

# Homework 2: Searching and Sorting

Due: February 4, 2026 at 11:59 p.m.

This homework must be typed in L<sup>A</sup>T<sub>E</sub>X. Please use the code released as part of the homework, remove these instructions and fill in the solutions.

Please ensure that your solutions are complete, concise, and communicated clearly. Use full sentences and plan your presentation before you write. Except where indicated, consider every problem as asking for a proof.

Your solutions must be submitted using Gradescope. When doing so, please correctly mark the pages with the solutions to each problem.

**Problem 1.** A *geometric sequence* with common ratio  $r$  is a sequence of numbers given by:

$$a_1, a_1r, a_1r^2, a_1r^3, \dots$$

For example, the following is a geometric sequence with a common ratio 2.

$$1, 2, 4, 8, 16, \dots$$

- (a) Describe an algorithm to find the value of a deleted term of a geometric sequence of length  $n$  with common ratio  $r$  in  $O(\log n)$  time. For example, the sequence

$$1, \frac{1}{3}, \frac{1}{9}, \frac{1}{81}, \frac{1}{243}$$

is missing the term  $\frac{1}{27}$ .

- (b) Provide proof of the correctness of the algorithm.
- (c) Prove an upper bound on the running time of your algorithm (asymptotic analysis is sufficient).

*Solution.*

□

**Problem 2.** Let  $X = [a_1, \dots, a_n]$  and  $Y = [y_1, \dots, y_n]$  be two sorted arrays (in non-decreasing order). For simplicity, assume  $n$  is a power of 2.

- (a) Describe an algorithm to find the median of all  $2n$  elements in the arrays  $X$  and  $Y$  in  $O(\log n)$  time.
- (b) Provide a succinct proof of the correctness of the algorithm.
- (c) Provide an analysis of the running time (asymptotic analysis is correct) and memory utilization of the algorithm.

*Hint:* Note that the given arrays are already sorted and of the **same size**! You may want to use binary search to exploit this fact. :)

*Solution.*

□

**Problem 3.** Let  $A$  be an array of  $n$  *distinct* integers. An *inversion* in  $A$  is a pair of indices  $i$  and  $j$  such that  $i < j$ , but  $A_i > A_j$ . For example, the following sequence has three *inversions*:

$$\{1, 5, 2, 8, 4\}$$
$$(5, 2), (5, 4), (8, 4)$$

- (a) Provide a succinct (but clear) description of an algorithm running in  $O(n \log n)$  time to determine the number of *inversions* in  $A$ . You may provide a pseudocode.
- (b) Provide a succinct proof of the correctness of the algorithm.
- (c) Provide an analysis of the running time (asymptotic analysis is correct) and memory utilization of the algorithm.

*Solution.*

□

**Problem 4.** Through this problem, assume you are working only with comparison-based sorting algorithms. All following questions refer to the following standard insertion sort:

```

for j = 2 to n:
    key = A[j]
    i = j - 1
    while i >= 1 and A[i] > key:
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key

```

You may assume constant-time overhead per loop iteration.

### Part I: Inversions and Insertion Sort

As previously defined, an *inversion* in array  $A$  is a pair  $(i, j)$  with  $i < j$  and  $A[i] > A[j]$ . Let  $\text{inv}(A)$  denote the number of inversions.

1. Show that each execution of the inner **while** loop removes exactly one inversion.
2. Deduce that insertion sort runs in

$$\Theta(n + \text{inv}(A)).$$

### Part II: $k$ Elements in Correct Final Positions (Known)

Exactly  $k < n$  elements are in their final sorted positions, and the algorithm **is told which elements these are**.

1. Analyze the running time of **standard insertion sort** under this assumption. (Do not modify the algorithm.)
2. Prove a comparison-based lower bound under this assumption.
3. Design a comparison-based sorting algorithm that exploits this knowledge to run in

$$O((n - k) \log(n - k)).$$

### Part III: $k$ Elements in Correct Final Positions (Unknown)

Exactly  $k < n$  elements are in their final positions, but the algorithm **does not know which elements**. Access is only via comparisons.

1. Give an asymptotic upper bound on  $\text{inv}(A)$  in terms of  $n$  and  $k$ .
2. Using Part I, analyze the running time of insertion sort.
3. Compare this running time with that obtained under the assumption of Part II. What do you observe?
4. Prove a comparison-based lower bound under this assumption.
5. Would the standard Merge Sort algorithm's running time be affected by this assumption? An informal explanation (without proof) will be sufficient.

**Part IV:  $k$  Elements in Correct Relative Order**

The array contains a strictly increasing subsequence of length  $k$ , but these elements are not necessarily in their final positions.

1. Give an asymptotic upper bound on  $\text{inv}(A)$  in terms of  $n$  and  $k$ .
2. Using Part I, analyze the running time of insertion sort.
3. Prove a comparison-based lower bound under this assumption.
4. Are there scenarios where insertion sort may perform better than Merge Sort under this assumption? An informal explanation (without proof) will be sufficient.

*Solution.*

□