Exam, June 6, 2011

Jesper Larsson, ITU

Instructions

What to bring. You can bring any written aid you want. This includes the course book and a dictionary. In fact, these two things are the only aids that make sense, so I recommend you bring them and only them. But if you want to bring other books, notes, print-out of code, old exams, or today's newspaper you can do so. (It won't help.)
You can't bring electronic aids (such as a laptop) or communication devices (such as a mobile phone). If you really want, you can bring an old-fashioned pocket calculator (not one that solves recurrence relations), but I can't see how that would be of any use to you.

What do check. In the multiple-choice questions, there is one and only one correct answer. However, to demonstrate partial knowledge, you are allowed to check 2 or more boxes, but this earns you less than full points for that question. If the correct answer is not among your checked boxes, the score will even be negative. If you don't check anything (or check *all* boxes) your score is 0. Also, if you check boxes at random, your expected score is 0. For more details, read [Gudmund Skovbjerg Frandsen, Michael I. Schwartzbach: A singular choice for multiple choice. SIGCSE Bulletin 38(4): 34–38 (2006)].

1 Algorithms and data structures only

1.

(a) (*1 pt.*) Consider some data structure and the following operations, where a number represent an insertion of that number into the data structure, and "$*$" means extracting and printing one number. The data structure is initially empty.

1 12 5 * 3 7 * * * 2 4 13 * 14 15 * * *

The output resulting from these operations is:

5 7 3 12 13 15 14 4

What is the data structure?

- ☐ A binary heap.
- ☐ A multi-way heap.
- ☐ A stack.
- ☐ A FIFO queue.

(b) (*1 pt.*) Given two vertices s and t in a directed unweighted graph, which of the following graph search algorithms is best suited for efficiently finding the shortest path from s to t?

- ☐ Depth-first search.
- ☐ Breadth-first search.
- ☐ Dijkstra's algorithm.
- ☐ The Bellman-Ford algorithm.

(c) (*1 pt.*) Consider the following recursive method.

```
int fun(int x) {
    if (x < 2) return 1;
    else return 2 * fun(x/2) + fun((x + 1)/2);
}
```

What would the call `fun(10)` return?

- ☐ 1.
- ☐ 10.
- ☐ 33.
- ☐ 51.

(d) (*1 pt.*) Why is Hibbard deletion not a suitable method of deletion in a red-black tree?

- ☐ The worst-case time complexity is linear in the size of the tree, rather than its height.
- ☐ It requires recursive calls.
- ☐ It does not guarantee that the height of the tree stays logarithmic in the number of nodes.
- ☐ It does not allow reusing the storage of the deleted node.

(e) (*1 pt.*) Consider a hash table of size M (i.e., each inserted item is given a position between 0 and $M-1$ based on its hash value). Linear probing is used for handling collisions. What can be said about the number of key comparisons required for inserting N elements, where $N \leq M/2$?

- ☐ The number of comparisons for each of the insertions is guaranteed to at most equal to some constant value.
- ☐ The amortized number of comparisons per insertion is constant, implying that the total number of comparisons is guaranteed to be proportional to N.
- ☐ In the worst case, the total number of comparisons is proportional to N.
- ☐ In the worst case, the total number of comparisons is proportional to N^2.

(f) (*1 pt.*) Which of the following is true about insertion sort, but *not* true about Quicksort?

- ☐ It uses only a constant amount of auxiliary space.
- ☐ It has linearithmic time complexity.
- ☐ It is in-place.
- ☐ It uses an optimal number of exchanges.

(g) (*1 pt.*) Consider the following Java method, which produces one string from another, removing all occurrences of a specific character:

```
String filterOutChar(String in, char charToGetRidOf) {
    String out = "";
    for (int i = 0; i < in.length(); i++) {
        char c = in.charAt(i);
        if (c != charToGetRidOf) out += c;
    }
    return out;
}
```

Assuming that all intermediate results are produced (i.e., neither compilation nor execution involves any shortcut or rewrite that would reduce the time complexity), and also that the operations of the string data type are implemented as in the current Java system, what is the time complexity of the method?

- ☐ Logarithmic in the length of `in`.
- ☐ Linear in the length of `in`.
- ☐ Linearithmic in the length of `in`.
- ☐ Quadratic in the length of `in`.

(h) (*1 pt.*) A standard insert procedure in a binary heap priority queue is to add the new key at the end of an array and then "swim" it, to maintain the heap property (as defined by Sedgewick & Wayne). Consider the following alternative strategy: Before the first operation that accesses the maximum element, we don't do any *swim*, but keep the keys in the order they were inserted. Then, at the first *max* or *delMax* operation, we reorder the array by "sinking" the keys in the first half of the array in reverse order, just like is done as the first step of *heapsort*. Any insertions after this point are handled in the usual way, with *swim*. What is the result?

- ☐ The priority queue keeps working correctly, but we increase the worst case total time for N operations, by a factor $N / \lg N$.
- ☐ The priority queue keeps working correctly, and we decrease the amortized time for an insertion operation that takes place before the maximum element is accessed, from logarithmic to constant time.
- ☐ The priority queue keeps working correctly, and all operations maintain the same time complexity, disregarding constant factors.
- ☐ The priority queue does not produce keys in the right order.

(i) (*1 pt.*) What is the best way of finding the minimum element in an array, where we know nothing about the initial order of the elements? Effectiveness (that the algorithm does what we want) is the first priority, time complexity expressed in the size of the array the second, and space complexity the third.

- ☐ Sequentially scanning through the array.
- ☐ Binary search.
- ☐ Building a priority queue (binary heap).
- ☐ Building a binary search tree.

(j) (*1 pt.*) Which of the following is the best choice to use as the hash function of an integer key, to provide the properties that we expect of a hash table.

- ☐ A constant prime number.
- ☐ A random value.
- ☐ The key value itself.
- ☐ The result of an integer division, where the key value is divided by a large power of two.

(k) (*1 pt.*) We are to maintain a data structure for numbered records. To access a record, the client specifies the number of the record. Because the number of records is very large they cannot all fit in the cache memory of our hardware, and caching is known to be a significant performance factor. It is expected that the client often requests a group of records whose numbers are close together. Which of the following would be the most efficient data structure under the given conditions (and nothing else considered)?

- ☐ An inverted index.
- ☐ A hash table.
- ☐ A red-black tree.
- ☐ A sorted array.

(l) (*1 pt.*) Consider the following Java code, where the method `StdRandom.uniform(a, b)` is used for producing a random number x such that $a \leq x < b$.

```
int[] sort(int[] a) {
     int[] b = new int[a.length]; // b is to hold the result
     int i = 0;
     while (i < b.length) {
```

```
            int j = StdRandom.uniform(0, a.length);
            b[i] = a[j];
            if (i == 0 || b[i] > b[i-1]) {
                i++;            // move on
            } else {
                i = 0;          // start over from zero
            }
        }
        return b;
    }
```

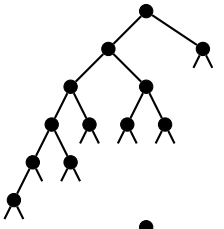
Which of the following invariant must hold at the point where `j` is assigned a value?

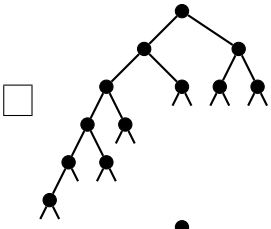
- ☐ No two elements in `a` are equal.
- ☐ The `i` smallest values of `a` are all contained in `b`.
- ☐ No two elements in `b[0]` through `b[i-1]` are equal.
- ☐ All elements in `b[i]` through `b[b.length-1]` are zero.

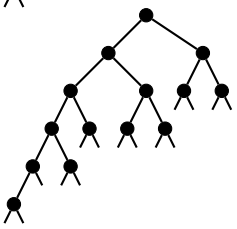
(m) (*1 pt.*) Which of the following must be true in a *complete* binary tree?

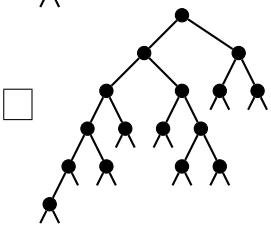
- ☐ If there is a node with distance $k > 0$ from the root, the number of nodes with distance $k - 1$ from the root is a power of two.
- ☐ The tree has no null links.
- ☐ For any node x, all nodes in the left subtree of x have key values less than or equal to that of x, and all nodes in the right subtree of x have key values greater than or equal to that of x.
- ☐ For any node x, all nodes in the subtree rooted at x have key values less than or equal to that of x.

(n) (*1 pt.*) Which of the following trees is a valid left-leaning red-black tree? Whether links are colored red or black is not disclosed by the figures.

☐ 

☐ 

☐ 

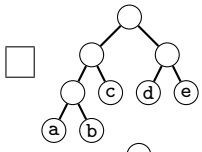
☐ 

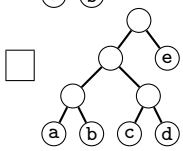
(o) (*1 pt.*) What is the minimum number of triples that can be produced by an LZ77 compression program when encoding a message consisting of 45 equal characters?

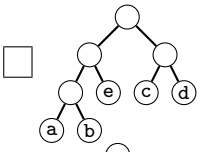
- ☐ 2.
- ☐ 6.
- ☐ 9.
- ☐ 23.

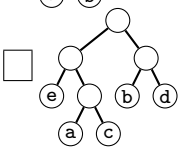
(p) (*1 pt.*) A message composed in the alphabet {`a`, `b`, `c`, `d`, `e`} is to be compressed using Huffman coding. The message is 84 characters long, and each of he characters `a`, `b`, `c`, and `d` occurs 14 times,

while e occurs 28 times. Which of the following trees can **not** be obtained through the Huffman trie construction algorithm?









2. A teaching institute is organizing a long series of lectures. We denote the number of lectures by N. The content of each of the lectures is already defined, and the next step is to decide the order in which they are to be given. The following metadata is available for each lecture:

1. The level of difficulty, a number from 1 to 10, where 1 is the easiest level, and 10 the most difficult.
2. A list of at most four topics that are *defined* (i.e. introduced and explained) during the lecture.
3. A list of at most seven topics that the lecture *refers to*, and which must have been defined in some previous lecture.

Each topic is *defined* during exactly one lecture, but may be *referred to* in zero, one, or several other lectures.

(a) (*2 pt.*) Sketch an efficient way of modeling and solving the problem of finding a lecture order where no topic is referred to before the lecture where it is defined. If it is impossible to find such an order, this circumstance should be detected and reported. State, with motivation, the order of growth expressed in N for the time and space complexity.

(b) (*4 pt.*) Whenever possible, we want easier lectures to be given before more difficult ones, to allow the students to get used to the subject gradually. We formalize this with the additional requirement that the number of times a lecture is followed by an easier lecture should be minimized. Consider solving this problem efficiently. Is there a simple change in your solution for (a) that can accomplish this? (If yes, how? If no, why not?) Sketch a solution and state, with motivation, the order of growth expressed in N for the time and space complexity.

Note: For this question, partial answers can be awarded some points, in relation to how much is (correctly) answered.