HW3 — Divide & Conquer

CMPSC 465 — Summer 2025

Instructor: AMS

Deadline 1: 5th July 2025 at 11:59pm

Deadline 2 (Final): 7th July 2025 at 11:59pm

Total Points: 100

General instructions. Deadline 1: Submit either a full or partial attempt with time spent, or an explanation and time spent if you didn't attempt. Informal formats (paper, iPad, LaTeX) are allowed. Deadline 2: Type your answers using LaTeX and upload the PDF on Canvas by the deadline. There will be a point deduction (5% of the total homework points) if not written in latex. For more details and instructions read the syllabus.

Describing an Algorithm: Please make sure you use plain wording to explain your algorithm. It is always a good practice to start with a summary of the high-level idea of your algorithm to ease graders understand your solution quickly. Then, explain your algorithm, using plain wording and including enough details.

The use of pseudo-code is optional, and it is your decision. No matter you use it or not, above description in words is always required. The pseudo-code has its own advantage in explaining structured (i.e., if-else, for-loop, recursive functions, etc) algorithms and in putting details in the right place. If you think pseudo-code better explains your algorithm, and/or helps graders understand your solution, and/or contains more details not included in the plain-wording description, then use pseudo-code. If you think everything is already clearly explained in the description with words, then you don't need to include pseudo-code. An algorithm that is only written in pseudo-code (i.e., missing above plain-wording description) is not acceptable, as it is extremely hard to read just pseudo-code without any explanation.

Here is a general situation that may help you decide whether to use pseudo-code or not. An algorithm could be "designed from scratch", i.e., you will need to come up with the step-by-step procedure. This usually involves in implementing a function with clear input and output. In this case, including pseudo-code usually helps. All algorithm we've seen so far (e.g., merge-two-sorted-arrays, merge-sort, etc) falls in this category. Second, an algorithm could also be "transformed into another algorithm", i.e., you use an existing algorithm to solve this problem. In this case you usually don't need to include pseudo-code but to describe how to transform one problem into the other. We will see such examples soon.

Problems

Problem 1. Suppose you have m sorted arrays, each with n elements, and you want to combine them into a single sorted array with mn elements. **[25 points]**

(a) If you do this by merging the first two arrays, next with the third, then with the fourth, until in the end with the last one. What is the time complexity of this algorithm, in terms of m and n? Please also provide the analysis.

(b) Give a more efficient solution to this problem, using divide-and-conquer. What is the running time? Please also provide the analysis.

Problem 2. Solve the following recurrence relations. Give the closed form of $T(n)$ in the Θ / $\mathcal{O}$ notation **[40 points]**

(a) $T(n) = 8 \cdot T(n/2) + 100 \cdot n^3$.

(b) $T(n) = 8 \cdot T(n/2) + 1000 \cdot n^{3.5}$.

(c) $T(n) = 16 \cdot T(n/2) + n \cdot \log(n)$

(d) $T(n) = 2 \cdot T(n/2) + n \cdot \log(n)$.

(e) $T(n) = 8 \cdot T(n/2) + n^{3.5} \cdot \log^2(n)$.

(f) $T(n) = T(n/2) + 1.5^n$.

(g) $T(n) = T(3n/5) + T(2n/5) + O(n)$ [Hint: Use induction. To elaborate, try to find a function g such that for all $k < n$, there is a constant $c_1 > 0$ such that $T(k) \leq c_1 \cdot g(k)$; prove that the inequality hold for every k by induction.]

(h) $T(n) = \sqrt{n} \cdot T(\sqrt{n}) + 10n$ [Hint: unfold the recursive relation by $\log \log n$ steps.]

Problem 3. Merge Sort Implementation **[35 points]**

Problem Description

Implement merge-sort to sort an array.

Input

Two lines, the first line gives only one number `n`, the number of integers in the array. And the second line gives `n` numbers which are the `n` integers in the array. All numbers are separated by space.

You can assume that $0 \leq n \leq 10000$, and that each number in the arrays are in the range of $[-2147483648, 2147483647]$.

Your code should read the input from standard input (e.g. using functions `input()/raw_input()` in Python and `cin/scanf` in C++).

Output

One line, describing the sorted array, in ascending order. There should be `n` integers in a line give all numbers in the sorted array (in ascending order). All numbers should be separated with space. (You don't need to output the size of the array `n`.)

Your code should write the output to standard output (e.g. using functions `print` in Python and `cout/printf` in C++).

Requirement

Your algorithm should run in $O(n \log n)$ time. You are not allowed to call any in-built sort function.

Time limitation: 5 seconds.

Memory limitation: 1.0 GB.

Examples and Testing

Some examples (e.g., input-x.txt and output-x.txt, x = 1, 2) are provided.

For Python code, try the following to test your code:

```
python ./solution.py < input-x.txt > my-output-x.txt
```

For C++ code, try the following to test your code:

```
g++ -o mybinary solution.cpp
./mybinary < input-x.txt > my-output-x.txt
```

Your output `my-output-x.txt` needs to be **match exactly** to the given `output-x.txt`.

On Unix-based systems you can use `diff` to compare them:

```
diff my-output-x.txt output-x.txt
```

On Windows you can use `fc` to compare them:

```
fc my-output-x.txt output-x.txt
```

Submission
You may upload a single solution file (.py or .cpp)