Homework Assignment #2
Due: January 24, 2020 at 2:30 p.m.

1. In olden days, people like your grandparents used to use printed books called dictionaries that listed words in alphabetical order, together with their definitions. It would be possible to look for a word in the dictionary using binary search, but this is not what people typically did. For example, to look up the word `confusticate`, instead of starting by looking at the middle of the dictionary, one might start by looking at a page that is about 15% of the way into the dictionary, since the letter c comes near the beginning of the alphabet. If the first word on that page is `catarrh`, then one might just flip a small number of pages forward to get closer to `confusticate`.

Willemina uses this idea as inspiration to design an algorithm to search for an integer k in a sorted array $A[1..n]$ of integers. She is hoping that it will outperform binary search if the values in the array are somehow uniformly distributed so that she can estimate roughly where in the array the desired key lies. As usual, the postconditions are that the algorithm does not modify the array A and

- if k appears in $A[1..n]$ then the algorithm returns an index i such that $A[i] = k$, and
- if k does not appear in $A[1..n]$ then the algorithm returns "not found".

As in binary search, the algorithm keeps track of a subarray $A[lo..hi]$. Instead of comparing k to the middle element of that subarray, she estimates where in the subarray the key k ought to be and compares k with that element. Willemina estimates that the fraction of elements in $A[lo..hi]$ that are less than k should be about $\frac{k-A[lo]}{A[hi]-A[lo]}$. She writes down the following algorithm.

```
1   SEARCH(k, A[1..n])
2       Precondition: n is a positive integer and A[1] ≤ A[2] ≤ ··· ≤ A[n]
3       lo ← 1
4       hi ← n
5       loop
6           exit when lo = hi
7           m ← lo + ⌊ (k−A[lo])/(A[hi]−A[lo]) · (hi − lo) ⌋
8           if k > A[m] then lo ← m + 1
9           else hi ← m
10          end if
11      end while
12      if A[hi] = k then return hi
13      else return "not found"
14      end if
15  end SEARCH
```

(a) Willemina implements the algorithm in Java and starts testing it.

(i) Give an example input where Willemina's algorithm generates an array out of bounds error.

(ii) Give an example input where Willemina's algorithm generates a division by 0 error.

(iii) Give an example input where Willemina's algorithm does not terminate.

(b) Modify the exit condition on line 6 so that Willemina's algorithm becomes a correct search algorithm. You should not modify or add any other line.

Hint: Think about how you can design the exit condition to eliminate *all* of the errors identified in part (a) and still give the right result when the code reaches line 12.

(c) State a loop invariant that can be used to prove the algorithm is correct.

(d) Prove your answer to part (c) really is a loop invariant.

(e) Prove that the algorithm terminates and satisfies its postconditions.

(f) What is the worst-case running time of the algorithm? State your answer in terms of n using Θ notation.

(g) Prove your answer in part (g) is correct. For full marks, you have to prove both the upper bound and a matching lower bound on the worst-case complexity.