Practice Midterm Solutions

Problem 1. Master Theorem

Here is a table of logarithms. In the row \(i\) and column \(j\) you find the value of \(\log_i j\).

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(\log_2 1 = 0)</td>
<td>(\log_2 2 = 1)</td>
<td>(\log_2 3 = 1.5849)</td>
<td>(\log_2 4 = 2)</td>
</tr>
<tr>
<td>3</td>
<td>(\log_3 1 = 0)</td>
<td>(\log_3 2 = .6309)</td>
<td>(\log_3 3 = 1)</td>
<td>(\log_3 4 = 1.2618)</td>
</tr>
<tr>
<td>4</td>
<td>(\log_4 1 = 0)</td>
<td>(\log_4 2 = .5)</td>
<td>(\log_4 3 = .7924)</td>
<td>(\log_4 4 = 1)</td>
</tr>
<tr>
<td>5</td>
<td>(\log_5 1 = 0)</td>
<td>(\log_5 2 = .4306)</td>
<td>(\log_5 3 = .6826)</td>
<td>(\log_5 4 = .8613)</td>
</tr>
</tbody>
</table>

Solve the following recursions (\(c\) is always a constant). Give only the final results.

(a) \(T(n) = T(n/3) + c\).
(b) \(T(n) = 4T(n/2) + cn^3\).
(c) \(T(n) = 4T(n/4) + cn\).
(d) \(T(n) = 3T(n/5) + c\sqrt{n}\).

Solution

(a) \(a = 1, b = 3, \text{ and } n^{\log_b a} = n^0 = 1\).
\[ f(n) = c. \]
Therefore, \(f(n) = \Theta(n^{\log_b a})\) and \(T(n) = \Theta(n^{\log_b a} \log n) = \Theta(\log n)\).

(b) \(a = 4, b = 2, \text{ and } n^{\log_b a} = n^2\).
\[ f(n) = cn^3. \]
Therefore, \(f(n) = \Omega(n^{\log_b a + 1})\) and \(T(n) = \Theta(f(n)) = \Theta(n^3)\).

(c) \(a = 4, b = 4, \text{ and } n^{\log_b a} = n^1 = n\).
\[ f(n) = cn. \]
Therefore, \(f(n) = \Theta(n^{\log_b a})\) and \(T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n \log n)\).

(d) \(a = 3, b = 5, \text{ and } n^{\log_b a} = n^{.6826}\).
\[ f(n) = c\sqrt{n} = cn^{.5}. \]
Therefore, \(f(n) = O(n^{\log_b a - .1826})\) and \(T(n) = \Theta(n^{.6826})\).
Problem 2. Divide and Conquer

You are given a (not necessarily sorted) array \( a_1, a_2, \ldots, a_n \) of \( n \) integers. You can assume that the numbers are all different, but you cannot assume that they come from a small range. Consider the collection \( C \) of \( n^2 \) numbers of the form: \( \min\{a_i, a_j\} \), for \( 1 \leq i, j \leq n \). Present an \( O(n) \)-time algorithm to find the median of \( C \) (note that the elements of \( C \) are not all different, even if \( a_1, \ldots, a_n \) were all different).

Solution

The basic difficulty of the problem is in establishing which element of the original array can be the median of the collection \( C \).

One can observe that if \( a_k \) is the minimum element of the given \( n \) integers it has to appear \( 2n - 1 \) times in the collection \( C \) (this is because \( \min\{a_k, a_j\} = a_k \forall j = 1 \ldots n \) and \( \min\{a_j, a_k\} = a_k \forall j = 1 \ldots n \). The -1 comes from the fact that in both the above equations the comparison \( \min\{a_k, a_k\} \) is performed). By a similar argument the “second” minimum has to be repeated \( 2(n - 1) - 1 \) times. As a matter of fact the total number of element, in \( C \), is given by

\[
\sum_{i=0}^{n-1} 2(n - i) - 1 = n^2
\]

Now all we have to establish is the value \( k \) such that

1. \( \sum_{i=0}^{k-1} 2(n - i) - 1 \leq n^2/2 \)
2. \( \sum_{i=0}^{k} 2(n - i) - 1 \geq n^2/2 \)

The algorithm that solves the problem is thus (where \( A \) is supposed to be the original array)

**Find Median\((A, n)\)**

\[
\begin{align*}
  k &= 0; \\
  position &= 0; \\
  \text{WHILE ( position }&\leq n^2/2) \text{ DO} \\
  \text{ position} &= \sum_{i=0}^{k} 2(n - i) - 1 \\
  k &= + \\
\text{END WHILE} \\
  m &= \text{Select}(A, k + 1, n) \\
\text{return } m
\end{align*}
\]

**Correctness**

The above argument justifies the correctness of the algorithm

**Analysis**

Note that the sum \( \sum_{i=0}^{k} 2(n - i) - 1 \), can be calculated in time \( O(c) \) where \( c \) is a constant, being it rewritable as:

\[
\sum_{i=0}^{k} 2(n - i) - 1 = (k + 1)(2n - 1) - k(k + 1)
\]
Thus the while loop takes at most $O(n)$ time. The procedure of $\text{Select}(A, k + 1, n)$ has also running time $O(n)$. So the total complexity of the algorithm $\text{Find Median}$ is $O(n)$.

With some more analytical work, we can avoid doing the search for the right value of $k$, and we can rather compute it directly. If $v$ is a value that is the $k$-th order statistics in \(a_1, \ldots, a_n\), then we can verify that $C$ contains \((2kn - 2n - k^2 + 2k - 1)\) elements that are smaller than $v$, \(2n - 2k + 1\) elements that are equal to $v$, and \(n^2 - kn + k^2\) elements that are bigger than $v$. If $k = \lfloor n(1 - 1/\sqrt{2}) \rfloor$ then $v$ is the median of $C$. A more efficient solution is then just to invoke $\text{Select}(A, \lfloor n(1 - 1/\sqrt{2}) \rfloor, n)$.

**Problem 3. A data structure problem**

Consider the problem of implementing a variation of a stack where there is also a $\text{min}$ operation. Formally, you have to implement the operations:

- $\text{CreateStack()}$ that returns an empty stack;
- $\text{Push}(x, S)$ that puts the integer $x$ in the stack $S$;
- $\text{Pop}(S)$ that returns the most-recently-pushed element of $S$ and deletes it from $S$;
- $\text{Min}(S)$ that returns the value of the minimum element of $S$ (but does not remove it from the stack).

Describe and analyse an implementation of such a data structure such that all operations take worst-case $O(1)$ time. You cannot assume that the integers are in a small range (in particular, it is impossible to implement in constant time a $\text{DeleteMin}$ operation, since otherwise you could sort in linear time).

**Solution**

The basic idea is to implement the stack as a linked list. Each item of the list simply contains an element of the stack and a pointer to the local minimum (i.e. the minimum of the elements up to that moment inserted).

Clearly $\text{CreateStack}(), \text{Push}(x, S), \text{Pop}(S)$ and $\text{Min}(S)$ require $O(1)$ time.