Aalto University, School of Science
Department of Computer Science
Tommi Junttila (050-4300861)

CS-A1140 Data structures and algorithms — Autumn 2018

Exam February 21st, 2019, at 13:00–16:00

Use of calculators is not allowed in the exam.
You can answer the questions in English, Finnish or Swedish. Note: your answers should be clear, well structured and concise.

1. (a) A fixed-size mutable array (Array in Scala) does not support efficient insertion of new elements at the end of the array. But mutable *resizable arrays* (ArrayBuffer in Scala) support such operations in amortized constant time — explain how this is done and what "amortized constant time" means (proofs with recurrences are not required).

   (b) A *mutable queue* data structure supports the following operations:

   - enqueue(*e*), which adds the element *e* at the end of the queue, and
   - dequeue(), which removes and returns the first element in the queue.

   Describe how such a data structure can be implemented so that both of these operations can be performed in constant time.

   10 points

2. Define the following concepts: (a) a sorting algorithm that *works in-place*, and (b) a *stable* sorting algorithm.
   Consider the Scala program on the right. (i) Which well-known sorting algorithm does it implement? (ii) Is the algorithm stable? (iii) Does the algorithm work in-place? (iv) What is the worst-case running time of the program? (v) What is the best-case running time of the program? (vi) Do the worst and best case running times change if we uncomment the first line in the method HELPER?
   In questions (iv), (v) and (vi), denote the length of the argument array *a* with *n*.
   Justify each answer with at most few sentences.

```
def sort(a: Array[Int]): Unit = {
  val aux = new Array[Int](a.length)
  def helper(l: Int, m: Int, r: Int): Unit = {
    // if(a(m-1) <= a(m)) return
    var (i, j, d) = (l, m, l)
    while(i < m && j <= r) {
      if(a(i) <= a(j)) {aux(d) = a(i); i += 1}
      else {aux(d) = a(j); j += 1}
      d += 1
    }
    while(i < m) {aux(d) = a(i); i += 1; d += 1 }
    while(j <= r) {aux(d) = a(j); j += 1; d += 1 }
    d = l
    while(d <= r) {a(d) = aux(d); d += 1 }
  }
  def inner(l: Int, r: Int): Unit = {
    if(l < r) {
      val m = l + (r - l)/2
      inner(l, m)
      inner(m+1, r)
      helper(l, m+1, r)
    }
  }
  inner(0, a.length-1)
}
```

   13 points

3. Explain how hashing and hash tables can be used to implement a mutable set data structure when "open addressing" (also called "closed hashing") is used as the collision resolution method. Your explanation should include answers to the following

questions: (i) What is a "hash function"? (ii) What do the terms "collision" and "load factor" mean? (iii) What does "rehashing" mean and when/why should one perform it? (iv) Define the concept "probe sequence"; what properties should such a sequence have? (v) How does "linear probing" work?

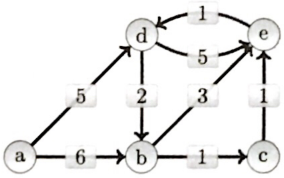
Assume that we are using a hash table for storing 32-bit integers. Let the size m of the hash table be 11 in the beginning. Use the hash function $h(x) = x \bmod m$ and open addressing with linear probing as the collision resolution method. Describe the contents of the hash table (after each step) when the keys 12, 3, 24, and 1 are inserted, in this order, in the table.
8 points

4. Explain (with pseudo-code or with a very clear and structured verbal description) Dijkstra's algorithm for finding shortest paths in edge-weighted directed graphs with non-negative edge-weights. What kind of data structures are needed in the algorithm?

   Describe/illustrate how the algorithm works on the graph shown on the right when the source vertex is the vertex a.

   Given an edge-weighted directed graph $G = (V, E, w)$ with $|V| = n$ vertices and $|E| = m$ edges, what is the worst-case running time of the algorithm? Justify your answer with few sentences.

   Explain (perhaps with a simple example) why the algorithm does not necessarily work when the edge-weights can be negative.
12 points



5. Consider the Scala program on the right. It computes the maximum of an array of integers in parallel. The par.parallel(code1,code2) construction is as in the lecture material, executing code1 and code2 in parallel and returning their return values. What are the (i) span, (ii) work, and (iii) amount of parallelism of the program? Denote the length of the argument array by n and justify each answer with at most few sentences. How could the program be improved to work faster in practise?

```
def parMax(a: Array[Int]): Int = {
  require(a.nonEmpty)
  def inner(start: Int, end: Int):Int={
    if(start == end) a(start)
    else {
      val mid = start + (end - start)/2
      val (l, r) = par.parallel(
        inner(start, mid),
        inner(mid+1, end)
      )
      l max r
    }
  }
  inner(0, a.length-1)
}
```

8 points

6. Define the following concepts:

   - A polynomial-time solvable decision problem.
   - A decision problem in **NP** (non-deterministic polynomial time).
   - An **NP**-complete problem.

   Describe one **NP**-complete problem and two different approaches for solving it in practise (detailed algorithms are not required).
8 points

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