

Name:

ID Number:

Exam Type:

...A...

Exam Number:

.....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: ... Total:

SOLVE ALL **7 (SEVEN)** PROBLEMS IN THE SPACE PROVIDED

Read the Problems **CAREFULLY!**

THERE ARE 6 (SIX) PAGES THIS PAGE INCLUDED; THE LAST PAGE IS LEFT BLANK
STATEMENT

On my honor, I pledge that I have not violated the provision of the NJIT Student Honor Code.

Sign below at the end of the exam

Signature

Any algorithm you present must be given in concise and complete form. Argue about its correctness. Analyze its (worst-case) running time and express it in asymptotic notation. Random ramblings or sketches will not be given any points. You may use algorithms presented in class as black-boxes without further description. For example, instead of repeating the code of MergeSort you can just write MergeSort(B,m) to indicate that you sort an array B of m keys.

List of useful formulae

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4}, \quad n! \approx \left(\frac{n}{e}\right)^n, \quad a^{\log_b n} = n^{\log_b a},$$

For $x \neq 1$, we have that

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}, \quad \sum_{i=0}^{n-1} ix^i = \frac{(n-1)x^{n+1} - nx^n + x}{(1-x)^2},$$

B1. $f(n) = \Theta(g(n))$ iff $\exists$ positive constants $c_1, c_2, n_0 : 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$.

B2. $f(n) = \Omega(g(n))$ iff $\exists$ positive constants $c_1, n_0 : 0 \leq c_1 g(n) \leq f(n) \quad \forall n \geq n_0$.

B3. $f(n) = O(g(n))$ iff $\exists$ positive constants $c_2, n_0 : 0 \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$.

Master Method. $T(n) = aT(n/b) + f(n)$, such that $a > 0, b > 1, f(n) > 0$.

M1 If $f(n) = O(n^{\lg_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.

M2 If $f(n) = \Theta(n^{\lg_b a} \lg^k n)$, then $T(n) = \Theta(n^{\lg_b a} \lg^{k+1} n)$, where $k \geq 0$ is a non-negative constant.

M3 If $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for large n , then $T(n) = \Theta(f(n))$.

Problem 1. (50 POINTS)

We perform an in-order and pre-order traversal on an arbitrary binary tree. For each one of the traversals we get as output:

In – order : 1, 2, 4, 3, 7, 6.

Pre – order : 4, 2, 1, 7, 3, 6.

Show the binary tree that is consistent with these two traversals.

Problem 2. (50 POINTS)

Solve the following recurrence relations by providing asymptotically tight bounds. You only need to provide the bound, intermediate derivations are not required. If no boundary case is given, the choice (of the constants) is yours. You may assume that $T(n)$ is positive and monotonically increasing, if you need to do so.

$$T(n) = T(n/2) + n.$$

$$T(n) = 2T(n/2) + \sqrt{n}.$$

$$T(n) = T(n - 5) + n$$

Problem 3. (50 POINTS)

Solve exactly the following recurrence. You may assume that n is a power of two.

$$T(n) = 2T(n/2) + 3n, \quad T(8) = 8.$$

Problem 4. (50 POINTS)

Mark clearly as **TRUE** or **FALSE** each one of the following statements. No justification required. Algorithms are the ones of the textbook/notes. All input instances below consist of n keys. Any answer other than a fully written TRUE or FALSE will be considered wrong.

- (1) Between MergeSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \dots, n^3 - 1$.
- (2) Between MergeSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \dots, n^n - 1$.
- (3) Between HeapSort and RadixSort, the former is more time-efficient in sorting n keys in the range $0, \dots, 2^n - 1$.
- (4) The solution of $T(n) = T(n/5) + T(7n/10) + n$, is $T(n) = \Theta(n \lg n)$.
- (5) CountSort does not sort in-place.

Problem 5. (50 POINTS)

For this problem you are allowed to add, multiply, shift or compare numbers in unit time (i.e. $\Theta(1)$ time); you are not allowed to use operations such as $\sqrt{\cdot}$. We are given an integer n and we are interested in determining whether n is a perfect square i.e. whether there exists another integer m such that $m^2 = n$. One way to solve this problem is to try all the numbers between 1 and about $\sqrt{n}$ (if you know how to find an approximation of the square root using the allowable operations) for m , an algorithm that requires $O(\sqrt{n})$ time. This is too slow. Can you solve this problem faster than $\Omega(\sqrt{n})$ time? Explain by providing an **efficient** algorithm that solves this problem. Analyze its worst-case running time.

Problem 6. (50 POINTS)

Professor I. M. Nuts of the Math Department is scratching his head over the following problem. Given two sequences of numbers $X = \langle x_1, \dots, x_n \rangle$ and $Y = \langle y_1, \dots, y_n \rangle$ he must find the $m \leq \sqrt{n}$ largest values of the sums $x_i + y_j$, $1 \leq i, j \leq n$ in SORTED ORDER (smallest to largest). One may assume that $m \leq \sqrt{n}$ but one must not assume that m is a constant. The proposed algorithm should have worst-case running time that is $o(n^2)$. He requests the assistance of the CS Department in dealing with this problem.

(a) Give an algorithm whose worst-case running time is $o(n^2)$ that solves this problem; an $\Omega(n^2)$ time algorithm will not get you any points.

(b) Does there exist an algorithm whose worst-case performance is $\Theta(n)$? Explain, i.e. show/prove that such an algorithm does not exist or give the algorithm and analyze its performance, if such an algorithm exists.

Example. If $X = \langle 2, 0, -2, -3, 1, 100, 290, 300, 4 \rangle$ and $Y = \langle -20, -19, -8, -200, 11, 1, 10, -7, 0 \rangle$ and $m = 3$ then the answer is $\langle 301, 310, 311 \rangle$. Note that 301 can result either as the sum of $290 + 11$ or $300 + 1$.

Problem 7. (33 POINTS)

You are given three sorted sequences of $\lg n$, n and n^2 keys respectively. Let these sequences be denoted by A, B, C respectively. You are asked to generate an array D that contains all the keys that are common to all three sets A, B, C , i.e. $d \in D$ if and only if $d \in A$ and $d \in B$ and $d \in C$. How fast can you find and generate D ? Explain by providing an efficient algorithm; analyze its worst-case running time.

This is the end of the exam.

Left blank intentionally