Algorithmics - Tutorial Sheet 1

Sorting algorithms

---

1. **[Work in pairs]** In Lecture 1 we traced the radix sorting algorithm for the sequence:

   $$15 \quad 43 \quad 5 \quad 27 \quad 60 \quad 18 \quad 26 \quad 2$$

   with $b = 2$ the size of the bucket. Trace the radix sorting algorithm for $b = 3$ for the same input sequence. The binary encodings are:

   $$\begin{array}{llll} 15 = 001111 & 43 = 101011 & 5 = 000101 & 27 = 011011 \\ 60 = 111100 & 18 = 010010 & 26 = 011010 & 2 = 000010 \end{array}$$

2. Suppose you have in front of you a pile of (several hundred) forms, in random order, each form identified by a unique (7 digit) matriculation number. Describe an algorithm, based on the idea of Radix Sort, that you could use, in practice, to sort the forms into order based on the matriculation number.

3. **[Work in pairs]** Draw a `trie` representing the following set of English words that can be formed using the character set $\{a, c, e, r, t\}$:

   $$\{\ ace, acer, acre, act, are, area, arete, art, at, ate, car, care, carer, caret, cart,$$
   $$cat, cater, crate, crater, create, ear, eat, eater, era, erect, race, racer, rare,$$
   $$rarer, rat, rate, react, tact, tar, tare, tart, tat, tear, trace, tract, treat, tree\ \}$$

   How much memory would your `trie` use if

   (a) implemented using an array of pointers at each node?

   (b) implemented using a linked list of pointers at each node?

4. Describe an algorithm to delete a specified word from a `trie`.

Difficult/challenging questions (non examinable)

5. Show that no comparison-based algorithm can guarantee to search a sequence of n elements for a particular value in better than $\mathcal{O}(\log n)$ time.

   Hint: recall any comparison-based algorithm can be represented by a decision tree.

6. Say you are tasked with developing an advanced contact management system that needs to support quick searches for names or numbers, even with partial inputs, and must scale to accommodate millions of contacts. The system should offer immediate suggestions for possible matches as the user types, enhancing user experience and system usability. How would you approach this problem? Consider the following points: trie data structure design (nodes and root), insert operation, search and autocomplete operation, deletion operation, optimisation and efficiency, particular implementation considerations.