**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:**

1. If while , is true, then .

Obviously, the inequality on the left is true.

Divide both sides of the inequality on the right by :

.

While , , the inequality on the right is true, so we have proved that . [4 points]

1. If while , is true, then .

Divide both sides of the inequality on the right by : .

While , , the inequality on the right is true, so we have proved that . [4 points]

1. If while is true, then .

Divide both sides of the inequality on the right by :

.

While , , , the inequality on both sides is true, so we have proved that . [4 points]

1. If while is true, then .

Obviously, the inequality on the left is true.

Divide both sides of the inequality on the right by :

.

While is large enough, for , , the inequality on the right is true, so we have proved that . [4 points]

1. If while is true, then .

Divide both sides of the inequality on the right by :

.

While is large enough, for , , the inequality on the right is true, so we have proved that . [4 points]

**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. .

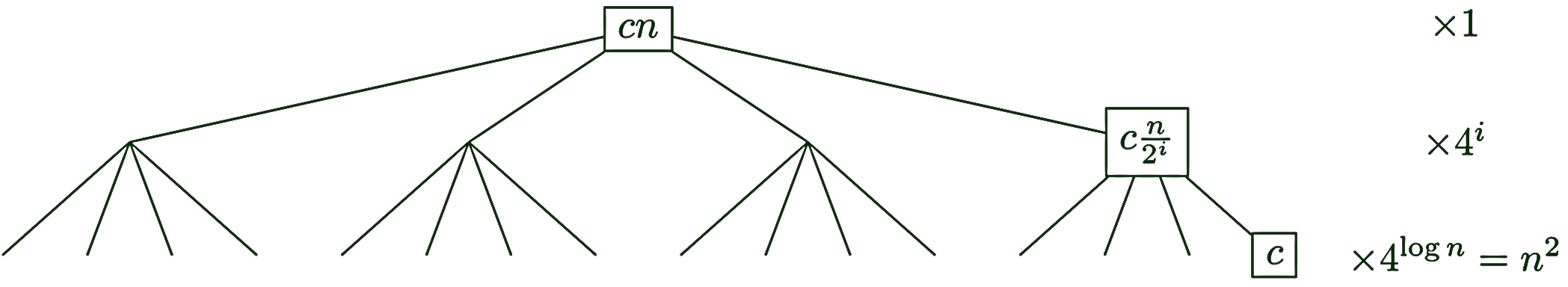
**Solution:**

Master Theorem (from CLRS 4th Ed.):

**文本

描述已自动生成**

1. (1) Solution to this recurrence using the Recursion Tree Method:



There are vertices at depth each doing at most work, so the total work at depth is at most . Summing over the entire tree, the total work is at most .

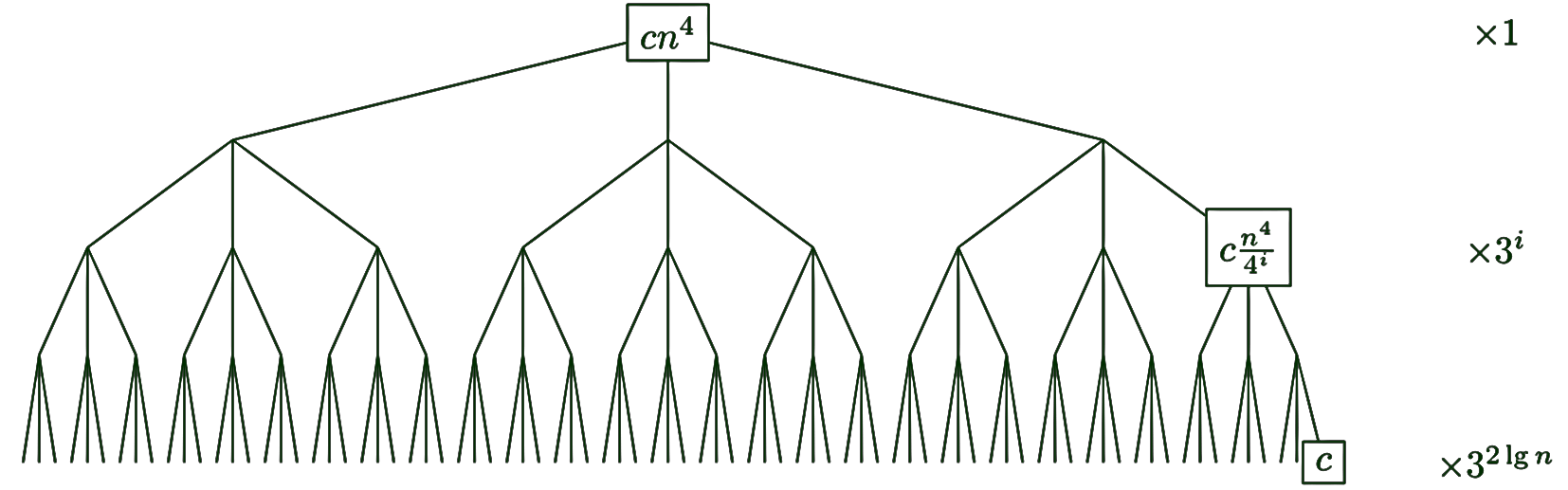
Since work is done at each leaf, and there are leaves, the total work is at least leading to running time.

(2) Solution to this recurrence using the Master Theorem:

by case 1 of the Master Theorem, since and for any positive . Note that this is true no matter the choice of .

1. (1) Solution to this recurrence using the Recursion Tree Method:

There are vertices at depth , each doing at most work, so the total work at depth is at most . Summing over the entire tree, the total work is at most , so . Alternatively, there are leaves in the tree, each doing work, so is at least .



(2) Solution to this recurrence using the Master Theorem:

We can upper bound by choosing . Then by case 3 of the Master Theorem, since and for any positive , and for any .

Alternatively, we can lower bound by choosing . Then by case 1 of the Master Theorem, since and for any positive .

1. (1) Solution to this recurrence using the Recursion Tree Method:

图片包含 多边形

描述已自动生成

There are vertices at depth each doing at most work, so the total work at depth is . In other words, the total work on level from the bottom is . Summing over the entire tree, the total work is .

(2) Solution to this recurrence using the Master Theorem:

by case 2 of the Master Theorem, since and .

1. Guess

So .

Parts a-c [18 points]

2 points for analysis via Master Theorem

2 points for drawing of tree

2 points for analysis based on tree

Part d [2 points]

2 points for correct analysis based on substitution

**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.

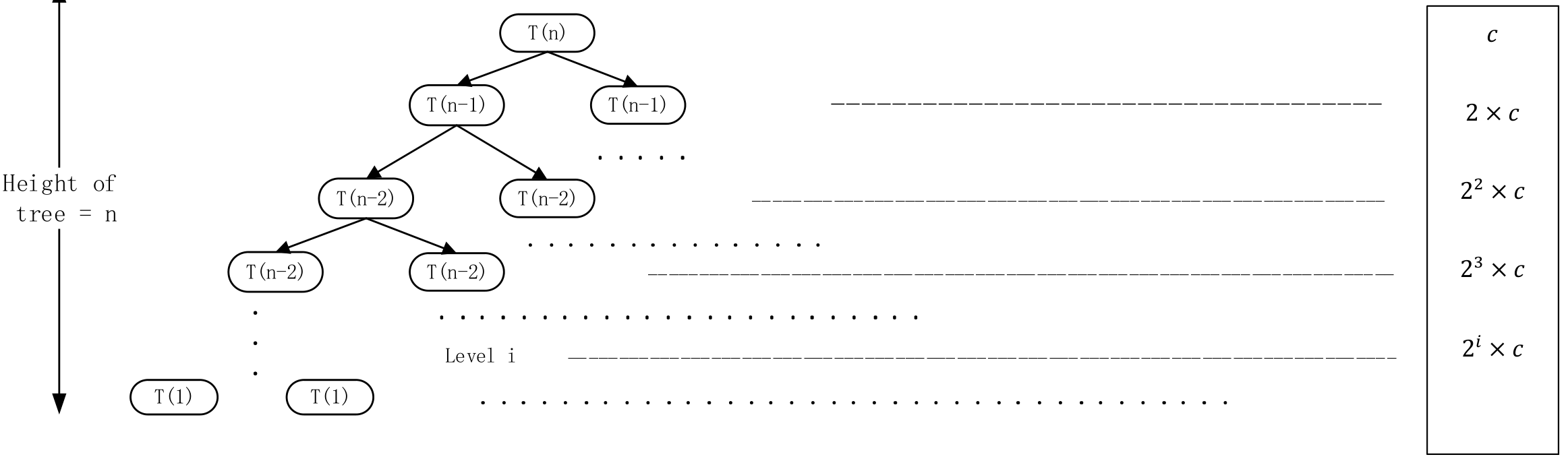
What are the running times of each of these algorithms, and which would you choose.

**Solution:**

Algorithm *A* has the recurrence .

by case 1 of the Master Theorem, since and , where . [5 points]

Algorithm *B* has the recurrence . [5 points]



. [3 points]

Algorithm *C* has the recurrence .

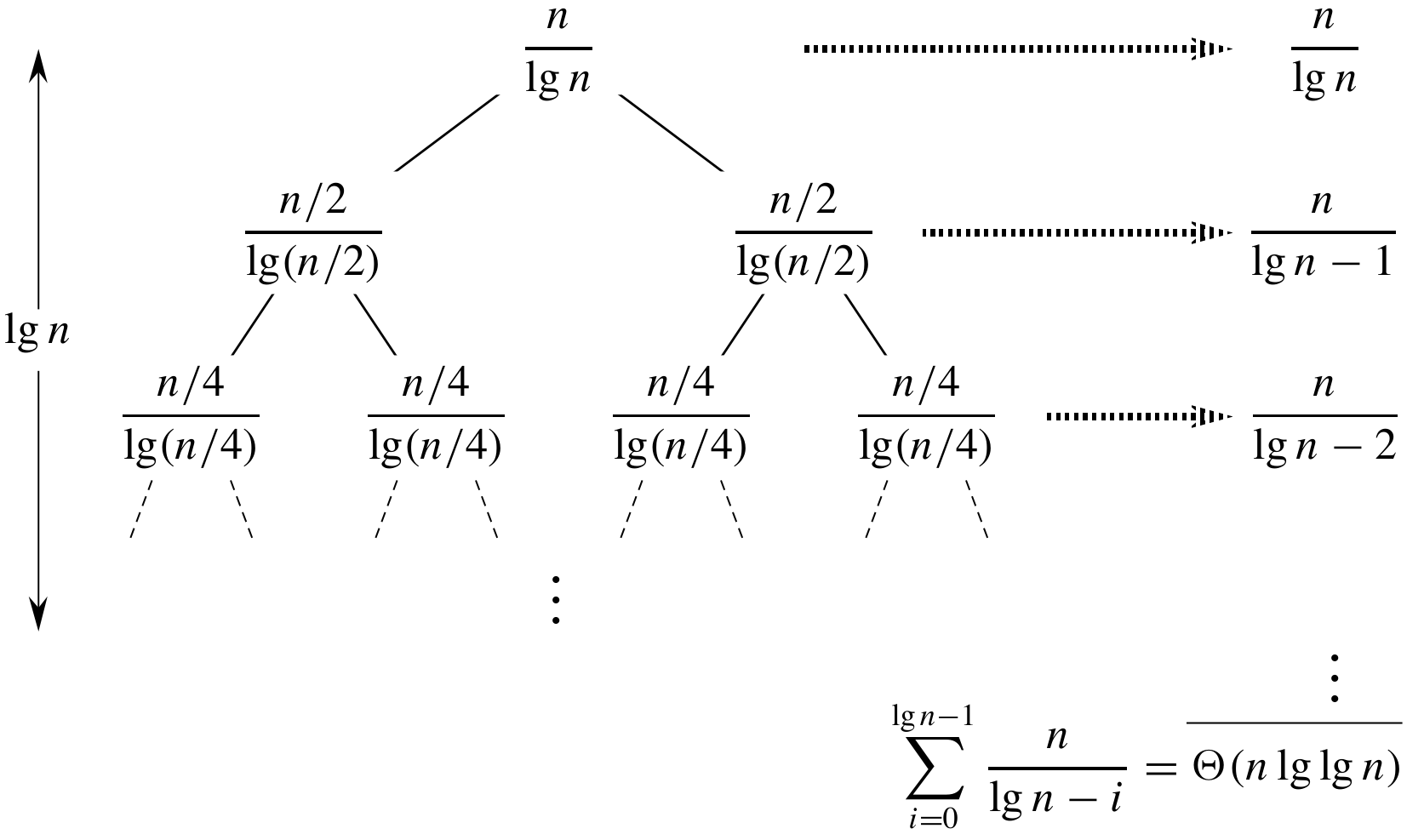
by case 2 of the Master Theorem, since and . [5 points]

Clearly, as , algorithm *C* is the fastest. [2 points]

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

**Solution:**

We get the sum on each level by observing that at depth , we have nodes, each with a numerator of and a denominator of , so that the cost at depth is . The sum for all levels is .



12 points for drawing of tree

8 points for analysis based on tree

**Problem 5. [20 points]**

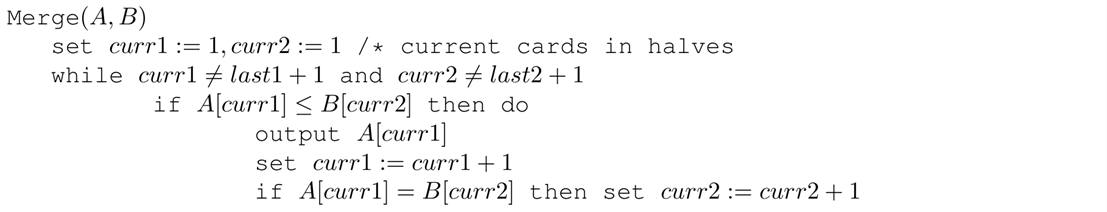
1. Suppose you have sorted arrays, each with elements, and you want to combine them into a single sorted array of elements. One strategy is to merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm in terms of and ? Propose 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*.

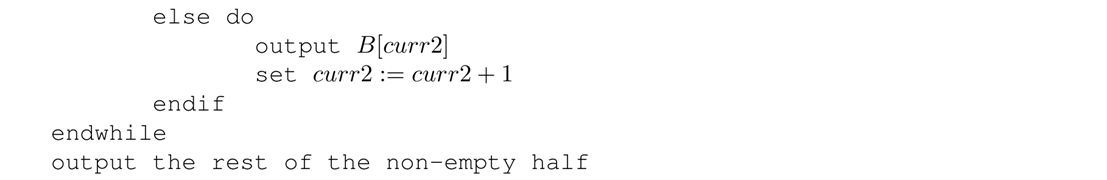
**Solution:**

1. Merging two sorted arrays with and elements, respectively, takes time. This strategy begins by merging two arrays of size to create an array of size . It then merges that with an array of size 𝑛, and so on. Thus, the running time is [6 points]

We can improve the running time by observing that the input is exactly the same as we see in the last level of the *MergeSort* recursion tree (assuming the input array breaks up evenly into halves). Therefore, if we merge pairwise, we get the same running time as for *MergeSort*, which is for an input array of size ; Since the total number of elements in the given problem is , we get . [7 points]

1. Modify the procedure *Merge* in *MergeSort*: when comparing two elements on the top of two halves, discard one of them if they are equal. More precisely, the modified *Merge* is as follows (here A, B are the two parts of the array being merged):





Note that this procedure recursively maintains the property: if A, B have no duplicates, then the combined array also has no duplicates. [6 points]

The entire process is to sort the array using modified *MergeSort* in time. [1 points]