ter 1

ectness - Recitation 2

orithms correctness is necessary to guarantee that our code computes its goal for every given input. In that recitation, we will examine se eirs running time and memory consumption, and prove they are correct.

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 $(i, j) \in [n$ hat calculates the $\max_{i \leq k \leq j} a_k$.*

rrectness And Loop Invariant.

ss. We will say that an algorithm $\mathcal{A}$ (an ordered set of operations) computes $f : D_1 \rightarrow D_2$ if for every $x \in D_1$ the following equality hol when it's obvious what is the goal function f, we will abbreviate and say that $\mathcal{A}$ is correct.
es of functions f might be: file saving, summing numbers, or posting a message in the forum.

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

lization. The property holds (even) before the first iteration of the loop.

ervation. As long as one performs the loop iterations, the property still holds.

onal) Termination. Exiting from the loop carrying information.

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

: returns the maximum of $a_1...a_n \in \mathbb{R}^n$
on *max(* $a_1..a_n$ *)*
$i \in [n]$ **do**
$j \leftarrow 1$

while $j \leq [n]$ *and* $a_i \geq a_j$ **do**
| $j \leftarrow j + 1$
end

if $j = n + 1$ **then**
| **return** a_i
end
l
urn Δ

Algorithm 1: naive maximum alg.

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.*

nitialization condition holds, as the at the first iteration $j = 1$ and therefore the property is trivial. Assume by induction, that for every ct, 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 th ls that $a_i \geq a_{j-1}, a_{j-2}, ...a_1$.

oof. *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 ho* $i \Rightarrow a_i \geq a_{j'}$ *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 fa an exercise. the running time is $O(n^2)$ and the space consumption is $O(n)$.*

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

rns the maximum of $a_1...a_n \in \mathbb{R}^n$
ax($a_i..a_j$)

1

$2, n]$ **do**
max (b, a_i)

Algorithm 2: maximum alg.

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

inear Space Complexity Algorithms.

imum. 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 an we achieve a better upper bound?

t the $\max\{a_i...a_j\}$ for each query $(i, j) \in Q$
ax($a_i...a_j$)
$\mathbb{M}^{n \times n}$

$n]$ **do**
$- a_i$

$[1, n]$ **do**
$\in [n]$ **do**
$i + k \leq n$ **then**
$A_{i,i+k} \leftarrow \max(A_{i,i+k-1}, a_{i+k})$
nd

Q **do**
$- q$
$A_{i,j}$

Algorithm 3: Sub-Array. $O(n^2)$ space alg.

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

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

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 th $x, y), z)$ we get that

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

ght 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

) **Space Solution.** Example for $O(n \log n + |Q| \log n)$ time and $O(n \log n)$ space algorithm. Instead of storing the whole matri number of rows.

```
: print the max{a_i...a_j} for each query (i, j) ∈ Q
on max( a_i...a_j )
A ← M^{n×log n}

 i ∈ [n] do
 A_{i,1} ← a_i
d

 k ∈ [2, 4, .., 2^m, ..., n] do
 for i ∈ [n] do
     if i + k ≤ n then
         A_{i,k} ← max(A_{i,k/2}, A_{i+k/2,k/2})
     end
 end
d

 q ∈ Q do
 i, j ← q
 decompose j − i into binary representation 2^{t_1} + 2^{t_2} + .. + 2^{t_l}
 print max{A_{i,2^{t_1}}, A_{i+2^{t_1},2^{t_2}}, ..., A_{i+2^{t_1}+2^{t_2}+..2^{t_{l-1}},2^{t_l}}}
d
```

Algorithm 4: Sub-Array. $O(n \log n)$ space alg.