

This exam is open-book and closed-collaboration. You have 120 minutes to complete the exam and submit your answers on Gradescope. You may consult inanimate materials or resources, including course materials, but inputting any exam questions or solutions into **any software, apps, or websites (Google, Stack Overflow, etc.) is in direct violation of the honor code.** You may review the Stanford Honor Code online at the following link.

<https://communitystandards.stanford.edu/policies-and-guidance/honor-code>

You may also review the detailed rules and instructions on the Exam portion of the CS 161 webpage, linked below.

<https://stanford-cs161.github.io/winter2021/exams/>

Good luck!

Section 1. Short Answer (20 points)

No explanation is required for the questions in Section 1. Please clearly mark your answers; if you must change an answer, either erase thoroughly or else make it **very** clear which answer you intend. **Ambiguous answers will be marked incorrect.**

Throughout this section, mark **ALL answer choices that apply.** Also, you may assume that $T(1) = 1$ for question (ii) and $T(1) = T(2) = 1$ for question (iii). Furthermore, if you are writing your solutions on another document, please **box** your choices of **(A)**, **(B)**, **(C)**, and/or **(D)** to make it **unambiguously clear** which answers you selected for each of questions (i) through (v).

- (i) **(4 pt.)** Which of the following correctly describes $T(n) = n^2 + 5 \log n + 10$?
(A) $O(n)$ **(B)** $O(n^2)$ **(C)** $\Omega(n^2 \log n)$ **(D)** $\Omega(n)$
- (ii) **(4 pt.)** Which of the following correctly describes $T(n) = T(n-1) + n$?
(A) $O(n^3)$ **(B)** $\Omega(n)$ **(C)** $\Omega(n^2)$ **(D)** $\Theta(n \log(n))$
- (iii) **(4 pt.)** Which of the following correctly describes $T(n) = 9T(n/3) + n^2$?
(A) $O(n^3 \log(n))$ **(B)** $O(n \log^2(n))$ **(C)** $O(n^2)$ **(D)** $\Theta(n^3)$
- (iv) **(4 pt.)** Let $T(n) = n^c$ for some constant $c > 1$. Which of the following describes $2^{T(n)}$?
(A) $O(2^n)$ **(B)** $\Omega(2^n)$ **(C)** $\Theta(2^{2 \cdot T(n)})$ **(D)** $\Omega(2^{100nc})$
- (v) **(4 pt.)** Which of the following correctly describes $T(n) = (n - 2 \lfloor \frac{n}{2} \rfloor) n$?
(A) $O(n^2)$ **(B)** $\Omega(n^2)$ **(C)** $O(n)$ **(D)** $\Omega(n)$

Section 2. Incorrect Proof (20 points)

Lucky the Lackadaisical Lemur is studying sorting algorithms in CS 161. After learning about MergeSort and performing some calculations of his own, he believes that he has a new, remarkable result to share with the computer science community: MergeSort runs in $O(n)$ time when run on an array A of length n .

He is super excited, because he has established a better bound for MergeSort than the $O(n \log n)$ bound we saw in class. Before releasing his result to the academic community, he decides to run his solution by you to double-check. His proof that MergeSort runs in $O(n)$ time is broken down into four parts below.

In each part, either assert that his logic is correct, giving 1-2 sentences explaining how you came to that conclusion, or assert that there is a flaw and give a brief explanation about why it is wrong. **Only state that a portion is incorrect if there is a mathematical error contained within.** Note that Lucky is allowed to ignore divisibility issues (like $\frac{n}{2}$), so **ignore any of his potential divisibility "errors."**

- (i) **(5 pt.)** Let $T(n)$ be the running time of MergeSort when run on a list of length n . we know that T satisfies the recurrence $T(n) \leq 2T\left(\frac{n}{2}\right) + cn$ for some constant c .

- (ii) **(5 pt.) Inductive Hypothesis:** There exists a constant c' such that $T(i) \leq c'i$.

Base Case: Let $i = 1$. $T(1)$ is a constant because a list of length 1 is always sorted. Therefore, we can pick a sufficiently large constant c' such that $T(1) \leq c'$, satisfying the inductive hypothesis for $i = 1$.

(iii) **(5 pt.) Inductive Step:** Let k be an integer ≥ 2 . Assume that the inductive hypothesis holds for all positive integers $i \leq k - 1$. We will show that the inductive hypothesis holds for $i = k$. Note that $T(k) \leq 2T\left(\frac{k}{2}\right) + ck$. Since $k/2 \leq k - 1$, we can apply the inductive hypothesis for $i = k/2$ to upper bound the $2T\left(\frac{k}{2}\right)$ term. Hence, there exists a constant c' such that $2T\left(\frac{k}{2}\right) \leq 2c'\left(\frac{k}{2}\right) = c'k$. Therefore,

$$T(k) \leq 2T\left(\frac{k}{2}\right) + ck \leq c'k + ck = (c' + c)k$$

Setting $c'' = c' + c$, we see that there exists a constant c'' such that $T(k) \leq c''k$. Hence the inductive hypothesis holds for $i = k$.

(iv) **(5 pt.) Conclusion:** It follows by induction that for any integer n , there exists a constant c' such that $T(n) \leq c'n$. Hence $T(n) = O(n)$.

Section 3. Algorithm Design (50 points)

After his mistakes understanding the runtime of MergeSort, Lucky has gotten into the habit of running his proofs by Plucky the Pedantic Penguin. Plucky is preoccupied with a lot of work, but still wants to help Lucky as much as possible.

Lucky has an estimate of how long it takes to run each proof by Plucky. So he sends an array $A = [a_1, \dots, a_n]$ of these time lengths (measured in minutes) to Plucky. Plucky has put aside a total of L minutes to help Lucky, and wants to go over **as many proofs as possible**.

In this problem, you will design an algorithm that helps Plucky find out the maximum number of proofs that fit within L minutes. You **may assume all time lengths in A are distinct**. If it helps, **you may ignore divisibility issues** (such as floors and ceilings) throughout this problem. You may use any algorithm we have learned in class and cite its correctness and runtime without proof.

Input: Array A of n distinct positive numbers indicating the time it takes to verify each proof, and number $L \geq 0$ indicating how much time Plucky has put aside.

Output: The maximum number of proofs that Plucky can verify in $\leq L$ minutes.

Examples. You can find some sample inputs and the correct outputs below.

- For $A = [5, 1, 6, 2, 3]$ and $L = 3$, the correct output is 2.
- For $A = [20, 10, 30]$ and $L = 5$, the correct output is 0.
- For $A = [4, 1, 2, 3]$ and $L = 1000$, the correct output is 4.

By following the steps below you will design an algorithm that solves this problem and runs in time $O(n)$, where n is the length of A . If the algorithms you design for parts (i), (ii), and (iii) have bugs or do not have the correct runtime, they **might still receive partial credit**.

- (i) **(10 pt.) Warmup.** Briefly describe or write pseudocode for an $O(n \log(n))$ time algorithm $\text{BaseLineAlgorithm}(A, L)$ that solves this problem. You do not need to prove its correctness or runtime in this part.

- (ii) **(15 pt.) Small or Large.** Briefly describe or write pseudocode for an $O(n)$ time algorithm `IsAnswerAtLeast(A, L, k)` which in addition to A and L receives a number k between 0 and n inclusive, and outputs whether there are **at least** k proofs that Plucky can verify in L minutes. Note that the output here is not a number; it is just True (if Plucky can verify k proofs) or False (if Plucky cannot). You do not need to prove the correctness or runtime of your algorithm in this part.

Below you can find some example inputs and outputs for `IsAnswerAtLeast`.

- For $A = [5, 1, 6, 2, 3]$ and $L = 3$ and $k = 2$, the correct output is True.
- For $A = [5, 1, 6, 2, 3]$ and $L = 3$ and $k = 3$, the correct output is False.
- For $A = [20, 10, 30]$ and $L = 5$ and $k = 0$, the correct output is True.
- For $A = [20, 10, 30]$ and $L = 5$ and $k = 3$, the correct output is False.

- (iii) **(15 pt.) Algorithm.** Briefly describe or write pseudocode for an $O(n)$ time algorithm `MaximumProofs(A, L)` that solves the original problem, that is, outputs the maximum number of proofs that Plucky can verify in L minutes. You do not need to prove its correctness or runtime in this part.

(Hint: This is the hardest problem on the exam! If you get stuck, move to other unsolved problems.)

- (iv) **(10 pt.)** Prove the runtime of your algorithm (`MaximumProofs`) from part (iii). That is, either argue using a recurrence relation or otherwise that your algorithm satisfies the time bound of $O(n)$.