

# 600.226: Data Structures

## Final Exam

Peter H. Fröhlich  
phf@cs.jhu.edu

December 17, 2005  
**Time:** 40 Minutes

**Start here:** Please fill in the following important information using a **permanent pen** before you do **anything** else! Your exam will **not** be graded if you use a pencil or erasable ink on this page.

**Name (print):** \_\_\_\_\_

**Login (print):** \_\_\_\_\_

**Ethics Pledge:** With your signature you **certify** the information above and you also **affirm** the following:  
*“I agree to complete this exam without unauthorized assistance from any person, materials, or device.”*

**Signature:** \_\_\_\_\_

**Date:** \_\_\_\_\_

**Instructions:** Please read these instructions carefully before you start. **Switch off** your phones, pagers, and other noisy gadgets! You are **not** allowed to have anything but a pen (pencil, eraser) and this exam on your desk. You are **not** allowed to talk to anyone during the exam. If you have a question, please raise your hand **quietly**. You must **remain seated quietly** until all exams have been collected. Remember that you can **not** claim grading errors if you do not use a **permanent** pen for your answers.

**Do not open before you are told to do so!**

**You got \_\_\_\_\_ out of 40 points.**

## 1 Binary Warmup

(10 points)

For each of the following statements, determine whether it is either **true** or **false**. (1 point each)

1. For arrays, the transpose heuristic requires less data movement than the move-to-front heuristic.
2. Object-oriented polymorphism means that the sending object decides which method to invoke.
3. Experimental analysis compares concrete implementations of the same abstract algorithm.
4. In JAVA the concepts of `interface` and `abstract class` are interchangeable.
5. On average Bubble Sort performs fewer assignments than Selection Sort.
6. The minimum spanning tree for a graph of  $n$  nodes has  $n - 1$  edges.
7. Binary search trees are more suitable than binary heaps for HUFFMAN compression.
8. Bounded stacks—which can become full—should be a subtype of unbounded stacks.
9. Garbage collection by reference counting increases the overhead for reference assignments.
10. Algebraic specifications of abstract data types describe syntactic as well as semantic properties.

## 2 Tough Choices

(8 points)

For each of the following questions, circle **one** answer out of the choices given. (2 points each)

1. You need an ORDEREDSET implementation. Your application requires extracting the minimum and maximum **most frequently**; it also needs to check membership **a lot**; but it **rarely** inserts or removes elements. Which of the following data structures seems **most** appropriate?
  - (a) Double-ended heap (“Deap”) on an expandable, cyclic array.
  - (b) Sorted, cyclic linked list.
  - (c) Sorted, expandable, cyclic array.
  - (d) Balanced binary search tree (e.g. red-black tree).
  - (e) None of the above.
2. Consider the “order” of complexity classes from “fastest” to “slowest” in the intuitive sense. Which of the following seems most appropriate ( $A < B$  means that if  $f \in A$  holds then  $f \in B$  also holds)?
  - (a)  $O(n) < O(1) < O(\log n) < O(n^2)$
  - (b)  $O(1) < O(n) < O(\log n) < O(n^2)$
  - (c)  $O(1) < O(\log n) < O(n \log n) < O(n^2)$
  - (d)  $O(1) < O(\log n) < O(n^2) < O(n \log n)$
  - (e) None of the above.
3. You need a DIGRAPH implementation for large, sparse graphs. Your application checks for edges between nodes **most frequently**; it also needs to find all nodes connected to a given node **a lot**; but it **rarely** inserts or removes anything. Which of the following data structures seems **most** appropriate?
  - (a) Linked lists used as adjacency lists.
  - (b) Nested arrays used as adjacency matrix.
  - (c) Nested hash-tables used as adjacency matrix.
  - (d) All of the above work equally well.
  - (e) None of the above.
4. Consider the three “simple” sorting algorithms for arrays: Bubble Sort, Selection Sort, and Insertion Sort. You start with the array [3,4,7,1,2]. After **three** iterations of the **outer** loop, you have the array [1,2,3,7,4]. Which algorithm are you running?
  - (a) InsertionSort
  - (b) SelectionSort
  - (c) BubbleSort
  - (d) All of the above produce that array after three iterations.
  - (e) None of the above.

### 3 Short Answer

(8 points)

For each of the following questions, answer in **one to three** sentences, the shorter the better. (2 points each)

1. You want to add an `Iterator<T>` with the usual operations (i.e. `get`, `put`, `valid`, `next`, `previous`) to an existing `Vector<T>` (aka `ArrayList<T>`) implementation. Describe briefly how you would implement the iterator.
2. You need to implement a `Queue` interface but all you are allowed to use is a `Stack` implementation. **All three operations should have constant amortized time complexity!** Describe briefly how you would implement `enqueue`, `dequeue`, and `front` in terms of `Stack` operations and instances; **include a basic “cyber dollar” argument.**
3. You have a binary search tree  $T$ . Give a pseudo-code implementation of `PORP( $T$ ,  $f$ ,  $t$ )`, an operation that prints all keys  $k$  in  $T$  for which  $f \leq k \leq t$  holds in **post-order**.
4. Give a **brief** account of how AVL trees work. You should focus on **basic** properties and **insertion**; you do **not** have to give details about rotations, but you **should** motivate them.

## 4 Fun with (2,3) Trees

(5 points)

Consider a (2,3) tree, i.e. a multiway search tree in which each internal node has either 2 or 3 children. Starting with an empty tree, sketch the evolution of the (2,3) tree as insertions of the keys 1, 3, 2, 7, and 8 are performed, in that order. Be sure to show the “before” and “after” in cases where the structure of the (2,3) tree is violated temporarily.

## 5 Fixing Points

(9 points)

Let  $A$  be an array of integers. A **fixed point** of  $A$  is an integer  $i$  for which  $A[i] == i$  holds. For example, the array  $A = [7, -4, 2, 1, 6]$  has  $i=2$  as a fixed point.

1. Without further restrictions on  $A$ , what can you say about  $A$ 's fixed points? How many are there? What range can their values be in? Anything else? (2 points)
2. Give a simple pseudo-code algorithm for finding a fixed point of a given array  $A$ . What is the asymptotic time complexity of your algorithm? (2 points)
3. Assume that  $A$  is **sorted** in ascending order. Give an **efficient** pseudo-code algorithm for finding a fixed point that takes advantage of this fact. This should be **significantly** faster than your previous algorithm. What is the asymptotic time complexity of your algorithm? (5 points)

This page is intentionally **mostly** blank in case you run out of space elsewhere. If you ended up here early, please go over **everything** again and remain seated **quietly**! Make sure that the title page is filled out correctly and in **permanent** pen. Maybe you want to "rewrite" your **answers** in permanent pen as well?