08.71.23/24 Tölvunarfræði 2/2a

Makeup exam **August 20th, 2003**
Professor: Hjálmtýr Hafsteinsson **Time: $9^{00} – 12^{00}$**

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 solving this problem you are to use the definition of the Big-Oh notation on page 44 of the textbook.

a) Does it hold that $N^{1.1} + N \log N$ is $O(N \log N)$?

b) Does it hold that $N^k / \log N$ is $O(N^k)$?

c) Does it hold that 2^{aN} is $O(2^N)$, where a is a constant?

2. The implementation of a queue with a singly linked list that is shown in the textbook on page 169 uses the pointers `head` and `tail`, that point to the front and the back of the linked list. Explain how you can, equally efficiently, implement a queue with one pointer, call it `p`, if the singly linked list is circular.

Show an implementation in C++ of the `QUEUE` class, that only uses the pointer `p`.

3. A binary tree is uniquely determined by its *preorder* and its *inorder*. Describe in detail a **recursive** algorithm that receives a preorder and an inorder of a binary tree, and constructs the tree. Assume that the algorithms receives tvo arrays, `pre` and `in`, in addition to the number of elements, *N*. The algorithm then constructs the nodes of the tree and returns a pointer to the root of the tree that the arrays define. Your description has to be so accurate that it should be easy to translate into C++ code. You can also give this algorithm as a C++ function, but it then has to be well commented.

4. Given N values (k_1, k_2, ..., k_N) that are to be placed into a binary search tree. After the binary search tree has been constructed these N values will be searched for. The frequency of the search for each value is known. Thus we will search p_1 times for value k_1, p_2 times for value k_2, etc. Note also that the shape of the binary search tree is determined by the order in which the items are inserted, when constructing the tree.

a) It is obvious that we want to have those items that are most frequently searched for, high up in the tree. Is it always best to have the item with the highest frequency as the root of the tree? Justify or show a simple counter example.

b) Describe roughly a method to use this knowledge of the search frequency to decide the shape of the binary search tree, in order to minimize the total search time.

5. A heap can be set up using a *ternary tree* instead of a binary tree as done in the usual implementation.

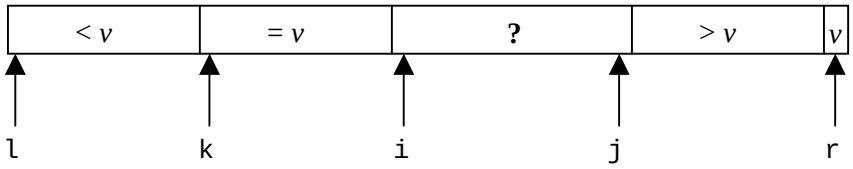
a) Show how it is possible to move up and down in the tree with simple arithmetic operations on the array indices, similar to the binary tree version.

b) Implement in C++ the functions `fixUp` and `fixDown`, corresponding to the programs on pages 384 and 385 on the textbook, if we view the heap as a ternary tree.

Only for students of Tölvunarfræði 2:

6. Chapter 7.6 of the textbook shows a method to partition the items of an array into three areas: "less than *v*", "equal *v*", and "greater than *v*". The method from the book places the items that are equal to *v* at the front and the back of the array during the partitioning process, (as described at the top of page 337), but then swaps them into the middle at the end.

You are to implement another method that uses a slightly different way of partitioning the array into three areas. The method can be described with the following figure:



Implement the method as a C++ function, similar to Program 7.2 on page 319 of the textbook. Also explain in more detail how the method works and compare it to the method of chapter 7.6 (it is implemented on page 338).

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

7. In the linear probing that is described in Chapter 14.3 of the textbook, the item *v* being entered into the hash table is placed in the next free position following the position it is supposed to go into, if that position is ocupied. This behaviour can be changed in various ways. Describe the benefits and the disadvantages of each of the following methods compared to the method in the book, and under what circumstances they are better and when they are worse.

a) Instead of moving *v* down the hash table when there is a collision, *v* is placed in the correct position, but the item already in that position is moved down to the next free position.

b) Instead of looking only for a free position down the table, we look for the next free position above **or** below the original position, if *v*'s position is taken.

c) Instead of going always one position down looking for a free position we use a predetermined list of numbers (always the same one), and add them to the original position. For instance look first at position + 25, then at position + 4, then position + 81, etc. All the numbers are modulo the size of the table.