Practice Midterm

The midterm will be a closed book exam. You can only use one two-sided sheet of notes. No book, no other notes, no calculator, no collaboration.

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

Problem 2. Median

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).

[Hint: if a value $v$ is the $k$-th order statistics in $a_1, \ldots, a_n$, how many elements are smaller than $v$ in $C$, and how many elements are bigger than $v$ in $C$?]

Problem 3. A Data Structure Problem

Consider the problem of implementing a variation of a stack where there is also a $\min$ operation.

Formally, you have to implement the operations:
• **CreateStack()** that returns an empty stack;
• **Push(x,S)** that puts the integer x in the stack S;
• **Pop(S)** that returns the most-recently-pushed element of S and deletes it from S;
• **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 `DeleteMin` operation, since otherwise you could sort in linear time).

(Hint: implement the stack as a linked list. Each item in the list contains an element of the stack and some additional information.)