worst-case expected runtime, and average-case expected runtime for `FrankSort`?

4 String Sorting

a) Suppose we have n strings (of varying lengths) from a fixed known alphabet Σ_1 of size $c_1 \geq 2$, where the total length of all strings is ℓ. Give an algorithm to sort the strings in $O(\ell)$ time in lexicographical ordering, e.g., "a" < "ab" < "b".

b) Suppose the strings now represent numbers from a fixed known alphabet Σ_2 of size $c_2 \geq 2$, and the total length is still ℓ. Give an algorithm to sort the numbers in $O(\ell)$ time by value, e.g. "8" < "12" < "13".

5 Pseudocode Runtime Analysis

Analyze the worst-case runtimes for the following pseudocodes as functions of n:

a)

```
$j \leftarrow 0$;
$k \leftarrow 1$;
while $j \leq n$ do
    $j \leftarrow j + k$;
    $k \leftarrow k + 2$;
end
```

b) For the following algorithm, you may assume that n is a power of 3.

```
Algorithm 1: STOOGE($A, i, j$)
Input: Array $A$ of size $n$, index $i$ (initially 0), index $j$ (initially $n - 1$)
Output: No output but the subarray $A[i \dots j]$ will be sorted
if $A[j] < A[i]$ then
    SWAP($A[i], A[j]$)
end
if $j - i + 1 > 2$ then
    $t \leftarrow \lfloor \frac{j-i+1}{3} \rfloor$;
    STOOGE($A, i, j - t$);
    STOOGE($A, i + t, j$);
    STOOGE($A, i, j - t$);
end
```

c)

```
$x \leftarrow n$;
while $x > 1$ do
    if $x$ is even then
        $x \leftarrow x/2$;
    end
    $x \leftarrow 3x + 1$;
end
```

6 *d*-ary Heaps

Suppose instead of binary heaps, we have d-ary heaps for $d \geq 2$, where each internal node contains d children, except possibly the last one.

a) What is the height of a d-ary heap of n nodes?

b) Suppose the d-ary heap is represented in an array similar to binary heaps. For a node at index i, give the indices of its parent and all of its children.

c) Give an efficient algorithm for `Insert` and analyze its runtime.

d) Give an efficient algorithm for `DeleteMax` and analyze its runtime.

7 Find Zero

Let $\mathcal{A}$ be a sorted array of n numbers (may include negative non-integers) such that the number 0 appears exactly k times. The elements of $\mathcal{A}$ cannot normally be accessed, except through a query function $CheckSign(\mathcal{A}, i)$ that returns either POSITIVE, NEGATIVE, or ZERO, based on the value of $\mathcal{A}[i]$. The objective is to utilize these queries to find any single index of $\mathcal{A}$ that carries a 0 value.

a) Give a precise (not order notation) lower bound on the number of *CheckSign* calls required to find a 0 value in the worst-case.

b) Design an algorithm $FindZero(\mathcal{A}, n, k)$ that returns an index containing a 0. For full credit, the algorithm must not access $\mathcal{A}$ outside of calls to *CheckSign*, and the number of calls to *CheckSign* should be of the same order as the lower bound in part (a), i.e., the number of calls should be in $\Theta(lowerbound)$, where *lowerbound* is the expression from part (a).

8 Updating Partial Sum

Consider the problem where we have a sequence of n elements: $S = a_1, a_2, ..., a_n$, and 3 operations:

- $Add(S, b) \rightarrow a_1, a_2, ..., a_n, b$
- $Update(S, i, \Delta) \rightarrow a_1, ..., a_{i-1}, \Delta, a_{i+1}, ..., a_n$
- $PartialSum(S, k) \rightarrow \sum_{i=1}^{k} a_i$

Design a data structure that can perform each of these operations in $O(\log n)$ expected time.

9 MTF Scenario Analysis

Consider a linked list of n items where we perform m searches using the Move-To-Front heuristic, where $m > n$. For each of the following scenarios, give $\Theta()$ bounds on the worst-case runtimes in terms of m and n.

a) 99% of the operations are on the same key x.

b) At most $\sqrt{n}$ different elements are searched.

c) After an element is searched, it is searched again in the next 100 queries, or it is never searched again.