The algorithms and analysis here are adapted from the reading, *Algorithms Unlocked* by Cormen.

An algorithm for searching a sorted array

Algorithm 1 An algorithm for searching a *sorted* array

```
1: procedure BinarySearch(A, n, x)          ▷ Find x in sorted array A of length n
2:     Set p to 1 and r to n.
3:     while p ≤ r do
4:         Set q to ⌊(p + r)/2⌋.
5:         If A[q] = x, then return q.
6:         Otherwise, if A[q] > x, then set r to q − 1.
7:         Otherwise, it must be that A[q] < x, so set p to q + 1.
8:     end while
9:     Return Not-Found.
10: end procedure
```

Correctness:

- Essential *invariant*: if x is anywhere in A, it will be between p and r.
- Each time through the while loop, either p or r is updated to maintain this invariant.
- Each time through, p and r get closer together and eventually will be equal to one another. This will either be the location of x (if it's there) or, in the next step, the algorithm will conclude that x is not there.

Runtime analysis: binary search is $O(\lg n)$

- How many times can we repeat the while loop?
- As long $p \leq r$. In other words, as long as the subarray $A[p \ldots r]$ still needs to be checked.
- Initially, the subarray $A[1 \ldots n]$ has n cells.
- Each time through the loop, the size of the subarray is cut in half.
- How many times can you cut n in half until you get down to 1? $\lg n$

Sorting algorithms

Useful resource: `https://visualgo.net/sorting`

Algorithm 2 An algorithm for sorting an array

```
1: procedure SelectionSort(A, n)   ▷ Given array A of length n, rearrange elements into nondecreasing order
2:     for i = 1 to n − 1 do
3:         Set smallest to i
4:         for j = i + 1 to n do
5:             If A[j] < A[smallest], then set smallest to j.
6:         end for
7:         Swap A[i] with A[smallest].
8:     end for
9: end procedure
```

Algorithm 3 An algorithm for sorting an array

```
1: procedure InsertionSort(A, n)   ▷ Input and result is same as SelectionSort.
2:     for i = 2 to n do
3:         Set key to A[i].   ▷ Copy A[i] into key so we don't lose it.
4:         Set j to i − 1.
5:         while j > 0 and A[j] > key do
6:             Set A[j + 1] to A[j].
7:             Decrement j (i.e., set j to j − 1).
8:         end while
9:         Set A[j + 1] to key.
10:    end for
11: end procedure
```

Algorithm 4 An algorithm for sorting an array

```
1: procedure MergeSort(A, p, r)   ▷ Result: the elements of subarray A[p . . . r] are sorted.
2:     If p ≥ r, then subarray has at most one element, so it's already sorted. Return without doing anything.
3:     ▷ Otherwise, do the following
4:     q = ⌊(p + r)/2⌋
5:     Recursively call MergeSort(A, p, q)
6:     Recursively call MergeSort(A, q + 1, r)
7:     Call Merge(A, p, q, r).   ▷ See book for details of Merge.
8: end procedure
```