Binary Search

- Consider an array $a_1, a_2, \ldots, a_n$ of numbers sorted in increasing order, and suppose we want to know if a particular number b appears in the array. The following algorithm returns *True* if b appears, and returns *False* otherwise.

```
1: BinarySearch(a_1, a_2, ..., a_n : array of reals, b : real)
2:   lo := 1; hi := n
3:   while lo <= hi
4:     mid := ⌊(lo + hi)/2⌋
5:     if a_mid = b then return True
6:     else if a_mid < b then lo := mid + 1
7:     else hi := mid – 1
8:   end while
9:   return False
```

- *Explanation*: Look at the mid-point and if $a_{\text{mid}} = b$ (line 5) then we return *True*. Otherwise, if $a_{\text{mid}} < b$ (line 6) then look in the range $a_{\text{mid+1}}, \ldots, a_n$ and if $a_{\text{mid}} > b$ (line 7) then look in the range $a_1, \ldots, a_{\text{mid}-1}$

Running time analysis of Binary Search

- Clearly the time before and after the loop is $O(1)$, and the loop body takes $O(1)$ time per iteration.

- Question: How many loop iterations are there? Clearly, this depends on the gap between "hi" and "lo". We can write a *recurrence* for the number of loop iterations $L(n)$ when the gap between "hi" and "lo" is n:

$$L(0) = 1$$
$$L(n) \leq 1 + L(\lfloor n/2 \rfloor)$$

- Thus $L(n)$ is $O(\log n)$ and hence the running time of BinarySearch is

$O(1) + O(\log n).O(1) + O(1)$ which is $O(\log n)$

Sorting

- Suppose we want to *sort* an array a_1, a_2, …, a_n of numbers in increasing order. We will look at two simple (but slow) strategies: Insertion Sort and Bubble Sort. Both these strategies take $O(n^2)$ time in the worst case.

```
InsertionSort(a_1, a_2, ..., a_n : array of reals)
  for j := 2 to n
    // insert a_j into the right location in a_1, ..., a_i
    temp := a_j
    i := 1
    while(a_j > a_i)
      i := i + 1
    end while
    // now move a_{i+1}, ..., a_j one position to the right
    for k := j down to i + 1
      a_k := a_{k-1}
    end for
    a_i := temp
  end for
```

Analysis of Insertion Sort

- Suppose we want to *sort* an array $a_1, a_2, \ldots, a_n$ of numbers in increasing order. We will look at two simple (but slow) strategies: Insertion Sort and Bubble Sort. Both these strategies take $O(n^2)$ time in the worst case.

```
InsertionSort(a_1, a_2, ..., a_n : array of reals)
  for j := 2 to n
    // insert a_j into the right location in a_1, ..., a_i
    temp := a_j
    i := 1
    while(a_j > a_i)
      i := i + 1
    end while
    // now move a_{i+1}, ..., a_j one position to the right
    for k := j down to i + 1
      a_k := a_{k-1}
    end for
    a_i := temp
  end for
```