Algorithms & Data Structures 2024/25

Practical Week 7

I realise that model answers tend to be easily available. Please however try to come up with your own solutions. If you do get stuck then ask the demonstrators for help.

0. Finish anything that may be left over from last week.
1. Fun with logarithms:

   (a) Evaluate the following logarithms. First, think about what the value should be. Then, use a calculator to confirm that your answer is correct.
   $\log_4 64$ $\log_{64} 4$ $\log_{16} 64$ $\log_9 27$ $\log_8 16$

   (b) Prove that $f(x) = a^x$ and $g(x) = \log_a x$ are inverses of each other (one proves that f and g are inverses by formally showing that $f(g(x)) = g(f(x)) = x$).

   (c) Express each of the following in terms of natural logarithms.
   $\log_5 42$ $\frac{5}{\log_2 10}$ $\log_3 \sqrt{e}$

   (d) Use the logarithm laws to simplify the following:
   $\log_2 xy - \log_2 x^2$ $\log_2 \frac{8x^2}{y} + \log_2 2xy$ $\log_3 9xy^2 - \log_3 27xy$
   $\log_4 (xy)^3 - \log_4 xy$ $\log_3 9x^4 - \log_3 3x^2$

   (e) Prove $a^{\log_b(n)} = n^{\log_b(a)}$.

   (f) Solve each equation for x (you may need to use a calculator).
   $100 = 50e^{-x}$ $1/4 = 5^{2x-1}$ $\ln(2x+5) = 0$ $\log_x 6 = 1/3$

2. Even more fun with InsertionSort:

   This is (pseudo-code for) a generic version of InsertionSort, one of the simplest sorting algorithms known:

   ```
   INSERTIONSORT (a_1, ..., a_n: real numbers with n ≥ 2)
   for j = 2 to n do
     x = a_j
     i = j − 1
     while i > 0 and a_i > x do
       a_{i+1} = a_i
       i = i − 1
     end while
     a_{i+1} = x
   end for
   ```

   (a) Simulate by hand the execution of InsertionSort on a few small examples, say, with $n = 8$ input elements. Permutations of the set $\{1, \ldots, 8\}$ are fine.

   You may also implement InsertionSort in whatever language you are comfortable with. Be careful though – some languages let arrays start at index 0 (that is, the input would be in positions $0, \ldots, n-1$), whereas above we assume that it starts at index 1 (i.e., with elements in positions $1, \ldots, n$).

   Specifically, try to find out what permutations make InsertionSort "take long", and what make it be "quick" (we're interested in "structural" properties). What part of the algorithm behaves differently when running it on different permutations?

   As is common practice in the analysis of sorting algorithms, you may focus on just counting the number of comparisons involving input elements, that is, the comparisons "$a_i > x$" in the code above.

(b) Try to prove, as formally, precisely and concisely, *correctness* of InsertionSort as given above. By that I mean, prove that regardless of what the input looks like, when the algorithm terminates then the corresponding numbers will be in sorted order.

Hint: You may wish to use induction on the number of rounds; an appropriate hypothesis might be "after so-and-so many rounds of the outer (for) loop the first so-and-so many input elements are in order".

No waffling and no hand-waving, please!