# Advanced Analysis of Algorithms - Homework II

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

## 1 Instructions

1. The homework is due on October 3, in class.
2. Each question is worth 4 points.
3. Attempt as many problems as you can. You will be given partial credit, as per the policy discussed in class.
4. The work must be entirely your own. You are expressly **prohibited** from consulting with colleagues or the internet (with the exception of the material on the course website, your class notes and [NN09]).

## 2 Problems

1. Solve the following recurrences:
   (a) $T(n) = T(\sqrt{n}) + 1$.
   (b) $T(n) = T(\frac{n}{3}) + T(\frac{2 \cdot n}{3}) + \Theta(n)$.

   In both cases, you may assume some constant value for $T(1)$.
2. Design a formal specification for the MERGE procedure discussed in class. Argue the correctness of your procedure using loop-invariants and analyze its running time.
3. Professor Krustowski claims to have discovered a new sorting algorithm. Given an array $\mathbf{A}$ of $n$ numbers, his algorithm breaks the array into 3 equal parts of size $\frac{n}{3}$, viz., the first third, the middle third and the bottom third. It then recursively sorts the first two-thirds of the array, the bottom two-thirds of the array and finally the first two-thirds of the array again. Using mathematical induction, prove that the Professor has indeed discovered a correct sorting algorithm. You may assume the following: The input size $n$ is a power of 3. Additionally, the algorithm sorts by brute-force, when $n$ is exactly 3. Formulate a recurrence relation to describe the complexity of Professor Krustowski's algorithm and obtain tight asymptotic bounds.
4. Design a divide-and-conquer algorithm to find the second smallest element in an unsorted array $\mathbf{A}[1 \cdot \cdot n]$. Your algorithm should make at most $(n + \lceil \log n \rceil)$ element to element comparisons.
5. Devise a Divide-and-Conquer procedure for computing the $k^{th}$ smallest element in an array of $n$ integers. Analyze the asymptotic time complexity of your algorithm. *(Hint: Use the Partition procedure discussed in class.)* Your algorithm can be deterministic or randomized. For maximum points, your algorithm should run in $O(n)$ time.

# References

[NN09] Richard Neapolitan and Kumarss Naimipour. *Foundations of Algorithms Using C++ Pseudocode*. Jones and Bartlett, 2009.