Binary search

- Can we do better than `O(len(L))` for search?
- If know nothing about values of elements in list, then no.
- Worst case, would have to look at every element

What if list is ordered?

- Suppose elements are sorted in ascending order

```
def search(L, e):
    for i in range(len(L)):
        if L[i] == e:
            return True
        if L[i] > e:
            return False
    return False
```

- Improves average complexity, but worst case still need to look at every element

Use binary search

1. Pick an index, `i`, that divides list in half
2. Ask if `L[i] == e`
3. If not, ask if `L[i]` larger or smaller than `e`
4. Depending on answer, search left or right half of `L` for `e`

A new version of a divide-and-conquer algorithm

- Break into smaller version of problem (smaller list), plus some simple operations
- Answer to smaller version is answer to original problem

Binary search

```
def search(L, e):
    def bSearch(L, e, low, high):
        if high == low:
            return L[low] == e
        mid = low + int((high - low)/2)
        if L[mid] == e:
            return True
        if L[mid] > e:
            return bSearch(L, e, low, mid - 1)
        else:
            return bSearch(L, e, mid + 1, high)

    if len(L) == 0:
        return False
    else:
        return bSearch(L, e, 0, len(L) - 1)
```

Analyzing binary search

- Does the recursion halt?
  - Decrementing function
    1. Maps values to which formal parameters are bound to non-negative integer
    2. When value is <= 0, recursion terminates
    3. For each recursive call, value of function is strictly less than value on entry to instance of function
  - Here function is high – low
    - At least 0 first time called (1)
    - When exactly 0, no recursive call, returns (2)
    - Otherwise, halt or recursively call with value halved (3)
- So terminates

Analyzing binary search

- What is complexity?
  - How many recursive calls? (work within each call is constant)
  - How many times can we divide `high - low` in half before reaches 0?
  - $\log_2$`(high - low)`
  - Thus search complexity is `O(log(len(L)))`