Algorithms **Spring Semester, 2026**

Homework 2 (NOT Graded)

Professor: Dr. Mudassir Shabbir *Deadline: Feb 02, 2026.*

This homework contains a set of practice problem and **no submission is required**.

1. *This problem models searching for a "self-consistent" index, which arises in fixed-point computations and debugging sorted data tables.* Suppose an array contains distinct integers. Design a divide-and-conquer algorithm to determine whether there exists an index i such that $A[i] = i$.
2. Solve the recurrence $T(n) = 2T(n/2) + n \log n$.
3. Solve $T(n) = 3T(n/4) + n$ and justify which case of the Master Theorem applies.
4. Give an example of a recurrence where the Master Theorem does *not* apply and explain why.
5. Show that if QuickSort always picks the smallest element as pivot, its running time becomes $\Theta(n^2)$.
6. What is the probability that the pivot chosen is among the smallest 10% of elements?
7. Show that MergeSort is stable.
8. The Karatsuba algorithm, discovered by Anatolii Karatsuba in 1960, was the first multiplication algorithm asymptotically faster than the classical "grade school" method. It elegantly demonstrates the power of divide and conquer by reducing the number of recursive multiplications needed. Consider multiplying two 4-digit numbers using the standard algorithm you learned in elementary school.

   (a) How many single-digit multiplications does the classical algorithm require to multiply two n-digit numbers? Express your answer as a function of n and explain your reasoning.

   (b) Let's multiply two 2-digit numbers: $X = 52$ and $Y = 37$. We can represent these as:

   $$X = 5 \times 10^1 + 2 \times 10^0 = 10 \cdot x_1 + x_0 \quad \text{where } x_1 = 5, x_0 = 2$$
   $$Y = 3 \times 10^1 + 7 \times 10^0 = 10 \cdot y_1 + y_0 \quad \text{where } y_1 = 3, y_0 = 7$$

   Write the expression for $X \times Y$ in terms of x_1, x_0, y_1, and y_0? How many multiplications of single-digit numbers does your expression require? This is still the classical approach. What's the issue with using this as a divide-and-conquer algorithm for large numbers?

   (c) Karatsuba observed that we can compute $X \times Y$ using only **THREE** multiplications instead of four. Given:

   $$z_2 = x_1 \times y_1$$
   $$z_0 = x_0 \times y_0$$
   $$z_1 = (x_1 + x_0) \times (y_1 + y_0) - z_2 - z_0$$

   Prove algebraically that $X \times Y = z_2 \times 10^2 + z_1 \times 10^1 + z_0$.

   (d) Verify this works for $X = 52$ and $Y = 37$ by computing all three z values.

   (e) Explain why computing z_1 this way is clever. What computation does it avoid?

   (f) Write pseudocode for the Karatsuba algorithm that works for n-digit numbers where n is a power of 2. Your pseudocode should:

   - Handle the base case appropriately
   - Recursively compute the three products
   - Combine results correctly

   (g) Let $T(n)$ be the number of single-digit multiplications needed to multiply two n-digit numbers using Karatsuba's algorithm. Write the recurrence relation for $T(n)$ and use Master Theorem to solve it.

(h) What is the time complexity when we count all operations (multiplications, additions, and digit shifts)? Assume all single digit operations (e.g., adding two single digit numbers, multiplying two single digit numbers, shifting a single digit), take constant time.

9. *Fast exponentiation appears in cryptography, scientific computing, and modular arithmetic used in security protocols.* Design a divide-and-conquer algorithm to compute a^n for integer $n \geq 0$. Compare it with the naive repeated multiplication method. You can use the Karatsuba Algorithm from the previous problem or the naive classical algorithm for Time Complexity of multiplying two numbers.

10. *Closest-point queries arise in sensor readings, timestamps, and financial tick data.* If the points are already sorted, give an $O(n)$ divide and conquer algorithm to find the closest pair of real numbers on a line.

11. *Duplicate detection is central in data cleaning, fraud detection, and log analysis.* Suppose you have a black-box function that tells you whether a subarray contains at least one duplicate element in time $f(n)$.

    Design a divide-and-conquer algorithm that finds a duplicate if one exists.*

12. *Detecting whether a signal or message stream is biased toward 1s, is an important problem in error detection and communications research.* You are given a binary string of length n. Design a divide-and-conquer algorithm that counts the number of substrings that contain more 1s than 0s.†

13. Construct a recurrence that *almost* satisfies case 2 of the standard Master Theorem but violates the regularity condition.‡

14. Suppose an algorithm splits into 5 subproblems of size $n/3$ and does n work outside recursion. Without solving the recurrence exactly, argue whether the runtime is closer to n, $n \log n$, or $n^{1.5}$.§

15. *Hybrid sorting (QuickSort + insertion sort) is used in real libraries such as C++ STL and Java.* Modify Randomized QuickSort so that it stops recursing on subarrays of size $\leq k$ and instead finishes using insertion sort. Argue how to maximally choose k so that the expected asymptotic runtime remains $O(n \log n)$.

16. Suppose QuickSort always chooses a pivot uniformly from the first $\sqrt{n}$ elements. Is the expected runtime still $O(n \log n)$?¶

17. *This question helps explain why QuickSort spreads its work across elements fairly evenly on average.* Consider the standard version of randomized QuickSort. Let X_k be the number of elements that are compared to the k-th smallest element. Find

18. *Cycle detection is used in deadlock detection, dependency resolution, and build systems.* Write pseudocode for DFS and explain how it can be used to detect cycles in a graph.

19. *Manhattan distance is used in grid navigation, robotics, and routing on city maps.* For the closest pair problem in 2D, Suppose the distance metric is Manhattan distance instead of Euclidean. Does the divide-and-conquer algorithm we discussed in class still work in $O(n \log n)$ time?‖

20. *This measures disagreement between two ranked lists — for example, how two users rate movies differently in a recommendation system.* Given two permutations P and Q, define their distance as the number of pairs ordered differently in the two permutations. Show how to compute this in $O(n \log n)$ time.**

21. *Weighted inversions arise when late mistakes are more costly than early ones, such as scheduling delays or ranking penalties.* Define a *distance-weighted inversion* as (i, j) with $i < j$ and $A[i] > A[j]$, weighted by $j - i$. Design an $O(n \log n)$ algorithm to compute the total weight.††

---

*Hint: Split the array into halves. If neither half contains a duplicate internally, where must a duplicate lie? What information would help you detect it?

†Think of translating the string into $+1$ and -1. The problem becomes counting subarrays with positive sum. What information must each recursive call return so the merge step can count cross-boundary substrings?

‡Try adding a slowly growing factor like $\log \log n$.

§Compare $n^{\log_3 5}$ with n.

¶What is the probability that such a pivot lies in the middle half of the array?

‖What geometric property of the strip argument must still hold?

**Map one permutation into the index order of the other.

††During merge, what prefix sums must you maintain?

22. Show that reversing a sorted array of size n produces exactly $\binom{n}{2}$ inversions.

23. Give an input where MergeSort performs exactly $n \log_2 n - n + 1$ comparisons.

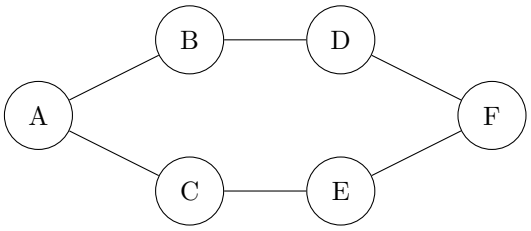
24. *Detecting already-sorted data allows systems to avoid unnecessary work (e.g., log files, nearly sorted databases).* Modify MergeSort so that it stops merging once it detects the two halves are already ordered, i.e, the merge won't change the order at all. What single comparison could detect this? What is the best-case runtime for this modified algorithm?

25. *Reducing extra memory is crucial in embedded systems and large-scale external sorting.* MergeSort is not an in-place sorting algorithm as it requires linear amount of extra space. Design a MergeSort variant that uses only $O(\sqrt{n})$ extra memory.[‡‡]

26. *Write an algorithm to find the number of connected components in a graph.*

27. *BFS and DFS*

    Consider the following undirected graph with vertices: A, B, C, D, E, F.



    Assume that neighbors are added to the queue (for BFS) or considered (for DFS) in **alphabetical order**.

    (a) Start BFS from vertex A. Write the order in which vertices are **discovered** (added to the queue).

    (b) In the BFS tree, what is the **minimum number of edges** required to reach vertex F from vertex A?

    (c) Perform a DFS starting from vertex A. Rule: Always explore the **alphabetically first unvisited neighbor**. Trace your recursion: $A \to B \to \ldots$
    Write the full **DFS discovery order**.

    (d) Which vertex was discovered **last** in BFS?

    (e) Which vertex was discovered **last** in DFS?

    (f) In general, which algorithm is better for finding **shortest paths in unweighted graphs**? Explain why.

[‡‡] Think about merging blocks instead of individual elements.