Algorithmic Combinatorics — Wi 2016–17

Shagnik Das — Tibor Szabó

Exercise Sheet 1

Due date: 14:00, Oct 25th, by the end of the lecture.
Late submissions will be erased and written over, à la the Palimpsest.

You should try to solve all of the exercises below, and submit two solutions to be graded — each problem is worth 10 points. We encourage you to submit in pairs, but please remember to indicate the author of each individual solution.

Exercise 1 Consider the following algorithm to find the minimum of a set of 2^k numbers.

```
Algorithm: MIN
Data: A = {a_1, ..., a_n}, n = 2^k ≥ 2
Result: MIN(A) = min{a_1, ..., a_n}
if n = 2 then
    if a_1 < a_2 then
        return a_1;
    else
        return a_2;
    end
else
    for 1 ≤ i ≤ n/2 do
        set y_i = MIN({a_{2i-1}, a_{2i}});
    end
    return MIN({y_1, ..., y_{n/2}});
end
```

(a) Show that the MIN algorithm requires $n-1$ comparisons to find the minimum element, and that this is the best possible.

(b) After running the MIN algorithm to find the minimal element, how many additional comparisons are required to find the second-smallest element?

(c) Deduce a sorting algorithm that requires $\sim n \log_2 n$ comparisons to sort $n = 2^k$ elements.

Exercise 2 Consider the following game. I think of an integer x between 1 and n, and your job is to try and determine x. You are allowed to ask questions of the form "Is $x < y$?" or "Is $x > y$?" for any y.

(a) Show that you can find x with only $\lceil \log_2 n \rceil$ questions, and that this is best possible.

To make your job slightly harder, I am now allowed to lie to you at most k times, for some constant k.

(b) How many questions do you now need to determine x? Provide the best lower and upper bounds that you can find.

Exercise 3 A man has just bought n horses, and wants to order the horses by speed. However, his private racecourse only has three lanes, so he can only race three of his horses against each other at a time and determine their relative order.[1]

(a) Prove that at least $\lceil \log_6(n!) \rceil$ races are needed to order the horses.

(b) Show that the horses can be ordered within at most $\sim n \log_3 n$ races.

Bonus (5 pts) The above bounds are separated by a constant factor. Can you give better bounds to close this gap?

Exercise 4 Given a connected graph G and an arbitrary vertex $v_0 \in V(G)$, show that the breadth-first search starting at v_0 returns a tree T_B that preserves distances[2] to v_0; that is, for every $v \in V(G)$, $d_G(v_0, v) = d_{T_B}(v_0, v)$.

Exercise 5 Show that the first m edges added in Kruskal's algorithm form a m-edge forest of minimum weight.

[1] He allows sufficient breaks between the races so that the horses do not get tired, and their performance is an accurate reflection of their speed.

[2] In an unweighted graph $G = (V, E)$, the *distance* $d_G(u, v)$ between vertices $u, v \in V$ is the minimum length of a path from u to v. If u and v are in separate connected components, we may take their distance to be infinite.

Exercise 6 Somebody suggests the following algorithm for building a minimum weight spanning tree, with the additional feature of always having a connected subgraph throughout the process.

```
Algorithm: LIGHT SPANNER
Data: A connected graph G = ([n], E), ω : E → ℝ
Result: LIGHT SPANNER(G, ω) = T ⊆ E, a minimum weight spanning tree of the
        component of the vertex 1

/* Initialisation: sort edges, start with empty tree at vertex 1      */
Sort edges by weight, E = {e_1, e_2, ..., e_m}, ω(e_i) ≤ ω(e_j) for all i ≤ j ;
Set C = {1} ;
Set T = ∅ ;
/* Build tree: always add lightest edge extending component C         */
while true do
    i = 1 ;
    while i ≤ m do // look for first edge extending C
        if |e_i ∩ C| = 1 then // found edge extending C
            T = T ∪ {e_i} ;
            C = C ∪ e_i;
            break;
        end
        i = i + 1;
    end
    if i = m + 1 then // no edge extended C, so C is a connected component
        return T;
    end
end
```

Will this even greedier algorithm succeed? Either prove its correctness or demonstrate some input on which it fails.