

Name:SAMPLE FINAL.....

ID Number:(5 points)

Exam Type:

...A...

Exam Number:

.....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: ... 8: ... 9: ... Total:

SOLVE **all the problems**

MARK ONE OF PROBLEMS 3-9 WITH THE WORD BONUS AND YOUR GRADE
FOR THAT PROBLEM WILL BE DOUBLED

THERE ARE 7 (SEVEN) PAGES THIS PAGE INCLUDED
Read and Sign the statement below at the end of the exam
Unsigned exams will be marked with grade 0 (zero).

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

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

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

NOTE: "Give an algorithm" or "Show how" means, present the algorithm, show that it works as claimed, and analyze its running time by providing an asymptotically tight bound. Rumblings or pseudocode without further description or proof of claims is neither 'give an algorithm' nor "show how". The algorithm universe, the set of algorithms introduced/discussed in class can be used without further discussion.

Problem 1. (64 POINTS)

Provide short answers to the following questions. (For running time provide as tight a bound as possible using asymptotic notation. The algorithm universe is the one introduced in class.)

- (1) What is the best-case running time of InsertionSort?
- (2) Is HeapSort stable?
- (3) Does MergeSort sort in-place?
- (4) Give the asymptotic solution of $T(n) = T(n/3) + n$.
- (5) Give the worst-case running time of RandomizedSelect.
- (6) Give the worst-case running time for building a MAX-HEAP with $t \lg t$ elements.
- (7) Name 2 dynamic programming algorithm.
- (8) Name 3 greedy algorithms that solve 3 different problems.

Problem 2. (54 POINTS)

For this problem input size is specified separately for each case and thus **Row-1, Column-1** indicates a problem size of n^2 keys. Graphs have n vertices and m edges and represented using an adjacency matrix; if needed a distance matrix is attached with positive distances. Find the worst-case running time of every (efficient) algorithm in **Column-1** by using the time bounds in **Column-2**; use only class-introduced algorithms. If multiple answers are possible, use the asymptotically tightest bound. The same answer in **Column-2** can be reused. Row 7 is filled for you; you should fill rows 1 through 6. If you see $\lg^t n$, it means $(\lg n)^t$.

Column-1	Answer	Column-2
1: BuildMaxHeap of n^2 keys		A. $O(n^3)$
2: Sorting n integers in $0..n-1$.		B. $O(n^4)$
3: QuickSort of n^2 keys		C. $O(n^2)$
4: Depth First Search		D. $O(n)$
5: InsertionSort on n^2 keys		E. $O(n \lg n)$
6: FloydWarshall		F. $O(n^2 \lg n)$
7: MergeSort of n keys	E	G. $O(n^2 \lg^2 n)$.

Problem 3. (30 POINTS)

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

$$T(n) = 2T(n/2) + 18n, \quad T(8) = 8.$$

Problem 4. (30 POINTS)

We use Dijkstra's algorithm to find the shortest paths from vertex s for the problem $(G = (V, E), d, s)$ where d contains positive values. We then double every edge distance i.e. $d'(u, v) = 2 * d(u, v)$. How does the solution for $(G = (V, E), d, s)$ relate to the solution for the problem $(G = (V, E), d', s)$? Explain briefly and justify your answer.

Problem 5. (30 POINTS)

We would like to find the **smallest**, **3rd-smallest**, **5th-smallest** and **7th-smallest** of n keys. Show how it can be done within $3n + o(n)$ comparisons.

Problem 6. (30 POINTS)

We have n keys in an arbitrary range. We would like to determine whether these keys contain duplicates or not. Thus if there are no duplicate keys (i.e. the keys are distinct) the algorithm says "NO", otherwise the algorithm says "YES". Give an efficient (time and space) algorithm for this problem. (The algorithm on input $\langle \text{Tom, Jerry, Mary, Mike, Jerry} \rangle$ would say "YES", and on $\langle \text{Tom, Jerry, Mary, Mike, Harry} \rangle$ would say "NO".)

Problem 7. (30 POINTS)

We define a Binary-Search-Tree (BST) with n nodes to be **nicely-balanced** if its height is $O(\lg n)$. We have two arbitrary (not necessarily balanced nor nicely balanced) BST trees T_1 and T_2 with n_1 and n_2 nodes, where $n = n_1 + n_2$. Assume all n nodes have distinct key values. We need to create a **nicely-balanced** BST T which contains all n keys. That is, T is the union of the key values of T_1 and T_2 and is nicely-balanced. Give an efficient algorithm for this task whose worst-case running time is faster than $O(n \lg n)$. Prove your claims.

Problem 8. (30 POINTS)

The Fibonacci sequence is given by the following recurrence $F_{n+1} = F_n + F_{n-1}$ for $n \geq 1$ and $F_0 = F_1 = 1$. Show how to compute F_n in $O(n)$ time and $\Theta(1)$ space.

Problem 9. (30 POINTS)

Social Networking in Facebook. Social networking is a graph problem. Vertices are persons and social connections become directed edges with edge (u, v) meaning that u considers v a friend (but this does not mean v calls u a friend; an edge (v, u) is needed for that). Let us have a social graph $G = (V, E)$ using the traditional terminology with $|V| = n$ and $|E| = m$.

Facebook uses proprietary technology to rank edges with a function L . Thus $L(x, y)$ which is a real number greater than 0 and less than 1, establishes the **likeness** of y by x . (Facebook's technology checks whether x accesses y 's page often, whether x posts messages on y 's space and so on.) User p is **"connected"** to user q if they have a sequence of connected/liked friends, i.e. there is a sequence $w_0 = p, w_1, \dots, w_{t-1}, w_t = q$, where w_0 considers w_1 a friend, and w_1 considers w_2 a friend and so on, and w_{t-1} considers w_t a friend. The **closeness** $C(p = w_0, w_1, \dots, w_{t-1}, q = w_t,)$ of this connection (path) between p and q can then be defined as the product of the likenesses of the edges from p to q along this connection path.

$$C(p = w_0, w_1, \dots, q = w_t) = \prod_{i=1}^t L(w_{i-1}, w_i).$$

For a given p and q we might have many connections between the two and each one of this gives rise to a different C . The largest value for C defines the **bondness** $B(p, q)$ of p and q (in that specific order). That is

$$B(p, q) = \max_{\text{over all possible connection paths from } p \text{ to } q} C(p, \dots, q)$$

We now assume that graph $G = (V, E)$ is enhanced with L i.e. $L(u, v)$ is known for each one of its edges i.e. $G = (V, E, L)$ is available. We would like to find for a given friend/vertex f , the bondness between f and every other vertex u i.e. $B(f, u)$. Give an algorithm for this problem, prove its correctness, and analyze its worst-case running time.

Hint. For $r = 0.1$ we have $\lg r = -3.32$ and $r' = 0.9$ we have $\lg r' = -0.152$.