CMPS 101: Winter 2016: HW 2

Due: 29th January 2016

- The assignment is to be attempted in groups of two.
- Each group needs to submit only one set of solutions.
- \LaTeX{} is preferred, but neatly handwritten solutions will also be accepted. All solutions need to be handed over in the class before the beginning of the lectures.
- The names of the group members, and their UCSC ID (@ucsc.edu email address) should prominently be written on the upper left corner of the first page.
- Multiple sheets should be stapled together in the upper left corner.
- Solutions to the problems should be clearly labeled with the problem number.
- Although no points are given for neatness, illegible and/or poorly organized solutions can be penalized at the graders option.
- Clearly acknowledge sources, and mention if you discussed the problems with other students or groups. In all cases, the course policy on collaboration applies, and you should refrain from getting direct answers from anybody or any source. If in doubt, please ask the instructors or TAs.

**Question 1 (1.5 points):** Solve the following recurrences and state the time complexity in $O$ or $\Theta$ notation:

\[
\begin{align*}
T(n) &= 9T(n/3) + n^2 \\
T(n) &= 9T(n/3) + n \\
T(n) &= 9T(n/3) + 1
\end{align*}
\]

You may choose any of the three methods we discussed in the class for solving the recurrence.

**Question 2 (2 point):** Consider the following pseudocode:

```python
def find_max(A, begin, end):
    if end - begin == 2:
        return max(A[begin], A[end-1])
```

1
\[ \begin{align*}
&\text{if end - begin} = 1: \\
&\quad \text{return } A[\text{begin}] \\
&\; \text{mid} = \text{begin} + (\text{end} - \text{begin})/2 \\
&\quad \text{return } \max(\text{find}\_\text{max}(A, \text{begin}, \text{mid}+1), \text{find}\_\text{max}(A, \text{mid}, \text{end}))
\end{align*} \]

which finds the maximum element in an array recursively. Analyze the runtime complexity of the above algorithm when it is given as input an array of size \( n \).

**Question 3 (1.5 points)** Given an array \( A \) with \( n \) elements, we want to find the \( k \)-th smallest element of \( A \). Consider the following observation: If we choose a random element \( a \in A \), we can split \( A \) into three parts \( A_\lt \), \( A_= \), and \( A_\gt \). Here \( A_\lt \) (resp. \( A_\gt \)) contains all elements of \( A \) which are smaller (resp. larger) than \( a \), while \( A_= \) contains all occurrences of \( a \) in \( A \). If the size of \( A_\lt \) denoted by \( |A_\lt| \geq k \), then, to solve our original problem, it suffices to find the \( k \) smallest element in \( A_\lt \). On the other hand, if \( |A_\lt| < k \), but \( |A_\lt|+|A_=| \geq k \), then \( a \) is the \( k \)-th smallest element. Finally, if \( |A_\lt|+|A_=| = k' < k \), then we need to find the \( k-k' \)-th smallest element in \( A_\gt \).

1. Write the pseudo-code for a recursive algorithm for finding the \( k \)-th smallest element of \( A \), based on the above idea.
2. Analyze the complexity of your algorithm, and provide \( O(\cdot) \), \( \Omega(\cdot) \), and, if applicable, \( \Theta(\cdot) \) bounds on the runtime.

**Question 4 (1 point)**: Given a min-heap \( H \) which contains \( n \) numbers and a query \( q \), write pseudocode for an algorithm which will check if the heap contains at least \( k \) elements which are smaller than \( q \). The time complexity of your algorithm should grow as \( O(k) \), independent of \( n \).