Homework #3
Linear Selection and Sorting & String Matching
Due: Nov. 26, 2018

- The assignment should be delivered digitally by email. Your message should be sent to **pribeiro@dcc.fc.up.pt** with subject *"[ALG HWK3] - FirstName Last Name StudentNumber"*
- Your delivery should be an **electronic document**. This could be something like:
  - scanned handwritten answers (make sure they are legible)
  - a plain text file (.txt)
  - a PDF created with another program (we recommend the usage of LaTeX)
- You may work in a (small) group, but you should do your **individual writeup**. This means you can collaborate by talking about the problems, but **you should not copy writeups**.
- Please **acknowledge any help you got** and state any references you consulted (including internet pages) and any students with whom you talked about the problem.
- Answers should be **submitted until 23:59 of the due date**. Up to 24h of delay will get you a 25% penalty. 24h to 48h of delay will get you a 50% penalty. After 48h your work will not be counted.

Median of Medians

1. Consider the deterministic median of medians selection algorithm given on class, that initially divides the numbers to sort into $n/5$ groups of size 5

   (a) Imagine that you now **you divide them into $n/9$ groups of size 9**. **Show that the resulting selection algorithm is still linear**, that is, it can compute the k-th smallest element in $O(n)$. (for simplicity you can assume in your proof that n is a multiple of 9).

   (b) Imagine that you now **you divide them into $n/3$ groups of size 3**. **Show that the resulting selection algorithm is <u>not</u> linear**. What would the runtime complexity be in this case? (for simplicity you can assume in your proof that n is a multiple of 3).

   Hint for the two previous questions: how many elements can you now guarantee that are smaller and how many are greater than the median of medians? what is the resulting recurrence?

Sorting numbers

2. Let k be a positive integer. Show how you can **sort** n numbers in the range 1 to n^k in $\mathcal{O}(n)$ time.

   Hint: what linear sorting algorithms do you know? which one could be applicable here? how to take advantage of the fact that the numbers are between 1 *and* n^k*?*

Sorting points in a circle

3. Suppose you have n points (x_i, y_i) uniformly distributed inside a circle of radius r and center in the origin coordinate $(0, 0)$. Suppose you want to **sort the points according to their distance** $d_i = \sqrt{x_i^2 + y_i^2}$ **to the origin**. Show how you could do this in an average-case running time of $\Theta(n)$.

   Hint: can you adapt bucket sort to this case? what points should each bucket contain?

Wildcard KMP

4. Describe a modification of Knuth-Morris-Pratt (KMP) algorithm in which you only want to report the **first occurrence** and the pattern can contain any number of **wildcard** symbols '*', each of which matches an arbitrary substring. For instance, the pattern "al*thm*s" matches "ilovealgorithmscourse": the first wildcard matches *"gori"* and the second wildcard matches an empty substring. The algorithm should run in $\mathcal{O}(n + m)$, where n is the size of the text and m the size of the pattern.

   Hint: consider the case were you have just one wildcard: essentially you need to have a match of what is before the wildcard, followed by anything, followed by a match of what is after the wildcard...

Simple Suffix Arrays

5. Suppose you already have available a suffix array SA of a string s with length n, and its corresponding LCP array (containing the longest common prefix of consecutive suffixes). **Describe an algorithm (based on suffix arrays)** that computes $L(s, k)$ the **length of the biggest substring that appears in** s **at least** k **times**. Indicate its runtime complexity.

   For instance, $L(\text{"}banana\text{"}, 2) = 3$ because "*ana*" appears 2 times and has length 3 (no other larger substring appears at least 2 times.).

   Besides describing the algorithm, you should exemplify its functioning for computing $L(\text{"}senselessness\text{"}, 2)$ (don't forget to show its corresponding suffix and LCP arrays).

   Hint: in what positions of SA appear the common substrings?

Palindromes

6. A **palindrome** is a word which reads the same backward as forward, such as "madam". Given a string S of size n, **describe an algorithm using suffix trees or suffix arrays (choose one)** that can compute $P(S)$, the size of the **largest palindrome which is a substring** of the word, and indicate its runtime and memory complexity (assume you can build a suffix tree or array in $\mathcal{O}(n)$).

   For instance, $P(\text{"}banana\text{"}) = 5$, because the largest substring which is also a palindrome is "anana". Another example is $P(\text{"}pedro\text{"}) = 1$, because there are no repeated letters. Yet another example would be $P(\text{"}racecar\text{"}) = 7$ because the entire word "racecar" is a palindrome.

   Besides describing the algorithm, you should exemplify its functioning for computing $P(\text{"}adambcmada\text{"})$.

   Hint: the homework should have at least one exercise without hints, right?