Programming (FEB22012[X])

Example exam

Block 1 / 2014

Instructions

Each question should be answerable within 1 page of text with normal size hand writing (but you are allowed to write more). Remember to write your name and student number in every answer sheet.

In the actual exam, no extra material (neither dictionaries nor calculators) is allowed. Take with you only a pencil, a sharpener and an eraser (or alternatively a pen if you're sure not to make any mistakes while writing).

Questions

1. Explain how algorithm running time analysis is usually performed, what does the big-O (e.g. O(n)) notation mean, and what is the pratical difference of having algorithms with time complexities O(n) and O(n!).
2. Describe the differences between procedures and functions, and what are the so-called 'undesired side-effects'.
3. Describe the structure of heap (the data structure), provide its operation complexities, and explain a programming application where heaps are applied.

Example answer (q1)

Running time analysis of algorithms refers to how many primitive operations are required for an algorithm to complete with a certain input. The goal in the analysis is to describe the running time in terms of the input size n. This size may refer to, for example, size of the array to be sorted or to the number of bits in the integers to be summed. The running time often changes not only according to the input size, but also according to the input contents. For example, the number of operations required by insertion sort depends on ordering of the elements in an array - the algorithm performs a lot faster with an array that is already (almost) sorted than with one that is not.

When analysing the running time, we are usually interested in the growth rate of the running time regarding growth of the input size. That is, when input size increases by 1, how much does the running time increase? The differences in programming languages and computer hardware cause calculation of the running time in terms of exact number of operations to be irrelevant, and the running time is instead often analyzed regarding it's asymptotic groth rate as a function of n. Upper bound to this is given with the O-notation, that describes the highest order term of the growth rate without the associated constants (e.g. $\mathrm{f}(n) = \mathrm{O}(n^2)$). Note that larger running time (e.g. $O(n^3) > O(n^2)$) does not necessarily mean that the algorithm is slower: it means that with input size larger than a certain value it is. This value can be so large that the asymptotically larger running time algorithm is faster in practice.

Most of the time we are interested only in the worst case running time of an algorithm, as it definitely happens sometimes and can be considerably higher than the best case one. For example, if an algorithm usually takes O(n) operations to complete, but with some inputs it takes O(n!) operations, it is useless in practice as these worst case inputs cause the computation to take a few hundred years.

Example answer (q2)

Both procedures and functions are methods. Whether the method is a procedure or a function depends on whether it produces side effects or not. Side effects mean, that one of the input parameters gets altered during the method execution so, that the change is visible to the caller as well. For example, a sorting method sort(array), that does not return a value but changes the order of the element in the input array, would accomplish the sorting functionality through a side effect: the input parameter gets altered and the modification would be visible to the caller. Procedures are methods with possible side effects, whereas functions never produce side effects (note that in matlab all methods are functions, i.e. there are no procedures).

In addition to desired side effects such as the case of sorting the array above, there are also undesired side effects. For example, if you have a method countSum(array) that counts the sum of numbers in the array given as the input parameter, and returns the sum, the caller would probably not expect the array contents to be changed during the counting. If the array contents change and this behaviour is not documented, then it is an undesired side effect: an unexpected change of one of the variables passed as a parameter. Undesired side effects are a cause of bugs that are very hard to find.

Example answer (q3)

Heap is a data structure that is essentially a binary tree with additional restrictions. Binary trees are node-based structures with one node being the root, and each node having up to two child nodes. With (max) heaps, each node has at least as large contents ('key' or something similar) as its children. That is, if a heap node contains number 9, then its children cannot contain values > 9. Additionally, the heaps are complete: each level of height x except the last one has exactly 2^x nodes (root has 1, first level 2, second 4, ...), and the last level has empty slots only on the right hand side of the last node.

Heaps allow only two operations: inserting a value, and deleting the top node. Insertion has complexity $O(\log n)$ because the new node is added to the first 'free' slot at the bottom-right of the heap, which possibly causes a violation of the heap condition of parent having to have at least large key value as its children. If this happens, the condition has to be restored leading to the non-constant $O(\log n)$ complexity. When the top node (largest value in heap) is deleted, it is replaced with the last node, and again the same condition might be violated. Thus, also the deletion operation is of $O(\log n)$ complexity. Accessing the maximum value is of O(1) complexity as it involves nothing more than inspecting the root node.

Heaps are useful, for example, as priority queues. The nodes can then contain information on the elements to be queued and the key values would be their priorities: in such applications there would always be a need to remove (dequeue) only an element corresponding to the largest key. Priority queues are used in standard operating systems for the scheduling of processes (which are essentially programs).