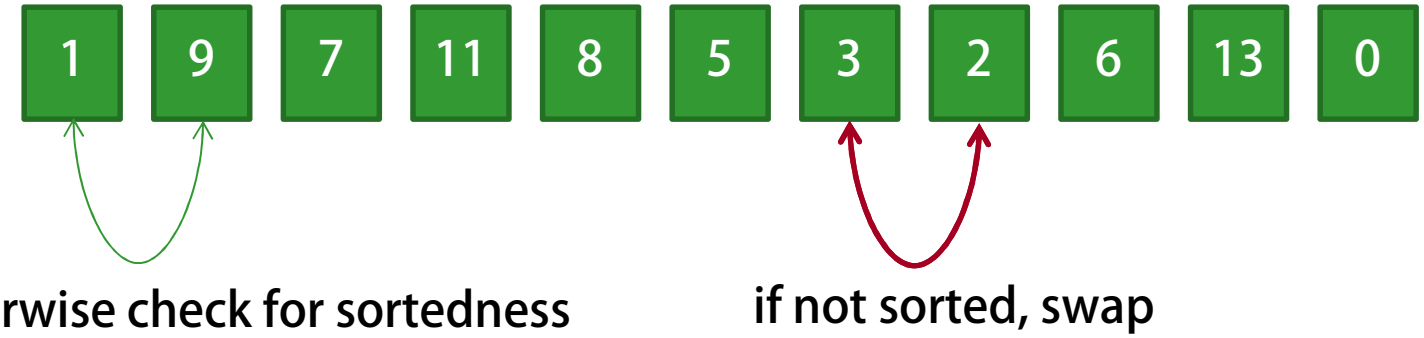
Lecture 9: Sorting and Invariants

Armando Solar-Lezama

MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

Bubble Sort



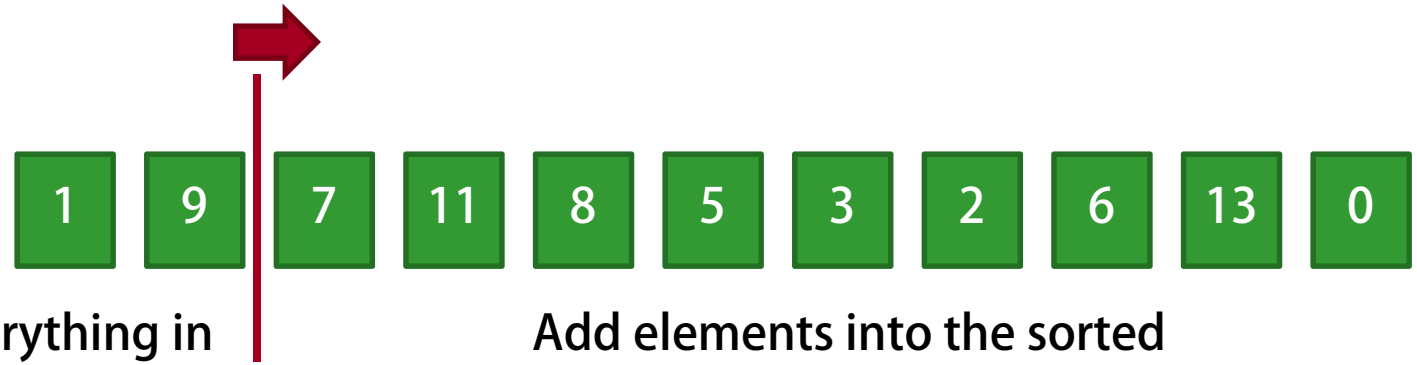
bubbleSort is $O(n^2)$

Bubble Sort

```
def bubbleSort(L):
    doMore = True
    while(doMore):
        doMore = False
        for i in xrange(0, len(L)-1):
            if(L[i] > L[i+1]):
                swap(L, i, i+1)
                doMore = True

    return L
```

Insertion Sort



insertionSort is also $O(n^2)$

Insertion Sort

```
def insertionSort(L):
    for end in xrange(1, len(L)):
        #what do we know here?
        #assert isSorted(L[0:end])
        temp = L[end]
        for i in xrange(0, end):
            if(temp < L[i]):
                temp2 = L[i]
                L[i] = temp
                temp = temp2
        L[end] = temp
        #what do we know here?
        #assert isSorted(L[0:end+1])
return L
```

Invariant

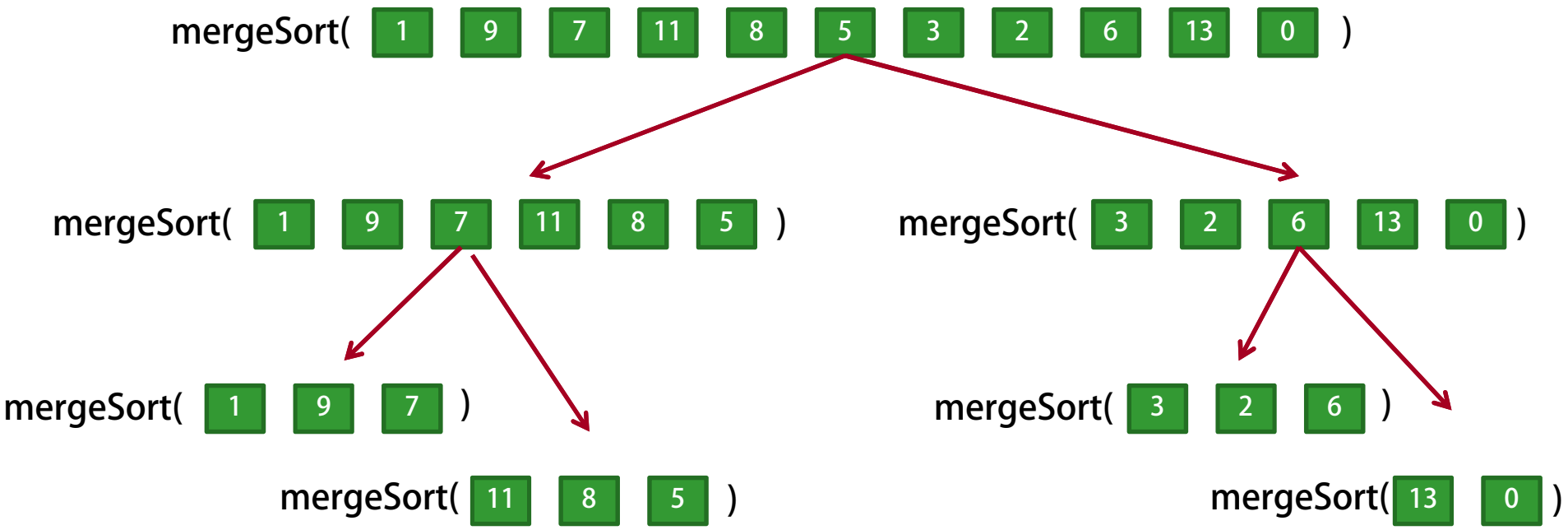
A property of the state that is true every time a program passes a particular point

Ex:

- at the beginning of every iteration of the outer loop in insertion sort, L[0:end] will be sorted

Invariants are essential in making correctness arguments

Merge Sort



Every level of calls to merge sort involves a linear time merge
There are $\log_2(N)$ levels of calls to merge sort, so the algorithm is $O(N\ log(N))$ where N is the length of the array

Merge Sort

```
def mergeSort(L):
    if(len(L) <= 1):
        return L
    mid = len(L) / 2
    L1 = mergeSort(L[0:mid])
    L2 = mergeSort(L[mid:])
    return merge(L1, L2)
```

Merge

```
def merge(L1, L2):
    result = []
    p1 = 0
    p2 = 0
    for i in xrange(0, len(L1) + len(L2)):
        if(p1 < len(L1) and p2 < len(L2)):
            if( L1[p1] < L2[p2] ):
                result.append(L1[p1])
                p1 += 1
            else:
                result.append(L2[p2])
                p2 += 1
        else:
            if p1 < len(L1):
                result.append(L1[p1])
                p1 += 1
            else:
                result.append(L2[p2])
                p2 += 1

    return result
```

Run the examples

You can find all the code here

- http://bit.ly/WYeUbO

You can find a random list of 10K words here

- http://bit.ly/XVrnLh
- Use it to time the different sort algorithms
- How does the performance compare?