

CSE 830: Design and Theory of Algorithms

Sample Exam I Questions

I. Problem Complexity and Asymptotic (Big-Oh) Notation

1. For each of the following, compute how large of a problem instance you need before algorithm A is faster than algorithm B? How much time do the algorithms take on that instance?

- (a) Algorithm A takes n^2 days to solve a problem of size n . Algorithm B takes n^3 seconds on the same problem.
- (b) Algorithm A takes n^2 days to solve a problem of size n . Algorithm B takes 2^n seconds on the same problem.

2. Prove or find a counter-example: if $f(n) = O(F(n))$ and $g(n) = O(G(n))$ then $f(n) + g(n) = O(F(n) + G(n))$.

3. Insertion sort takes one second to sort 1000 items (worst-case time) on a particular computer. Estimate the amount of time it would take to sort 100,000 items. If Quicksort took 1 second for 1000 items, how long would you expect it to take for 1,000,000?

4. List the functions below from the lowest to the highest order. If any two or more are of the same order, indicate which.

n	$n \log n$	2^n	$\ln n$
$n - n^3 + 7n^5$	$\log n$	$\text{sqrt}(n)$	e^n
$n^2 + \log n$	n^2	2^{n-1}	$\log \log n$
n^3	$(\log n)^2$	$n!$	$n^{1+\epsilon}$ (where $0 < \epsilon < 1$)

II. Data Structures and Data Types

5. A deque (Double Ended QUEUE) is a container type that is more flexible than either a stack or a queue -- you can insert or remove data from either end. If you needed to implement a deque using only two of the simpler container types, which would be fastest: 2 stacks, 2 queues, or 1 stack and 1 queue? Describe your implementation and give the time complexity for InsertFront, RemoveFront, InsertTail, and RemoveTail.

6. List the three rules that define the structure of a heap.

7. Show how to implement a standard first-in, first-out queue using only a heap as the underlying data structure. What is the run time for inserting and removing items from such a queue?

8. Draw a heap for the key values 2, 14, 10, 23, 8, 3, 12, 20, 1 and 17. Draw what this heap would look like after an "Extract-Max" is run on it.

9. Given a hash table with 100 bins and a good hashing function, what is the probability that the first two entries end up in different bins? If there were three entries, what is the probability that they all end up in different bins?

III. Recursion

10. Convert the following into *simple* and *efficient* iterative (non-recursive) functions:

- (a) $f(n)$
if $(n == 1)$ return 1
else return $f(n-1) + 1$
- (b) $f(n)$
if $(n == 1)$ return 1
else return $f(n-1) * 2$
- (c) $f(n)$
if $(n == 1)$ return 1
else return $f(\text{floor}(n/2)) + 1$
- (d) $f(n)$
if $(n == 1)$ return 1
else return $f(\text{floor}(n/2)) * n$

11. Find the rate of growth of the following recurrence relations using recursion trees:

- (a) $T(n) = T(n/3) + T(n/6) + n$
- (b) $T(n) = T(\alpha n) + T((1 - \alpha)n) + n$, where $0 < \alpha < 1$.

12. Professor Q.T. Beeny became a bit over-zealous with the "Divide and Conquer" approach to designing algorithms, and came up with the following function. It takes an array A of integers and finds the maximal value between the i^{th} and j^{th} entries in the array. Assume that the floor is taken for the result during any integer division.

```
FindMax( A, i, j )
  if (i == j) return A[i]
  else
    k = (i + j) / 2
    return max( FindMax(A, i, k), FindMax(A, k+1, j) )
```

- (a) What is the worst-case time complexity of this algorithm?
- (b) Prove it correct or find a counter-example.

13. Professor E.Z. Dozit came up with his own method of using recursion to find the maximal value in an array. His method finds the maximal value of the first n elements of the integer array A . Again, assume the floor is taken for the result of any integer division. Also note that this pseudo-code assumes that an array of size n is indexed from 1 to n .

```
FindMax( A, n )
  if (n == 1) return A[1]
  else
    k = n/2
    for (i = 1 to k)
      A[i] = max( A[i], A[i+k] )
    return FindMax(A, k)
```

- (a) What is the worst-case time complexity of this new algorithm?
- (b) Prove it correct or find a counter-example.

14. Consider a binary tree where each node has exactly *zero* or *two* children.

- (a) How does the number of internal nodes compare with the number of leaves?
- (b) Prove your answer to (a) using mathematical induction.

IV. Sorting Methods and Applications

15. Re-construct the proof from class that the fastest possible comparison-base sorting algorithm on unknown data is $\Omega(n \log n)$ in the worst case.

16. Prove that you can never *search* unknown data in faster than $\Omega(\log n)$ time using a comparison-based search algorithm.

17. Choosing a random pivot point improves quicksort by removing the worst case due to bad data. What effect would happen to Insertion Sort if we chose a random element to insert rather than the next one in the input sequence?

18. Suppose an array consists of n elements, each of which is red, white, or blue. We seek to sort the elements so that all the reds come before all of the whites, which come before all the blues. The only operations we are permitted to use on the array are:

- Examine(i) - report the color of the i th element in the array.
- Swap(i, j) - Swap the i th element with the j th element in the array.

- (a) Find a correct and efficient algorithm for red-white-blue sorting. [*Hint*: There is a linear time solution.]
- (b) Explain how, if there are k colors, the above algorithm can be modified to take $O(n \log k)$ time.

V. Problem Specific Data Structures and Algorithms

19. Devise an algorithm for finding the k smallest elements of an unsorted set of n integers in $O(n \log k)$ time. Is this always worse than the $O(n + k \log n)$ that you showed in your homework?

20. Design a data structure for a container that supports the following operations:

- Insert(x) Insert item x into the container.
- Delete(k) Delete the k^{th} smallest element from the container.
- Search(x) Return true if x is in the container, else return false.

All operations must take $O(\log n)$ time on an n -element set.

21. Suppose that we are given a sequence of n values: x_1, x_2, \dots, x_n and seek to quickly answer repeated queries of the form: given i and j , find the smallest value in x_i, \dots, x_j .

- (a) Design a data structure that uses $O(n^2)$ space and answers queries in $O(1)$ time.
- (b) Design a data structure that uses $O(n)$ space and answers queries in $O(\log n)$ time. For partial credit, your data structure can use $O(n \log n)$ space, but must have $O(\log n)$ query time.

Note: You may use as much time as you need to initialize your structures and don't need to worry about ever inserting or removing values.

VI. Mathematical Exercises

22. Estimate the number of words in your textbook "Introduction to Algorithms".

23. Estimate the number of individual class sessions you've sat through in your lifetime.

24. Consider the numerical 20 questions game. In this game, player 1 thinks of a number in the range 1 to n . Player 2 has to figure out this number by asking the fewest number of true/false questions. Assume that nobody cheats.

- (a) What is an optimal strategy if n is known?
- (b) What is a good strategy if n is not known, but you are limited to k questions?

25. You have n coins, all but one of which are gold. The one non-golden coin weighs 10 milligrams more than the rest, but is otherwise indistinguishable.

- (a) If you have a balance scale (where you can put an arbitrary number of coins on either side and it will tell you which pile weighs more or if they are the same weight), what is the fewest number of weighings that you need to do to discover the fake?

- (b) If you have a digital scale that will inform you of the exact weight of anything placed on it, how many weighings will it take you to discover the fake coin? Is this better or worse than the balance scale?
- (c) If you now have n *bags* of coins (each bag containing at least n coins in it) and you discover an entire bag is counterfeit, which type of scale would now require fewer weighings? (you can take any number of coins out of each bag to weigh at once).

VII. Optimization Problems

For these problems, be prepared to give an intuitive explanation for your algorithm. If you use dynamic programming, be ready to provide the chart of partial solutions that you store.

26. Index-Element Matching. Suppose that you are given a sorted sequence of n distinct integers $\{a_1, a_2, \dots, a_n\}$. Give an $O(\log n)$ algorithm to determine whether there exists an index i such that $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$. In $\{2, 3, 4, 5, 6, 7\}$, there is no such i .

27. The Currency Problem. Consider a country who has coins with denominations of $\{d_1, d_2, \dots, d_k\}$ units (you may assume that d_1 is 1 unit). Give an efficient algorithm to determine the minimum number of coins required to make change for n units of currency. Analyze the running time for this algorithm.

28. The Currency Problem II. Formulate an efficient algorithm to determine the total number of possible ways to make change for n units of currency given coins of denominations of $\{d_1, d_2, \dots, d_k\}$ units (you may assume that d_1 is 1 unit). Analyze the running time for this algorithm.

29. Longest Common Substrings. Given two strings of lengths m and n , give an efficient algorithm to determine the longest common sub-string. For example, the strings “asymptotic” and “unsympathetic” have the longest common sub-string of “symp”. Analyze the time complexity for this algorithm.

30. Shuffled Strings. The string Z is a *shuffle* of strings X and Y if Z contains all the letters of X and Y in their original order but interspersed. For example, “chcohciolpaste” is a shuffle of “chocolate” and “chips”. Give an efficient algorithm to determine if a string Z is a shuffle of strings X and Y . Analyze the time complexity for this algorithm.

31. The Knapsack Problem. A greedy algorithm will often not produce the optimal answer for the most general form of the Knapsack Problem. Give examples of where the following greedy algorithms would fail:

- a. Always choose the most valuable items first.
- b. Always choose the smallest items first.
- c. Always choose the items of maximum value/size first.

32. String Alignment. Construct the dynamic programming table that would be used to calculate the minimum number of insertions, deletions, and mutations it would take to go from

the word “bear” to the word “flea” using the algorithm explained in class. What is the optimal alignment between these words?

33. Recursion v.s. Dynamic Programming. Given the equation $T(a, b) = \min(T(a-1, b) ; T(a-1, b-1) ; T(a, b-1)) + ab$, with boundary conditions of $T(a, 0) = a$ and $T(0, b) = b$, compare the running time of a recursive implementation of this equation to that of an efficient dynamic programming implementation.

34. The bookshelf problem. In one of your homework problems, you constructed an algorithm to find the minimum height of a bookshelf where each book b_i has a thickness (t_i) and a height (h_i), and all of the shelves have a maximum width W and a height equal to its tallest book. Given the books:

i	t_i	h_i
1	1	1
2	2	2
3	1	5
4	1	4
5	2	3
6	1	1

Determine the placement of these books when $W=4$ that would result in the minimal height. What height would the greedy algorithm have produced?

35. Text Justification. You are given a body of text with n words in it, and you want to print out that text to minimize the number of large gaps of whitespace wasted at the end of lines. Assume that word i is c_i characters long, and a full line can hold up to L characters in it. If k characters are unused in any line, than that line carries a cost of k^2 . Thus many small gaps will typically be better than one big one. Design an algorithm to minimize the total whitespace cost of printing this body of text. Analyze the worst-case time complexity of this algorithm.