CMPSC 465

Chen Sun

cxs1031@psu.edu

- Recitation Section: 2, 3, 6, 7
- Email: cxs1031@psu.edu
- Office Hour: W, F 10:30-11:30
- Location: 506 Wartik Building

- Having problem? go to piazza first!

Review

```
INSERTION-SORT(A)
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the sorted sequence A[1 .. j − 1].
4       i = j − 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i − 1
8       A[i + 1] = key
```

Loop invariant

- At the start of each iteration of the for loop of line 1-8, the subarray A[1..j-1] consists of the elements originally in A[1..j-1], but in sorted order.

Initialization

- When j= 2, the subarray A[1..j-1], consists of the single element A[1], which is the original element in A[1]
- The subarray is sorted because there is only one element.

Maintenance

- Assume A[1..j-1] consists of the elements originally in A[1..j-1] but in sorted order.
- Considering A[1..j], the body of the for loop works by moving A[j-1], A[j-2], A[j-3] and so on by one position to the right until it finds the proper position for A[j], at which point it inserts the value of A[j].
- The subarray A[1..j] then consists of the elements originally in A[1..j], but in sorted order.

Termination

- When the loop terminates, j > A.length = n. And because each loop iteration increases j by 1, we have j=n+1.
- Substituting n+1 for j in the loop invariant, we have that subarray A[1..n] consists of the elements originally in A[1..n], but in sorted order.
- Since subarray A[1..n] is the entire array, we conclude that the entire array is sorted.

Binary Search

```
def binary_search(A, target):
    lo = 0
    hi = len(A) - 1
    while lo <= hi:
        mid = (lo + hi) / 2
        if A[mid] == target:
            return mid
        elif A[mid] < target:
            lo = mid + 1
        else:
            hi = mid - 1
```

Assumption:

1. target is in A
2. A is sorted

Loop Invariant

- At each step of the while loop, lo and hi surrounded the actual location of where target is.

Initialization

- When the algorithm begins, lo = 0 and hi = len(A)-1, lo and hi enclose all values, lo and hi surrounded the target.

Maintenance

- Suppose in previous iteration, lo and hi surrounded the actual location of where target is.
- In current iteration, there are 2 cases:
  - Case 1: If **A[mid] > target**, then the target must be between lo and mid. We update hi = mid-1
  - Case 2: If **A[mid] < target**, then the target must be between mid and hi. We update lo = mid+1
- In either cases, we preserve the loop invariant for the next iteration.

Termination

- For each iteration, lo always increase and hi always decrease.
- Loop stop in two cases:
  - Case 1: **A[mid] == target**, we find target location = mid.
  - Case 2: **lo=hi**. According to loop invariant, target location is surrounded by lo and hi, (i.e. A[lo]<=target<=A[hi]), so we also find the target location = lo = hi.
- So when loop stop, target is found.