

CSCI 0500: Data Structures, Algorithms, and Intractability (Fall 2026)

Assignment 1

Due at 11:59pm ET, Friday, Sep 25

1. (1 point) Recall from lecture the definitions of big- O and big- Ω we use in this course.

Definition 1 (Big- O). We say $f = O(g)$ if there are constants $c > 0$ and $n_0 \geq 0$ such that $0 \leq f(n) \leq c \cdot g(n)$ for all integers $n \geq n_0$.

Definition 2 (Big- Ω). We say $f = \Omega(g)$ if there are constants $c > 0$ and $n_0 \geq 0$ such that $f(n) \geq c \cdot g(n) \geq 0$ for all integers $n \geq n_0$.

- (a) Decide if the following statement is true or false: $f = O(g)$ if and only if $g = \Omega(f)$. Briefly justify your answer.

The following lemma gives a sufficient (but not necessary) condition for $f = O(g)$.

Lemma 1. Suppose $f(n) \geq 0$ and $g(n) > 0$ for all sufficiently large n . If $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ exists and is finite, then $f = O(g)$.

You are encouraged to try to prove this lemma, but it is not required for this assignment.

- (b) Arrange the following functions in ascending order of their asymptotic growth rate. If $g(n)$ immediately follows $f(n)$ in your list, it should hold that $f(n) = O(g(n))$. (You do not need to justify your ordering, but you should know how to do so using Definition 1 or Lemma 1.)

$$f_1(n) = \frac{1}{10}n^2 \ln n$$

$$f_2(n) = 2^n$$

$$f_3(n) = \ln^3 n = (\ln n)^3$$

$$f_4(n) = \ln \ln n = \ln(\ln(n))$$

$$f_5(n) = 5n^{4/3}$$

$$f_6(n) = n!$$

2. (1 point) In this question, we study applications and analysis of sorting and selection.

- (a) Consider the problem of finding the t largest elements in an array X . Given a 0-indexed n -element array X and an integer t with $1 \leq t \leq n$, the goal is to return the t largest

elements in X (in any order). The computational cost (i.e., runtime) is measured by the number of pairwise comparisons between elements in X .

Design a randomized algorithm with worst-case expected runtime $O(n)$ and prove that it achieves this runtime. You can assume that all elements in X are distinct.

(Hint: You can use `quickselect` as a subroutine without implementing it, and use its runtime upper bound without proving it: `quickselect(X, k)` returns `sorted(X)[k]`, the $(k + 1)$ -th smallest element in X , and has worst-case expected runtime $O(n)$.)

- (b) Consider the problem of deciding whether an element appears in a sorted array. Given a 0-indexed array S with n elements in ascending order and another element v , the goal is to return `True` if v appears in S and `False` otherwise. The runtime is measured by the number of pairwise comparisons between v and elements in S .

Design a deterministic algorithm with worst-case runtime $O(\log n)$ and prove that it achieves this runtime. You can assume that all elements in S are distinct.

- (c) Consider running randomized quicksort on the input $X = (11, 3, 22, 7, 2, 15, 5, 30)$. Recall that each pivot is chosen uniformly at random from the current subarray.

During the entire execution of `quicksort(X)` (including recursive calls), what is the probability that 5 and 7 are compared, and what is the probability that 3 and 15 are compared? Briefly justify your answers.

3. (1 point) In this question, we explore non-comparison-based sorting algorithms.

Consider sorting an n -element array X , where each element is an integer in $\{0, 1, \dots, n - 1\}$ (possibly with duplicates). Consider the following algorithm `sort1`.

```
def sort1(X):
    n = len(X)
    count = [0 for i in range(n)]
    for a in X:
        count[a] += 1
    S = []
    for i in range(n):
        for j in range(count[i]):
            S.append(i)
    return S
```

For this problem, we measure computational cost (i.e., runtime) by the number of array entries created plus the number of array accesses (for all three arrays X , S , and `count`).

- (a) Prove that the worst-case runtime of `sort1` is $\Theta(n)$.
 (b) Does this contradict the $\Omega(n \log n)$ lower bound on the worst-case runtime of comparison-based sorting? Briefly justify your answer.

Remark. Suppose now each element of X is an integer in $\{0, 1, \dots, n^2 - 1\}$ instead of $\{0, 1, \dots, n - 1\}$. How can we still sort X in $\Theta(n)$ time? You are encouraged to think about this, but it is not required for the assignment.