# HW2: Find the missing element

**Describe your algorithm in words：**

We will apply a modified binary search to find the missing element within the given arithmetic sequence A. We assume the element in A is a[1], … , a[n].

***Step 1***: Considering that the missing one element is not at the beginning or end, we calculate the common difference d by (a[n]- a[1])/n.

***Step 2***: Apply the binary search adaptation, we start with the whole sequence as the search interval, lower bound L is set to be 1, and upper bound R is set to be n. The middle index m = (L+R)/2 by using integer division, the expected value at the middle index "m" can be calculated as a[1] + d*(m-1).

To find the missing element, compare a[m] with the expected value. If they match, the missing element is in the right part, lower bound L needs to be updated to m. Otherwise, the missing element is in the left part, and the upper bound R needs to be updated to m.

Repeat this process until the search interval is narrowed down to a 1 (L +1 = R), the missing element is a[L]+d.

***Step3***: Return the missing element.

**Justify its correctness ：**

1) The variation range of (a[n]- a[1]) is made up of (n+1) elements when the missing element is not at the beginning or end, so the common difference is (a[n]- a[1])/n.
2) The adapted binary search can be used in the sorted progression. A is a sorted sequence, so the search space could be narrowed down to half each time.
3) If the missing element value is on the left of the middle, the middle element's value should be bigger than the expected one. Thus, by comparing the middle element's value with the expected one, we can identify whether the missing element is on the right of the middle element or the left of it.
4) We update the middle index m to L when we suspect the missing element is in the right half. This is different from a standard search approach, where m would be set to L+1. By doing this, we can avoid the circumstance where the missing element is located between a[L] and a[L+1].
5) When the search interval is narrowed down to a 1, it means that the missing element is not located before L or after R. The loop ends here, and the missing element is a[L] + d. This is because a[L] is always located in the right location, the missing element is between a[L] and a[L+1], so it should be a[L]+d.

**Analyze and state its runtime. ：**

The binary search has a running time of O(logn).

Each iteration of the binary search involves constant-time arithmetic operations, such as calculating the middle index and computing the expected value, which takes O(1) running time.

Therefore, the total running time is dominated by the binary search process, which takes O(logn).