1 Minimum Excludant[1]

Consider the following problem: given an array of integers, find the smallest **non-negative** integer that is not present in the array. For instance, if the array is $\begin{array}{|c|c|c|c|c|c|}\hline -3 & 1 & 4 & -7 & 0 & 1 \\ \hline\end{array}$, then the answer is 2.

In what follows, we propose four different algorithms to solve this problem. The implementations are given in Java, but should be straightforward to translate to any programming language with arrays and integers. For each implementation, the time and space complexity are given (in terms of the array size N), but you do not have to prove them.

Version 0: Naive Implementation

This is a straightforward, naive implementation with time complexity $\mathcal{O}(N^2)$ and space complexity $\mathcal{O}(1)$.

```java
static int mex0(int[] a) {
  int n = a.length;
  for (int v = 0; v < n; v++) {
    int i = 0;
    while (i < n && a[i] != v)
      i++;
    if (i == n)
      return v;
  }
  return n;
}
```

Tasks for Version 0

(a) Verify the naive implementation for safety.

(b) Verify the naive implementation for correctness.

Version 1: Boolean Marks

This is a more efficient implementation, using an array of Boolean marks. Time complexity $\mathcal{O}(N)$ and space complexity $\mathcal{O}(N)$.

```java
static int mex1(int[] a) {
  int n = a.length;
  boolean[] seen = new boolean[n];
  for (int x: a)
    if (0 <= x && x < n)
      seen[x] = true;
  int r = 0;
  while (r < n && seen[r])
    r++;
```

[1] We warmly thank Jean-Christophe Filliâtre from CNRS for contributing the idea for this challenge.

```
  return r;
}
```

Tasks for Version 1

(c) Verify the Boolean marks implementation for safety.

(d) Verify the Boolean marks implementation for correctness.

Version 2: Mutated Array

This is an even more efficient implementation, *assuming we are allowed to mutate the array.* Time complexity $\mathcal{O}(N)$ and space complexity $\mathcal{O}(1)$.

```
static void swap(int[] a, int i, int j) {
  int tmp = a[i]; a[i] = a[j]; a[j] = tmp;
}

static int mex2(int[] a) {
  int n = a.length;
  int i = 0;
  while (i < n) {
    int x = a[i];
    if (x < 0 || x >= n || a[x] == x)
      i++;
    else if (x < i)
      swap(a, i++, x);
    else
      swap(a, i, x);
  }
  for (i = 0; i < n; i++)
    if (a[i] != i)
      return i;
  return n;
}
```

Tasks for Version 2

(e) Verify the mutated array implementation for safety.

(f) Verify the mutated array implementation for correctness.

Version 3: Sorted Array

Assuming the input array is sorted, we can achieve time complexity $\mathcal{O}(N)$ and space complexity $\mathcal{O}(1)$ without mutating the array.

```
static int mex3(int[] a) {
  int n = a.length;
  int last = -1;
  for (int i = 0; i < n; i++) {
    if (a[i] >= last + 2)
      return last + 1;
    if (a[i] >= 0)
      last = a[i];
    assert last < n;
  }
  return last + 1;
}
```

Tasks for Version 3

(g) Verify the sorted array implementation for safety.

(h) Verify the sorted array implementation for correctness.