09.71.23/24 Tölvunarfræði 2/2a
(English exam)

Final exam **May 11th, 2001**

Professor: Hjálmtýr Hafsteinsson **Time: 13^{30} ? 16^{30}**

The first 5 problems are for all students (both from Tölvunarfræði 2 and 2a) Problem 6 is only for students in Tölvunarfræði 2, but problem 7 is only for students in Tölvunarfræði 2a (engineering students). In both cases ***five best problems out of six count****. All problems have the same value.*

All written materials and a calculator are allowed.

?? Note that when asked to "Describe" or "Show" then it is enough to do that in words and with drawings. If you are to write C++ code you will be asked for that specifically.

?? Give supporting arguments for all answers and remember that it is not necessary to write up definitions from the book.

1. In *path compression by halving* (Program 1.4 in the book) for the Union-Find problem, only **every other node** on the way from `p` (and `q`) to the root is made to point to its grandparent (i.e. the parent of its parent).

a) Show that this is what happens, with a 6 node example, and explain why Program 1.4 skips in such a way over every other node.

b) Implement (in C++) a version where all nodes on the way from `p` to the root are made to point to their grandparent.

2. Given a sorted circular (singly connected) linked list *L* with a head node, where the last node of the list points to the head node. Write a function in C++, that gets as input a pointer to *L* (i.e. to its head node) and a value *x*, and splits *L* up into two circular lists *A* and *B*, such that *A* contains the nodes with values ? *x*, but *B* the nodes with values > *x*. Please notice that the nodes of *L* are moved over to either *A* or *B*. Below is the a header for the function.

```
void skipta( node* L, int x, node* &A, node* &B )
```

3. In a normal *inorder* listing we always go first down the left child, then to the node itself, and finally down the right child. We can imagine a different order for this, that we call *outorder*, where we first go down the **right** child, then to the node itself, and finally down the **left** child.

Is it enough to know an inorder and outorder listing of nodes for a binary tree to uniquely determine it? In other words: are there more than one binary trees that have both the same inorder listing and the same outorder listing? Describe a method to find the tree or show a counter example.

4. We can represent ordinary binary trees in vectors in the same way as heaps are.

a) Mention the changes to the representation that are needed because ordinary binary trees are not as regular as heaps.

b) What is the maximum memory used for a *N*-node binary tree in the worst case?

c) Show a C++ function that prints out the *postorder* listing of a binary tree with this representation.

5. If we allow the insertion of multiple items with the same value into a *binary search tree* then it could be useful to know how many items have a specific value in the tree.

a) Explain carefully where items with the same value end up in the binary search tree (i.e. relative to each other). A rough drawing with an explanation should be sufficient.

b) Describe carefully an implementation of the function `count( v )`, which returns the number of items in the tree with the value `v`. The method has to be faster than just going through the entire tree and counting the items with value `v`. (Hint: Use a two-step method)

Only for students in Tölvunarfræði 2:

6. Shellsort uses Insertionsort to *h*-sort the vector for decreasing values of *h*, until *h*=1.

a) Could we use Bubble sort instead of Insertionsort? Describe in words and drawings how such an implementation of Shellsort would work and in what way it would be different from the original Shellsort.

b) What about Quicksort instead of Insertionsort in Shellsort? Describe the disadvantages of that version and the problems with implementation. Is it likely to be faster than the original Shellsort?

Only for students in Tölvunarfræði 2a (engineering students):

7. We can define a new type of heap, so called *2D heap*, for items of the type (i, j) in the following way: For a node *v* at an **even depth**, the value of the **first coordinate** of the item is larger than the value of the first coordinate of all nodes in the subtree below *v*. In the same way it holds that for a node *v* at an **odd depth**, the value of the **second coordinate** of the item is larger than the value of the second coordinate of all nodes in the subtree below *v*. We will then have two versions of the function `getmax` depending on whether we want the item with the largest first or largest second coordinate.

a) Show a possible 2D heap that contains the items (3, 8), (7, 1), (10, 5), (9, 2), (5, 7).

b) Describe the version of `getmax` for the second coordinate.

c) Describe the insertion of an item into a 2D heap.