**Advanced Algorithms**

**Exercise for Lecture 2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Student ID** |  |
| **Problem 1** |  | | |
| **Problem 2** |  | | |
| **Problem 3** |  | | |
| **Problem 4** |  | | |
| **Problem 5** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2023-09-10 24:00**  Submission Format: ‘**Lecture2\_Name\_Student ID.docx**’, and please send to: **[algorithms\_23fall@163.com](mailto:algorithms_23fall@163.com)**.  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Problem 1. [20 points]** Let , where , be a degree-*d* polynomial in *n*, and let *k* be a constant. Use the definitions of the asymptotic notations to prove the following properties.

1. If , then .
2. If , then .
3. If , then .
4. If , then .
5. If , then .

**Solution:**

**Problem 2. [20 points]** Derive solutions to the following recurrences. A solution should include the tightest upper and lower bounds that the recurrence will allow. Assume .

Solve parts **a**, **b**, and **c** in **two ways**: drawing a recursion tree and applying Master Theorem. Solve part **d** **only by substitution**.

1. .
2. .
3. .
4. .

**Solution:**

**Problem 3. [20 points]** Suppose you are choosing between the following three algorithms:

* Algorithm *A* solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.
* Algorithm *B* solves problems of size by recursively solving two subproblems of size and then combining the solutions in constant time.
* Algorithm *C* solves problems of size by dividing them into nine subproblems of size , recursively solves each subproblem, and then combining the solutions in quadratic time.

What are the running times of each of these algorithms, and which would you choose.

**Solution:**

**Problem 4. [20 points]** Give asymptotic upper and lower bounds for the recurrence . Make your bounds as tight as possible.

**Solution:**

**Problem 5. [20 points]**

1. Suppose you have sorted arrays, each with elements, and you want to combine them into a single sorted array of elements. One strategy is to merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm in terms of and ? Propose a more efficient solution to this problem, using divide-and-conquer.
2. You are given an array of elements, and you notice that some of the elements are duplicates; that is, they appear more than once in the array. Show how to remove all duplicates from the array by modifying the procedure *Merge* in *MergeSort*.

**Solution:**