COMP 3711H Design and Analysis of Algorithms
2026 Fall Semester
Homework 1
Handed out: Feb 20
Due: Mar 6

Problem 1. Consider the recurrence for the running time $T(n)$ of a divide-and-conquer algorithm:

$$T(1) = 1, \qquad T(n) = 6\,T(n/2) + n^2 \quad \text{for } n > 1.$$

Assume n is a power of 2. Answer the following questions about the *recursion tree*.

(a) How many subproblems (nodes) are there at level i of the recursion tree? (The root is level 0.)

(b) What is the input size of each subproblem at level i?

(c) For a single subproblem at level i, how much *non-recursive* work is done at that node (i.e., the *combine* cost only; do *not* include work from recursive calls)?

(d) What is the total *non-recursive* work summed over all subproblems at level i?

(e) How many levels are in the recursion tree *including* the root and the leaves?

(f) Give an asymptotically tight upper bound on $T(n)$ (try to express your answer as n raised to a suitable power).

Problem 2. Repeat all parts of Problem 1 for the recurrence:

$$T(1) = 1, \qquad T(n) = 3\,T(n/2) + n^2 \quad \text{for } n > 1,$$

again assuming n is a power of 2.

Problem 3. For each recurrence in Problems 1 and 2, establish an asymptotically tight upper bound on $T(n)$ using *mathematical induction*. You may assume $n = 2^k$ for an integer $k \geq 0$ (i.e., induct on k).

Problem 4. Assume f and g are non-negative functions and that $f(n) = O(g(n))$. For each statement below, decide whether it is true or false and give a proof or a counterexample.

(a) $2^{f(n)}$ is $O(2^{g(n)})$.

(b) $(f(n))^3$ is $O((g(n))^3)$.

Problem 5. Give asymptotically tight upper bounds for $T(n)$. *Just give the answers; no explanation is needed.*

For parts (a), (b), and (d), you may assume $T(1) = 1$ and n is a power of 2. For parts (c) and (e), you may assume $T(1) = 1$ and n is a positive integer.

(a) $T(n) = 4\,T(n/2) + n$, for $n > 1$.

(b) $T(n) = 7\,T(n/2) + n^2$, for $n > 1$.

(c) $T(n) = T(n-1) + n^2$, for $n > 1$.

(d) $T(n) = T(n/2) + 10$, for $n > 1$.

(e) $T(n) = 3\,T(n-1) + 1$, for $n > 1$.

Problem 6. Arrange the following running times in order of increasing asymptotic complexity:

$$2^{n^2}, \quad 3^n, \quad n^{4/3}, \quad n \log n, \quad n^{\log n}, \quad 2^{2^n}, \quad 2^{\sqrt{\log n}}.$$

Write $f(n)$ before $g(n)$ if $f(n) = O(g(n))$. *Just give the answer; no explanation is needed.*

Problem 7. Suppose we run *Randomized-Quicksort* on the input

$$8,\ 5,\ 6,\ 12,\ 3,\ 15,\ 1,\ 9.$$

Assume the algorithm chooses pivots uniformly at random from the current subarray at each recursive call. Condition on the event that the *first* pivot chosen is 5.

Answer parts (a)–(c) below, where "compared" means compared at any time during the entire execution.

(a) What is the probability that 8 is compared with 3?

(b) What is the probability that 8 is compared with 15?

(c) What is the expected number of distinct items that get compared with 8?

Problem 8. You have k sorted arrays, each containing n numbers, and you want to combine them into a single sorted array of kn numbers. We will use the standard MERGE procedure from mergesort: merging two sorted arrays of sizes x and y takes $O(x + y)$ time.

(a) Consider the strategy: merge array 1 with array 2, then merge the result with array 3, then merge the result with array 4, and so on. What is the running time in terms of k and n?

(b) Give a more efficient algorithm and analyze its running time. For full credit, your algorithm should run in $O(nk \log k)$ time.

Problem 9. You are stress-testing a particular jar model to find the highest rung of a ladder from which it can be dropped without breaking. The ladder has rungs numbered $1, 2, \ldots, n$.

Assume there exists a threshold rung h (possibly 0) such that: dropping from rungs $\leq h$ never breaks the jar, and dropping from rungs $> h$ always breaks the jar. A jar that breaks cannot be used again. Each drop counts as one experiment.

(a) You have a budget of 2 jars. Describe a strategy that finds h using at most $f(n)$ drops in the worst case. Make $f(n)$ as small as possible, and give your bound using asymptotic notation.

(b) Same question as in part (a), but with a budget of 3 jars.

Problem 10. In the *max-min* problem, you must find both the largest and the smallest number in an array of n values.

(a) Show that when $n = 8$, the max-min problem can be solved using at most 10 comparisons. (For reference: the naive approach of scanning twice takes 14 comparisons.)

(b) Assume n is a power of 2. Design a divide-and-conquer algorithm for max-min that uses at most $3n/2 - 2$ comparisons, and prove your algorithm meets this bound.