

Final Exam Review

The CS231 final is a self-scheduled final exam held during the normal final exam period. The exam is open book; you may refer to the textbook, class handouts, your notes, and whatever additional materials you find useful. By the Honor Code, you are not allowed to talk to anyone about the details of the exam before or after taking it, until the final examination period is over.

The exam has **7** problems totalling 100 points. Each problem has many parts, most of which are very short; point values for the parts range between 1 and 6 points. Since there are *lots* of little parts, you need to be careful allocating your time in order to finish the exam. In particular, if you spend a lot of time looking up techniques/answers in your book/notes, you will *not* have enough time to finish the exam. It is best to go into this exam prepared to answer lots of questions without having to look them up, and to save the book/notes for the small portion you can't answer off the top of your head. In particular, you should be very familiar with general techniques for designing and analyzing algorithms (e.g., asymptotic notation, recurrences, probability, augmentation, memoization, greediness, amortization) as well as details of particular algorithms we have studied (e.g., how do they work? what is their best worst-case running time?). **You may find it helpful to prepare (one or more) cheat sheet(s) for quick reference.**

To help you be mentally prepared for the exam, I am including the actual cover sheet of this term's final exam as the last page of this handout. Read it carefully so that you don't have to spend any time reading it during the exam.

The final exam is comprehensive, covering material the entire course. Below is a list of topics covered in CS231 that are fair game on the final exam (not all of these are actually covered).

Mathematical Tools:

- asymptotic notation (o , O , Θ , Ω , ω)
- recurrence equations
- simple summations
- basic combinatorics
- basic probability (event spaces, expected values)
- proof techniques: induction, loop invariants, optimality of greedy algorithms
- amortized analysis

Sorting:

- Comparison-based sorting
 - selection sort
 - insertion sort
 - merge sort
 - quick sort
 - tree sort
 - heap sort
 - decision tree model (implies $O(n \cdot \lg(n))$ best worst-case time)
- Linear sorting (can only be used in special situations)
 - counting sort
 - radix sort
 - general bucket sort

Order Statistics (find the k th smallest element of a collection)

- naive method (first sort, then find k th element)
- quicksort-like partitioning
- median-of-median-of- c ($c = 5$ is CLRS's/CLR's `Select`).

Dynamic Sets (operations = search, insert, delete, minimum, maximum, predecessor, successor)

- array implementations (sorted/unordered)
- list implementations (sorted/unordered; singly-linked/doubly-linked)
- binary search trees
- AVL trees
- red-black trees
- augmenting data structures

Heaps

- complete heaps
- leftist heaps

Dynamic Programming/Memoization

- exponentiation
- Fibonacci numbers
- Pascal's triangle
- longest common subsequence
- matrix chain multiplication

Greediness

- scheduling problems
- knapsack problems (0/1, fractional)
- coin-changing (arbitrary denominations/restricted denominations)
- Huffman coding trees
- single source shortest path (BFS, Dijkstra)

Compression

- Uniform-length coding
- Variable-length coding: character-based and word-based Huffman coding
- Adaptive dictionary methods: LZ77, LZ78

Graphs

- Graph terminology: undirected vs. directed, weighted graphs, paths, cycles, trees, DAGs
- depth-first search: edge classification (tree, back, forward, cross), discovery/finish times
- topological sorting
- connected components, strongly connected components
- single-source shortest path (breadth-first search, Dijkstra's algorithm)

The rest of this handout contains problems that are intended to help you review material for the final exam. They are not necessarily indicative of the kinds of questions that will be asked on the exam (i.e., some review questions are more difficult/time consuming than what would be on an exam.) They also do not cover all of the above topics.

Problem 1: Asymptotics

Consider the set $S = \{a, b, c, d, e\}$ of five functions a through e , which are classified as follows:

- $a \in \Theta(1)$
- $b \in \Theta(\lg(n))$
- $c \in \Theta(n)$
- $d \in \Theta(n \cdot \lg(n))$
- $e \in \Theta(n^2)$

In the following table, each row is labeled by an asymptotic notation that designates a set of functions R . For each row, indicate the result of the intersection $R \cap S$, i.e., the set of all functions in S that are also members of R .

R	$R \cap S$
$\Theta(n \cdot \lg(n))$	
$O(n)$	
$o(\lg(n))$	
$\Omega(1)$	
$\omega(n^2)$	

Problem 2: Asymptotics

For each entry (*row*, *col*) in the following table, determine whether membership in the set labelled *row* implies membership in the column labelled *col*. Fill in each entry of the table with one of the following three answers:

- o *Yes*: Every member of set *row* is a member of set *col*.
- o *No*: No member of set *row* is a member of set *col*.
- o *Maybe*: Some, but not all, members of set *row* are members of set *col*.

	$o(n \cdot \lg(n))$	$O(n \cdot \lg(n))$	$\Theta(n \cdot \lg(n))$	$\Omega(n \cdot \lg(n))$	$\omega(n \cdot \lg(n))$
$o(n)$					
$O(n)$					
$\Theta(n)$					
$\Omega(n)$					
$\omega(n)$					
$o(n^2)$					
$O(n^2)$					
$\Theta(n^2)$					
$\Omega(n^2)$					
$\omega(n^2)$					

Problem 3: Recurrences

Use Θ notation to describe solutions to each of the following recurrence equations. You may assume in all cases that $T(n) = 0$ for $n \leq 1$.

- a. $T(n) = T(n - 3) + 2$
- b. $T(n) = T(n - 7) + 3n$
- c. $T(n) = T(n/3) + 5$
- d. $T(n) = T(n/5) + 2n$
- e. $T(n) = 2T(n/2) + 5$
- f. $T(n) = 2T(n/2) + 3n$

Problem 4: Recurrences

Below are four list reversal functions written in Haskell_{eager}. For each function, (1) write a recurrence equation that describes the running time of the function, and (2) express the solution to the recurrence equation in Θ notation. Recall that `++` takes time proportional to the size of its left operand.

```
reverse1 [] = []
reverse1 (x:xs) = (reverse1 xs) ++ [x]

reverse2 xs = revTail xs []
  where revTail [] zs = zs
        revTail (y:ys) zs = revTail ys (y:zs)

reverse3 [] = []
reverse3 [x] = [x]
reverse3 xs = (reverse3 rights) ++ (reverse3 lefts)
  where lefts = take half xs -- (take n xs) returns the first n
                             -- elements of xs in Theta(n) time
        rights = drop half xs -- (drop n xs) returns all but the first n
                               -- elements of xs in Theta(n) time
        half = div (length xs) 2

reverse4 [] = []
reverse4 [x] = [x]
reverse4 (x:xs) = h : (reverse4 (x : (reverse4 t)))
  where h:t = reverse4 xs
```

Problem 5: Probability

Suppose you are given a black-box procedure `Roll()` that simulates rolling a fair six-sided die. That is, `Roll()` returns an integer in the range $[1..6]$, with each integer being equally likely.

For each integer k in the range $[2..7]$, answer the following questions:

- a. Using `Roll()`, write pseudocode for a function `Rollk()` that returns an integer in the range $[1..k]$, with each integer being equally likely.
- b. What is the expected number of times that `Roll()` is called for a single top-level call of `Rollk()`?

Problem 6: Running Times

Below is a table containing brief descriptions of various data structures that represent a set of integers S . For each such data structure, think of the best algorithm for solving each of the following three problems: (1) finding the maximum of the collection; (2) finding and deleting the maximum element of the collection; (3) determining if a given integer is a member of the collection. Indicate the asymptotic worst-case running time (using Θ notation) of these algorithm in the second through fourth columns of the table. You should phrase your answer in terms of the number n that appears in each description. Unless the description indicates otherwise, you should not make any assumptions about the arrangement of elements in the data structure.

Representation of S	Best worst-case running time of $\text{Max}(S)$	Best worst-case running time of $\text{Delete-Max}(S)$	Best worst-case running time of $\text{Member}(x, S)$
An unordered array of n integers.			
An array of n integers ordered from smallest to largest.			
An array of n elements ordered from largest to smallest.			
An $n \times n$ two-dimensional array of unordered elements.			
An unordered linked list of n elements.			
A linked list of n elements ordered from smallest to largest.			
A linked list of n elements ordered from largest to smallest.			
A binary tree with n elements.			
A binary tree of height n .			
A binary search tree with n elements.			
A binary search tree of height n .			
An AVL tree with n elements.			
An red-black tree with n elements.			
A complete heap of n elements in which a has a higher priority than b if $a > b$.			
A complete heap of n elements in which a has a higher priority than b if $a < b$.			
A linked list of n balanced binary search trees, each with n elements.			

Problem 7: Recurrences, Dynamic Programming

Given a numeric array A , the partial sum array $\text{PSA}(A)$ is an array P that has the same length as A in which $P[i]$ is the sum of all the elements in the array segment $A[1..i]$. For example, the following figure show an array A and its partial sum array $P = \text{PSA}(A)$:

A	2	5	3	6	1	4
P	2	7	10	16	17	21

Here is one algorithm that correctly computes the partial sum array P for an array A :

```
PSA1(A)
  P ← new array[1..length[A]] ▷ Make a new array for the result.
  for i ← 1 to length[A] do
    P[i] ← Segment-Sum(A, 1, i)
  return P

Segment-Sum(A, lo, hi)
  if lo > hi
  then return 0
  else return A[hi] + Segment-Sum(A, lo, hi - 1)
```

- a. Write a recurrence equation defining $T(n)$ that expresses the worst-case running time of PSA1 on an array of length n .
- b. Solve the recurrence equation from part a to obtain a asymptotic worst-case time bound using Θ .
- c. For every j , $1 \leq j \leq \text{length}[A]$, $\text{PSA1}(A)$ calculates $\text{Segment-Sum}(A, 1, j)$ j times. This is wasteful. Using dynamic programming techniques, develop a $\Theta(n)$ algorithm $\text{PSA2}(A)$ for computing the partial sum array of A . Write the pseudocode for PSA2 and argue that it runs in worst-case time $\Theta(n)$. *Hint:* Use the resulting array P as the "table".

Problem 8: Sorting

a. Assume that you want to sort an array of n integers (not necessarily distinct) taken from the range $[1..n]$. For each of the following sorting algorithms, indicate the asymptotic (1) worst-case running time (2) best-case running time and (3) average-case running time.

Sorting Algorithm	Worst-Case	Best-Case	Average-Case
selection sort			
insertion sort			
merge sort			
quick sort			
tree sort			
heap sort			
counting sort			
radix sort			
bucket sort			

b. Which of the above methods are not viable if the array may contain any numbers in the range $[1..n]$, not just integers?

Problem 9: Sorting, Priority Queues, Order Statistics (An adaptation of *CLR* Problem 10-1.) Given a set of n numbers, we wish to find the i largest in sorted order using a comparison-based algorithm.

a. Find the algorithm that implements each of the following methods with the best asymptotic worst-case running time, and analyze the running times of the methods in terms of n and i .

1. Sort the numbers and list the i largest.
2. Build a priority queue from the numbers and call `Delete-Max` i times.
3. Use an order-statistic algorithm to find the i th largest number, partition, and sort the i largest numbers.

b. Can linear sorting techniques be used to reduce any of the running times from part (a)?

c. Suppose that the problem statement is weakened so that the goal is to find the i largest elements, but in any order (not necessarily sorted). How would this change the running times from part (a)?

Problem 10: Binary Trees

- a. For any n , are there trees with n nodes that are both complete heaps and leftist heaps?
- b. For any n , are there trees with n nodes that are both binary search trees and complete heaps?
- c. For any n , are there trees with n nodes that are both binary search trees and leftist heaps?
- d. For each of the following types of trees, indicate the (1) minimum possible height and (2) maximum possible height as a function of n , the number of nodes in the tree. (Do *not* use asymptotic notation!)
 - o binary tree
 - o complete heap
 - o leftist heap
 - o AVL tree
 - o red-black tree
- e. In red-black tree insertion, explain under what conditions the black-height of a tree can increase upon insertion.
- f. In red black tree deletion, explain under what conditions the black-height of a tree can decrease upon deletion.
- g. In AVL trees, the height of a tree before and after rotation after insertion is the same. Explain how the height of a tree can increase upon insertion.
- h. In AVL trees, under what conditions does the height of a tree decrease upon deletion?

Problem 11: Heaps, Amortization

- a. Bud Lojack claims he can delete the n elements of a complete heap in sorted order from high to low priority in in $O(n)$ time. Prove that Bud is lying. *Hint 1:* How long does it take to build a heap with n elements? *Hint 2:* Remember Ima Fleik!
- b. Under the standard analysis of complete heaps, both **Insert** and **Delete-Max** have $\Theta(\lg(n))$ worst-case running times. Give an amortized analysis of complete heaps in which **Insert** has $\Theta(\lg(n))$ worst-case amortized running time but **Delete-Max** has $\Theta(1)$ worst-case running time.
- c. Part (a) shows that it is impossible to delete the n elements of a heap in $\Theta(n)$ time. Yet, part (b) suggests that n **Delete-Max** operations can be performed in $\Theta(n)$ time. Reconcile the seeming contradiction between parts (a) and (b).

Problem 12: Greediness (*CLR* Exercise 17.2-3)

Suppose that in a 0-1 knapsack problem (i.e. you either take an entire item or not), the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

Problem 13: Compression

- a. What is the uncompressed string S corresponding to the following LZ77 encoding?

$[(0,0,S), (0,0,H), (0,0,A), (0,0,_), (0,0,N), (3,10,!)]$

- b. How many bits does it take to encode S using the minimal fixed-length character encoding? Do *not* include the bits needed to represent the table mapping each character to its fixed-length encoding.
- c. How many bits does it take to encode S using a character-based Huffman encoding? Do *not* include the bits needed to represent the tree/table mapping each character to its Huffman encoding.
- d. What is the LZ78 encoding of S ?

Problem 14: Compression

- a. Consider the following claim:

Excluding the space needed to encode the tree/table associating characters with encodings, a message compressed with Huffman code is no longer than a uniform-length encoding of the message.

Is the claim true or false? Explain your answer.

- b. Standard ASCII is a uniform-length 8-bit encoding. We know from Problem Set 7 Problem 1 that every compression algorithm that shortens some messages must lengthen others. Yet the claim in (a) implies that Huffman coding can only shorten a message, never lengthen it. Reconcile these two seemingly contradictory claims.

Problem 15: Compression Ann has a small Java program `Prog.java` that generates a huge output file `Prog.out` when applied to a small input file `Prog.in`. Bob, who lives far away from Ann, needs a copy of `Prog.out` so that he can analyze it as part of his research. Briefly describe a very simple compression scheme that allows Ann to send Bob a highly compressed version of `Prog.out`.

Problem 16: Graphs Draw a weighted, directed, connected graph G with three vertices A , B , and C such that depth-first search starting at A , breadth-first search starting at A , and Dijkstra's single-source shortest path algorithm starting at A all induce different trees. For depth-first search and breadth-first search, assume that vertices and adjacency lists are all in alphabetical order.

Problem 17: Graphs

- a. Draw a single directed graph G that has all of the following features:
 - o 5 vertices = $\{A, B, C, D, E\}$
 - o 6 edges
 - o 0 self edges
 - o exactly 1 forward edge and 1 cross edge (assuming a depth-first search in which vertices are explored alphabetically).
 - o 3 strongly connected components
 - o 2 connected components (for the undirected graph \overline{G} corresponding to G)
- b. Solve the same problem as (a) where “6 edges” is replaced by “7 edges”
- c. Solve the same problem as (a) where “6 edges” is replaced by “8 edges”
- d. Can you perform a topological sort on any of your graphs from parts (a), (b), or (c)? Explain.

Problem 18: Graphs

- a. For each of the following numbers n , draw a three node DAG such that the DAG has exactly n distinct topological sorts.
 - o a. 1
 - o b. 2
 - o c. 3
 - o d. 6
- b. Argue that any DAG with three nodes must have a number of distinct topological sorts that is one of the four numbers given in part (a)
- c. What are the possible numbers of topological sorts that four-vertex DAGs can have?

CS231 FINAL EXAM
Fall Semester 2001

YOUR NAME: _____

This exam has **7** problems. Each problem has several parts. The number of points for each problem and part is shown in square brackets next to the problem or part. There are 100 total points on the exam. **The last problem – worth 28 points and covering 4 pages – consists of many smaller problems covering a wide range of topics.**

Write all your answers on the exam itself. Whenever possible, show your work so that you have a chance to receive partial credit even if the final answer is incorrect. You may use the blue book for your work; there is no need to copy the work (except the final answer) to the exam itself.

The exam is open book. You may refer to the textbook, class handouts, your notes, and whatever additional materials would be useful. By the Honor Code, you are not allowed to talk to anyone about the details of the exam before or after taking it, until the final examination period is over.

Please keep in mind the following tips:

- *Briefly skim the entire exam before starting any problem.*
- *Work first on the problems on which you feel most confident.*
- *Try to do something on every problem so that you can potentially receive partial credit. A guess is better than no answer at all!*
- *Allocate your time carefully.* If you are taking too long on a problem, wrap it up and move on.
- *If you finish early, go back and check your answers.*

GOOD SKILL!

Problem	Topic	Points	Score
1	Asymptotics	12	
2	Running Times	18	
3	Recurrences	12	
4	Heaps	10	
5	Graphs	10	
6	Compression	10	
7	Potpourri	28	
Total		100	