MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Examinator: Lars Arvestad

Tentamensskrivning i
DA3018 Datalogi för matematiker
7.5 hp
2022-05-30

- **Write clearly.** Hard-to-read answers risk zero points.
- **Use one side of the paper**.
- Justify your answers (unless otherwise stated).
- **Grading thresholds:** E: 25, D: 30, C: 35, B: 40, A: 45

---

1. (a) What do computer scientists mean with ”unit time” when analyzing time complexity? (1p)

(b) What is the time complexity of accessing element i in an array of size n (and $i < n$)? (1p)

(c) Give one justification for *asymptotic* time complexity analysis. (1p)

(d) What is the *worst-case* time complexity of QuickSort and when does it happen? (2p)

2. (a) Write pseudocode for a *recursive* function `write_keys()` that, given a *binary search tree* with integer elements, writes the keys in order, from smallest to largest. (2p)

(b) Explain why your pseudocode gives ordered output. (2p)

(c) What is this kind of traversal called? (1p)

(d) Give an example of a ”unbalanced” binary search tree and discuss briefly whether the performance of `write_keys()` is affected by the binary search tree being balanced or not. (2p)

3. Explain breadth-first traversal.

(a) Present pseudocode for the algorithm. (3p)

(b) Explain the intuition of breadth-first traversal, i.e., give a brief explanation in words, referring to your pseudocode. (1p)

(c) State and justify the time complexity of breadth-first traversal. (3p)

4. We want be able to detect whether a string s is a substring of another string t and will consider two ways of doing that.

If s is a string, then $s[i]$ is the i'th character of s and $s[i, j]$ is the substring from position i to j. If s is a substring of t, then there exists i and j such that $s = t[i, j]$. For example, $s =$ ”our” is a substring of $t =$ ”course”, because $s = t[1, 3]$.

(a) The pseudocode below defines the function `is_substring` that determines whether s is a substring of t or not. What is the time complexity of this algorithm? (3p)

```
def is_substring(s, t):
    def at_position(s, t, i):
        for j from 0 to len(s)-1:
            if s[j] != t[i+j]:
                return False
        return True

    for i from 0 to len(t)-len(s):
        if at_position(s, t, i):
            return True
    return False
```

(b) A friend (who has taken the course *Algorithms and complexity*) helps you construct a *suffix array* A for a string t with the property $t[A[i], n] \leq t[A[i + 1], n]$ for all $0 \leq i < n$ and $n = \text{len}(t)$. A defines suffices of t, *sorted* alphabetically, by giving indices into suffix starting points in t. For example, if $t =$ ”ABBA” then $A = [3, 0, 2, 1]$ because ”A” < ”ABBA” < ”BA” < ”BBA”. Note that $t[A[0], 3] =$ ”A” and $t[A[1], 3] =$ ”ABBA”.

Use a suffix array to suggest an asymptotically faster algorithm for checking substrings than `is_substring` above. You get points for pseudocode, explanation of the algorithm, and justifying the time complexity. *Note*: You do not have to construct the suffix array yourself. Assume a function `suffix_array(t)` returns a correct suffix array. (5p)

5. Having inherited some land with spruce and pine (Swe: *gran och tall*), June Iper has a problem. She needs to thin the forest (Swe: *gallra skogen*), i.e., make enough space around "good" trees that they will grow better and produce wood of higher quality. Identifying bad trees is easy for June, but sometimes good trees grew close to each other and one has to select which good trees to keep and which ones to cut down. Of course, one would like to keep as many good trees as possible too!

Help June by formulating a *computational problem* (Swe: *beräkningsproblem*) to communicate to her forester that would do the actual work. *Note*: we are not asking for an algorithm, just a formal statement of the problem. (5p)

6. Suppose you are helping the Physics department process about 10^9 observations from the IceCube Neutrino Observatory, in which each observation is a point, with coordinates (x, y, z), a time stamp (stored using a 4-byte value), and a measurement of "neutrino intensity" at (x, y, z).

For some reason the data has to be sorted by time.

(a) How much RAM memory do you need to store the data in an array? State assumptions you need and make an estimate. (3p)

(b) Suppose you have a colleague that suggests sorting the data using selection sort. How do you dismiss that idea? (3p)

(c) Which sorting algorithm would you use? Justify your choice and analyze how much more memory would be needed. (3p)

(d) Instead of putting the data in an array and sorting it, one could use a binary search tree and get the sorting "for free". How much more memory would is needed for the binary search tree? Make an estimate with clear assumptions. (3p)

7. Suppose you have a friend who works in a lab where they are studying genes. There are more and more genes being studied, so it becomes important to work efficiently. A common computational problem in the lab is to group genes that share positive outcomes in experiments. Can you help your friend?

In a general scenario, there is a set S of n genes for which we have m experiments with binary outcomes. The outcome of experiment j for gene i is $x_{i,j} \in \{0, 1\}$. There are more genes than experiments and for each gene there is at most a constant c experiments with positive outcomes.

The objective is now to partition the set of genes into subsets S_i, $S = \bigcup_i S_i$, using positive experiments. If $x_{a,j} = x_{b,j} = 1$, then genes a and b should be in the same subset and, transitively, if a and b is in the same subset and there is an experiment $k \neq j$ such that $x_{b,k} = x_{c,k} = 1$, then a and c should be in the same subset, even if $x_{a,k} = 0$ and $x_{c,j} = 0$.

Your input is a list of *positive* experiments, see figure 1, as a list of pairs (g, e) stating that there was a positive outcome of experiment e for gene g.

Give an algorithm that partitions S correctly in time $O(n)$. It might help modeling this as a graph problem. (6p)

```
[(g1, 2), (g1, 3), (g2, 1), (g2, 3), (g3, 4), (g4, 4), (g5, 4), (g6, 5)]
```

Figur 1: Example input for problem 7. There are six genes and five experiments, inducing the groupings $\{1, 2\}, \{3, 4, 5\}, \{6\}$ since g1 and g2 share a positive outcome in experiment 3; g3, g4, and g5 share experiment 4, and g6 is along with experiment 5 coming out positive.