

General Instructions

1. Please type your answers or write clearly. There are five problems on the exam, please start each problem on a different sheet of paper.
2. Express your answers **clearly** and **precisely**. Informal answers get very little credit even if they express the right intuition. **Conciseness** is also important: lengthy or less pretty answers get less credit.
3. Read each question carefully. Unfortunately, if you misread and answer a different question than the one asked, you won't get credit.
4. Even if you did very well on the homeworks, I urge you to carefully study the homework solutions supplied to you. I am saying this because I expect your midterm answers to be as clear and precise as the solutions distributed. The midterm grading will be much stricter than the homework grading (because you now have the experience and access to sample solutions).
5. Since this is an exam, I cannot help you with the particular problems on this exam. However, as you attempt to solve these problems, if you discover that your understanding is not complete on some topics, please see me or the TAs and we can help you with the concepts. To be fair to all, please don't ask me to check if your solution is correct or any other exam specific question.
6. **Do not discuss these problems with your classmates or anyone else.** Unlike the homeworks, the work on the exams must be **entirely** yours. You may not consult any source other than the text book (CLRS) and your class notes. In particular, you should not look up other books, internet, handwritten or printed material from previous years, including homework/exam solutions distributed from previous offerings of this course.
7. Please refer to the *Course Syllabus* (on the CS 25 web page) for the honor code and also note that Dartmouth Honor code applies to this exam.

Specific Instructions

1. When describing an algorithm, you can use pseudocode or English, whichever is convenient. The important thing is that the description be both unambiguous and clear to the reader.
2. **Whenever** you are asked to give an algorithm, even if the problem does not explicitly ask for all of the following three parts, **you must supply** all of of the following three parts:
 - (a) The algorithm.
 - (b) Proof of correctness of the algorithm, i.e., a clear and convincing argument that the algorithm is correct.
 - (c) Analysis of its running time.

Please note that all three components are assigned points in our grading guide and so an algorithm, even if correct, gets little credit unless the other components are also supplied.

3. When solving a **dynamic programming** problem, be sure to include all of the following steps:
 - (a) Describe clearly the optimal substructure of the problem (i.e., how a solution to a problem can be constructed from solutions to smaller subproblems).
 - (b) Use appropriate notation to clearly state what the general version of the problem is. If, for instance, you introduce $b[i, j]$, you must clearly state the meaning of the value in $b[i, j]$.
 - (c) Using the notation that you introduced, state the recurrence, including all the base cases.
 - (d) Give the pseudocode that outputs an optimal solution.
 - (e) Analyze the time and space requirements of your algorithm. **Note** that a less efficient algorithm gets a lot fewer points, even if correct.
4. When designing a **greedy algorithm**, be sure to include all of the following steps:
 - (a) State the greedy choice.
 - (b) Prove the greedy choice property, i.e., that the greedy choice leads to exactly one subproblem such that the solution to the subproblem, together with the greedy choice, yields a solution to the original problem.
 - (c) Give the pseudocode that outputs an optimal solution.
 - (d) Analyze the time and space requirements of your algorithm. **Note** that a less efficient algorithm gets a lot fewer points, even if correct.
5. If you use a red-black tree, be sure to state what the key is, and what the other fields are. Where appropriate, be sure to argue that the additional fields are maintainable without affecting the $O(\lg n)$ performance of *insert* and *delete* (if that happens to be the case).
6. When designing an algorithm, you can use any procedures that we covered in class (e.g., Select, Partition) without giving their code.
7. These instructions are not exhaustive, and are merely a guide to you to emphasize the need for clarity, brevity and rigor.

1. **Short Questions** (6 + 6 + 6 + 10 = 28 points)

- (a) Arrange the following functions in a sequence so that if $f(n)$ comes before $g(n)$ in your sequence, then either $f(n) = o(g(n))$ or $f(n) = \Theta(g(n))$. Once you arrange the functions in a sequence, for every two consecutive functions $f(n)$ and $g(n)$ in the sequence, you should show why $f(n)$ is $o(g(n))$ or $\Theta(g(n))$. Since there are 4 functions below, you need to provide 3 justifications, and each carries 2 points.
(For example, if the functions are $n^2, n^3, 0.5n^2$, you should arrange them in the order $n^2, 0.5n^2, n^3$ (or in the order $0.5n^2, n^2, n^3$) and then state, with justification, the relationships $n^2 = \Theta(0.5n^2)$ and $0.5n^2 = o(n^3)$.)

- i. n^6
 - ii. $\lg(n!)$
 - iii. $n^{\lg n}$
 - iv. $(\lg n)!$
- (b) Part (f) of Problem 4-3 in the book. I require you to use the substitution method to prove your upper and lower bounds.
- (c) Suppose that you are given an unordered list of n integers and a non-negative integer k , where $0 < k < n$. The problem is to find an integer in the list that is the k th closest to the median of the list (if n is even, you can interpret the median to be the lower median). The “closeness” metric is as follows: a is closer to m than b if $|a - m| < |b - m|$. Give a $O(n)$ algorithm to solve this problem.
 For example, if the list is 5, 2, 8, 3, 4, 19, 12 the median is 5. The first closest is 4, the second closest is 3, the third closest is 2 or 8 (either answer is acceptable), the fourth closest is also 2 or 8, the fifth closest is 12, and the sixth closest is 19.
- (d) An array A of n elements has a majority if at least $\lfloor n/2 \rfloor + 1$ of the elements of A are the same. Suppose that the elements of A are drawn from a set whose elements cannot be compared except for equality, i.e., you can test if $A[i] = A[j]$, but you can't test if $A[i] < A[j]$ because “less than” and “greater than” don't make sense for the set the elements are drawn from. Give a $O(n)$ time algorithm to determine if an array A of n elements has a majority.
2. (12 points) Let $P[1], P[2], \dots, P[n]$ be the prices of a stock on n consecutive days. Assume that you are allowed to buy the stock only once (on any day of your choice) which, if you wish, you may sell at a later date. Thus, if you buy the stock on day i and sell it on a later day $j > i$, you will make a profit of $P[j] - P[i]$ if $P[j] > P[i]$. Design an $O(n)$ time algorithm that, given $P[1], P[2], \dots, P[n]$, outputs the maximum profit you can make by buying and selling exactly once. If there is no opportunity to make a positive profit, your algorithm should say so.
 (Note that you are required to design an $O(n)$ time algorithm.)
3. (20 points) Problem 14-1 (the Point of maximum overlap problem).
4. (20 points) Problem 15-2 (the longest palindrome subsequence problem)
5. (20 points) Problem 16-1 (the Coin changing problem)