Aalto University, School of Science
Department of Computer Science

CS-A1140 Data structures and algorithms — Autumn 2023

Exam December 4th, 2023, at 17:00–20:00

Use of calculators and material is not allowed in the exam.
You can answer the questions in English, Finnish or Swedish.
Your answers should be clear, well-structured and concise.
Note: the questions continue on the other side of the paper!

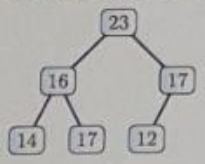
1. First, (i) define the concept of "a *stable* sorting algorithm".
   Then, consider the Scala program on the right and answer the following questions. Remember to *justify* answers (iii)–(vi) (each with *at most* few sentences!). Use the Θ- and O-notations when needed, and denote the length of the argument array a by n.

```
def sortX(a: Array[Double]): Unit =
  var i = 1
  while i < a.length do
    val element = a(i)
    var j = i
    while j > 0 && element < a(j-1) do
      a(j) = a(j-1)
      j -= 1
    a(j) = element
    i += 1
```

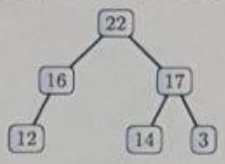
   (ii) Which well-known sorting algorithm does the program implement?
   (iii) Is the algorithm stable?
   (iv) What is the worst-case running time of the program?
   (v) What is the best-case running time of the program?
   (vi) How much extra memory does the algorithm use?
   (vii) In what kind of situations should one use this sorting algorithm in practise?

   12 points

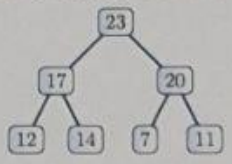
2. As discussed in the round 3 of the course, maximum priority queues can be implemented with the binary max-heap data structure. (i) Define the properties that a binary tree must satisfy in order it to be a binary max-heap. (ii) Do the properties hold for the binary trees (a) and (b) below? Justify your answer briefly. In the figures, as well as in the course material, the elements are integers drawn inside the nodes of the tree and the priority ordering is the usual less-or-equal ordering $\leq$. (iii) Describe the efficient method, discussed in the round, for storing binary max-heaps in computer memory and illustrate it for the binary max-heap (c) below.



(a)



(b)



(c)

   Recall that inserting a new element, and removing an element with the highest priority, can be done in time $O(\log n)$, where n is the number of elements currently in the binary max-heap. (iv) Illustrate the steps the insertion algorithm makes when the element 18 is inserted in the max-heap (c) above. (v) After this, illustrate the steps done when removing an element with the highest priority from the max-heap just obtained.

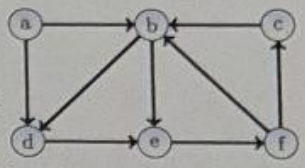
   10 points

3. (i) Suppose we use a hash table for storing a set of 32-bit integers. Let the size m of the hash table be 7 in the beginning. Use the hash function $h(x) = x \bmod m$ and *open addressing with linear probing* as the collision resolution method. Describe the contents of the hash table, after each step, when the integers 15, 9, 8, and 22 are first inserted in this order in the table, and 9 is then removed from the table.

   Also answer the following questions about hashing and hash tables. (ii) What do the terms "collision" and "load factor" mean? (iii) What does "rehashing" mean and when/why should one perform it? (iv) Why the hash function $h(x) = x \bmod m$ for integers may not be the best choice when the size m of the hash table is a power of two (that is, $m = 2^k$ for some positive integer k)? 9 points

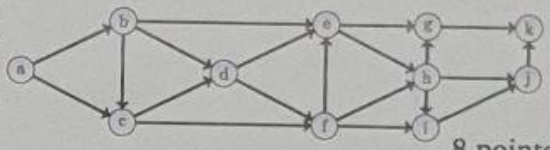
4. (i) Give pseudo-code, or a structured compact verbal description, of the *breadth-first search* algorithm for finding shortest paths from a source vertex to all other vertices in a directed graph. (ii) What data structures are needed to implement the algorithm?

   Consider the graph on the right.
   (iii) Illustrate its *adjacency list representation*.
   (iv) Explain/illustrate how the breadth-first search algorithm works on the graph when the source vertex is a.



   (v) Define the concept of a "strongly connected component" for directed graphs. (vi) What are the strongly connected components of the graph shown above? 12 points

5. Take the following computational problem: Given a directed *acyclic* graph $G = (V, E)$ and two vertices s (the source) and t (the target) in it. How many different paths there are from s to t in the graph?
   (i) Describe a dynamic programming algorithm for solving the problem. Hint: denote, for each vertex v, the number of paths from v to t by $\alpha(v)$ and develop a recursive definition for the value $\alpha(v)$. Your algorithm should run in time $O(|V|+|E||V|)$, where the latter term $|V|$ is for handling the possibly quite large values $\alpha(v)$.
   (ii) Apply your algorithm to the graph on the right when the source vertex is a and the target vertex is j. It is enough to give the values $\alpha(v)$ for each vertex v *in the order they are evaluated or updated.*



   8 points

6. Consider the Scala program on the right. The par.parallel(code1,code2) construction is as in the course material, executing code1 and code2 in parallel and returning their return values. (i) Describe with one or two sentences, what kind of value the program computes. What are (ii) the span and (iii) the work of the program in the O-notation? Denote the length of the argument array by n and justify each answer with at most few sentences.
   (iv) How could the program be improved to work faster in practise? 8 points

```
def parX[A](a:Array[A], f: A => Int) =
  require(a.nonEmpty)
  def inner(start: Int, end: Int): Int =
    if start == end then
      f(a(start))
    else
      val mid = start + (end - start)/2
      val (l, r) = par.parallel(
        inner(start, mid),
        inner(mid+1, end)
      )
      l + r
  inner(0, a.length-1)
```

7. At what time did you finish answering the exam questions? 1 points