Chapter 1

Correctness - Recitation 2

Proving algorithms correctness is necessary to guarantee that our code computes its goal for every given input. In that recitation, we will examine several algorithms, analyze theirs running time and memory consumption, and prove they are correct.

Example 1.0.1 (Leading Example.)**.** *Consider n numbers $a_1, a_2, ..., a_n \in \mathbb{R}$. Given set Q of $|Q|$ queries, such each query $q \in Q$ is a tuple $(i, j) \in [n] \times [n]$. Write an algorithm that calculates the* $\max_{i \leq k \leq j} a_k$.

1.1 Correctness And Loop Invariant.

Correctness. We will say that an algorithm $\mathcal{A}$ (an ordered set of operations) computes $f : D_1 \to D_2$ if for every $x \in D_1$ the following equality holds $f(x) = \mathcal{A}(x)$. Sometimes when it's obvious what is the goal function f, we will abbreviate and say that $\mathcal{A}$ is correct.

Examples of functions f might be: file saving, summing numbers, or posting a message in the forum.

Loop Invariant. Loop Invariant is a property that characteristic a loop segment code and satisfy the following conditions:

1. Initialization. The property holds (even) before the first iteration of the loop.
2. Conservation. As long as one performs the loop iterations, the property still holds.
3. (optional) Termination. Exiting from the loop carrying information.

Example 1.1.1. *Before dealing with the hard problem, let us face the naive algorithm to find the maximum of a given array.*

Claim 1.1.1. *Consider the while loop. The property: "for every $j' < j \leq n+1 \Rightarrow a_{j'} \leq a_i$" is a loop invariant that is associated with it.*

Proof. first, the initialization condition holds, as the at the first iteration $j = 1$ and therefore the property is trivial. Assume by induction, that for every $m < j$ the property is correct, and consider the j-th iteration. If back again to line (5), then it means that $(j-1) < n$ and $a_{j-1} \leq a_i$. Combining the above with the induction assumption yields that $a_i \geq a_{j-1}, a_{j-2}, ...a_1$. □

Result: returns the maximum of $a_1 ... a_n \in \mathbb{R}^n$

```
1
2  for i ∈ [n] do
3  |   j ← 1
4  |
5  |   while j ≤ [n] and a_i ≥ a_j do
6  |   |   j ← j + 1
7  |   end
8  |
9  |   if j = n + 1 then
10 |   |   return a_i
11 |   end
12 end
13 return Δ
```

Algorithm 1: naive maximum alg.

Correctness Proof. *Split into cases, First if the algorithm return result at line (9), then due to the loop invariant, combining the fact that $j = n + 1$, it holds that for every $j' \leq n < j \Rightarrow a_i \geq a_{j'}$ i.e a_i is the maximum of $a_1, ...a_n$. The second case, in which the algorithm returns Δ at line number (10) contradicts the fact that n is finite, and left as an exercise. the running time is $O(n^2)$ and the space consumption is $O(n)$.*

Loop Invariant In The Cleverer Alg. *Consider now the linear time algorithm:*

Result: returns the maximum of $a_1 ... a_n \in \mathbb{R}^n$

```
1
2  let b ← a_1
3
4  for i ∈ [2, n] do
5  |   b ← max(b, a_i)
6  end
7  return b
```

Algorithm 2: maximum alg.

What is the Loop Invariant here? "at the i-th iteration, $b = \max\{a_1 ... a_{i-1}\}$". The proof is almost identical to the naive case.

1.2 Non-Linear Space Complexity Algorithms.

Sub-Array Maximum. Consider the leading example; It's easy to write an algorithm that answers the queries at a total time of a $O(|Q| \cdot n)$ by answers separately on each query. Can we achieve a better upper bound?

Sub-Array. $O(n^2)$ space alg.

Claim. Consider the outer loop at the k-th step. The following is a loop invariant:

$$for\ every\ k' < k,\ s.t\ i + k' \leq n \Rightarrow A_{i,i+k'} = \max\{a_i, a_{i+1}, ..., a_{i+k'}\}$$

Proof. The initialization condition trivially holds, assume by induction that $A_{i,i+k-1} = \max\{a_i ... a_{i+k-1}\}$ at beginning of k iteration. By the fact that $\max(x, y, z) = \max(\max(x, y), z)$ we get that

$$\max\{a_1 ... a_{i+k-1}, a_{i+k}\} = \max\{\max\{a_1 ... a_{i+k-1}\}, a_{i+k}\} = \max\{A_{i,i+k-1}, a_{i+k}\}$$

And the right term is exactly the value which assigned to $A_{i,i+k}$ in the end of the$k$-th iteration. Thus in the beginning of $k + 1$ iteration the property is still conserved.

$O(n \log n)$ **Space Solution.** Example for $O(n \log n + |Q| \log n)$ time and $O(n \log n)$ space algorithm. Instead of storing the whole matrix, we store only logarithmic number of rows.

Sub-Array. $O(n \log n)$ **space alg.**