

Name:

ID Number:

Exam Type:

.....

Exam Number:

.....

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

SOLVE ALL **8 (EIGHT)** 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

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

Problem 1. (40 POINTS)

Solve the following recurrence relations by providing asymptotically tight bounds for the first and an exact solution for the last case. If no boundary case is given (eg first case), the choice (of the constants) is yours. You may assume that $T(n)$ is positive and monotonically increasing, if you need to do so.

$$\begin{aligned} T(n) &= 4T(n/2) + n!. \\ \text{Solve exactly } T(n) &= 2T(n/2) + 3n, \quad T(8) = 8. \end{aligned}$$

Problem 2. (40 POINTS)

TRUE or FALSE? No justification required. Anything other than a clearly written TRUE or FALSE will be considered incorrect. Terms **stable-sort** and **in-place** have their traditional meaning in the context of sorting. Efficient is time-efficient.

1. Heap-sort is stable.
2. Count-sort is stable.
3. Merge-sort sorts in-place.
4. Count-sort is stable.
5. Worst-case running time of Dijkstra's algorithm for the single-source shortest-path problem for arbitrary graphs (adjacency-list and min-heap implementation, all weights positive) is $O((n + m) \lg n)$.
6. Chained matrix multiplication. If we have t^2 matrices we can determine the best schedule to multiply them using the dynamic programming algorithm presented in class in time $O(t^3)$.
7. Among radix-sort, insertion-sort, quick-sort, and heap-sort the most efficient way to stable-sort n keys in the range $0, \dots, n^2 - 1$ is radix-sort.
8. Among merge-sort, quick-sort, insertion-sort and count-sort the most efficient way to in-place and stably sort n keys in the range $0, \dots, n - 1$ is by using count-sort.

Problem 3. (40 POINTS)

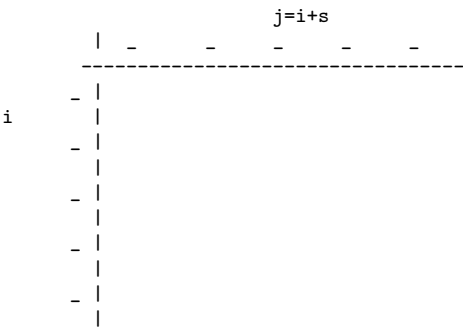
A sequence of n distinct values $A[0..n-1]$ is said to be **downup** if there is an index p with $0 \leq p < n$ such that the values of A decrease up to $A[p]$ and then increase for the remainder of the sequence. The index p of value $A[p]$ is the valley of the sequence. For example sequence 50, 10, 5, 2, 1, 20, 30 is downup with valley 4, since $A[5] = 60$ and the sequence decreases to 1 and then increases. The sequence 5, 1, 4, 3 is not downup. Design an $o(n)$ worst-case running time algorithm that when given as input an downup sequence, it finds its valley p . Analyze the performance of your proposed algorithm and show that it is $o(n)$ (this is a little-oh).

Problem 4. (40 POINTS)

Professor I.M.Nuts suggested the following work-around for Dijkstra's algorithm when the graph has negative-weight edges: (a) Find the most negative edge weight (i.e. find the minimum weight edge) and let that weight be W , (b) then add $W + 1$ to the weight of every edge of the graph. The resulting graph has only positive weights. Run Dijkstra's algorithm with source s on this modified graph. If for vertex v , quantity $D[v]$ is the cost of the shortest path to v in this modified graph, then report $D[v] - k(W + 1)$ as the cost of the shortest path from s to v in the original graph. Quantity k is the number of edges from s to v . This number can be found if we maintain a parent array (i.e. when $D[v]$ is modified because of $D[u]$ and edge (u, v) , we set $p[v] = u$). What is the time-overhead of this approach? Does it affect the asymptotic running time? Does this method work, i.e. does it indeed find the shortest path from s to any v in the original graph? Justify your answer.

Problem 5. (40 POINTS)

You are given 4 matrices M_1, M_2, M_3, M_4 and you are asked to determine the optimal schedule for the product $M_1 \times M_2 \times M_3 \times M_4$ that minimizes the number of operations (addition/multiplication) involved. The dimensions of the four matrices are respectively 100×50 , 50×200 , 200×50 , and 50×10 . What is the best (cheapest) schedule to multiply all the matrices together and compute $M_1 \times M_2 \times M_3 \times M_4$? What is the total cost for this schedule?



Problem 6. (40 POINTS)

NJIT decides to introduce its own coinage with three different types of coins: 1 cent, 5 cent, and 8 cents. We would like to know what is the minimum number of coins we can use if we pay for an item worth n cents. Give an efficient algorithm that if given n as input, it prints as output the minimum set of coins that has value exactly n . Analyze the time and space requirements of your algorithm. Prove its correctness. For example, you can pay an item worth 40 cents by giving five 8-cent coins; other alternatives is eight 5-cent coins, or forty 1-cent coins, or say four 8-cent, one 5-cent and three 1-cent coins.

Problem 7. (53 POINTS)

When we search in Google for “A B C D” this is equivalent to searching for “A AND B AND C AND D”, i.e. searching for all documents that contain all four keywords A, B, C and D . We ask you to design an algorithm that does so efficiently. The first thing a search engine does is to generate a hit-list for each keyword. A hit-list is a sorted sequence of document-identifiers (doc-id’s) that contain the keyword in question. Let the hit-lists be in the form of sorted arrays $A[1..a], B[1..b], C[1..c], D[1..d]$ of doc-id’s for the four keywords respectively. Let the lengths of the arrays A, B, C and D be respectively a, b, c, d , where $a = n^3$, $b = n \lg n$, $c = n^2$ and $d = n$ for some parameter n . Give an efficient algorithm that generates the hit-list of doc-ids in which all four keywords A, B, C, D appear; this output hit-list should also be sorted by doc-id. How big is the output hit-list ? Express your answer in asymptotic notation in terms of n with a bound that is as asymptotically tight as possible. Express the running time of your proposed efficient algorithm in terms of n using asymptotic notation (as tight as possible).

Problem 8. (40 POINTS)

Google ranks web-pages with an arbitrary numerical value. One cannot make any assumptions about the range of the rank. Suppose that a query returns n pages (n is part of the input) and information about rank is stored for the i -th page in $A[i]$. We are interested in determining the top 20%, the bottom 40% and the middle 40% of all pages. For that we need to determine two values m and M , the values for which 40% of the pages have ranks less than m , 20% more than M and those whose rank is between m and M . Give an efficient algorithm (whose performance cannot be improved upon) that solves this problem, i.e. it finds m , and M given n and A .

This is the end of the exam.
Left blank intentionally