Prolegomena unit outline:

- Algorithms and correctness (last week Wednesday and Friday)
- Algorithms and efficiency (today and Friday)
- Abstract data types (next week Monday)
- Data Structures (next week Wednesday and Friday)

Today and Friday:

- ▶ Go over Ex 1.(6 & 7)
- The general meaning of efficiency
- ▶ The analyses of bounded linear search, binary search, and selection sort

◆□▶ ◆□▶ ◆注▶ ◆注▶ 注 のへで

- The precise meaning of big-oh, big-theta, and big-omega
- The costs of elemental algorithms
- The analysis of quick sort

**1.6** Write a loop invariant to capture the relationships among sequence, smallest\_so\_far, smallest\_pos, and i in the following algorithm to find the smallest element in a sequence.

◆□▶ ◆□▶ ◆三▶ ◆三▶ ○○ ○○○

```
def find_smallest(sequence):
    smallest_so_far = sequence[0]
    smallest_pos = 0
    i = 1
    while i < len(sequence) :
        if sequence[i] < smallest_so_far :
            smallest_pos = i
            smallest_so_far = sequence[i]
        i += 1
    return smallest_pos</pre>
```

**1.7** State and prove a loop invariant to show that the following loop clears the list sequence, that is, it sets all of its positions to None. Your loop invariant should explain and relate the variables sequence and i.

◆□▶ ◆□▶ ◆三▶ ◆三▶ ○○ ○○○

```
i = 0
while i < len(sequence):
    sequence[i] = None
    i += 1</pre>
```

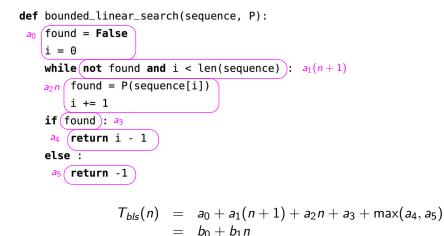
From the correctness proof of bounded\_linear\_search:

By Invariant 1.c [i is the number of iterations], after at most n iterations, i = n and the guard will fail.

From the correctness proof of binary\_search (rewritten):

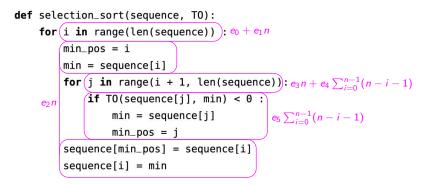
Let *i* be the number of iterations completed. Suppose  $i \ge \lg n$ . Then  $2^i \ge n$  and  $\frac{n}{2^i} \le 1$ . By Invariant 3.b, [high  $-\log \le \frac{n}{2^i}$ ], we have high  $-\log \le 1$  and the guard fails.

▲ロト ▲圖ト ▲画ト ▲画ト 三回 - のへで



```
def binary_search(sequence, T0, item):
   low = 0
Cn
    high = len(sequence)
    while high - low > 1): c_1(\lg n + 1)
  c_2 \lg n \mod = (low + high) / 2
        compar = TO(item, sequence[mid])
        if compar < 0 : # item comes before mid</pre>
             high = mid
        elif compar > 0 : # item comes after mid
             low = mid + 1
        else :
                             # item is at mid
             assert compar == 0
            low = mid
             high = mid + 1
    if (low < high and TO(item, sequence[low]) == 0): c3
     c<sub>4</sub> (return low)
    else :
     c<sub>5</sub> (return -1)
```

$$T_{bs}(n) = c_0 + c_1(\lg n + 1) + c_2 \lg n + c_3 + \max(c_4, c_5)$$
  
=  $d_0 + d_1 \lg n$ 



$$T_{sel}(n) = f_1 + f_2 n + f_3 n^2$$

▲ロト ▲圖ト ▲画ト ▲画ト 三回 - のへで

- ∃ T : D → N relating input to running time on some platform. Interpret the codomain N as natural numbers in some unit time.
- → A T<sub>absolute</sub> : N → N relating input size to running time on some platform.
  Interpret the domain N as the number of items in the list (or other structure, for
  other algorithms).
- ▶  $\exists T_{worst} : \mathbb{N} \to \mathbb{N}$  relating input size to the maximum running time on some platform for all inputs of the given size.
- ∃ T<sub>best</sub> : N → N relating input size to the minimum running time on some platform for all inputs of the given size.
- ▶  $\exists T_{expected} : \mathbb{N} \to \mathbb{N}$  relating input size to the expected value of the running time on some platform over all inputs of the given size.

▲ロト ▲圖ト ▲画ト ▲画ト 三回 - のへで

What is big-oh notation?

Big-oh is a way to categorize *functions*:

O(g) is the set of functions that can be bounded above by a scaled version of g.

f(n) = O(g(n)) (or, more properly  $f \in O(g)$ ) means

 $\exists c, n_0 \in \mathbb{N}$  such that  $\forall n \in [n_0, \infty), f(n) \leq cg(n)$ 

Objections to and misconceptions of big-oh notation take forms such as

- Big-oh notation specifies only an upper bound of running time, which might be widely imprecise.
- Big-oh notation measures only the worst case, when the best case or the typical case might be much better.
- Big-oh ignores constants, which can greatly affect running time in practice.
- Algorithms that have the same big-oh category can have widely different running times in practice.
- Big-oh considers only the *size* of the input, when in fact other attributes of the input can greatly affect running time.

◆□▶ ◆□▶ ◆三▶ ◆三▶ ○○ ○○○

## Coming up:

Due Today: Finish reading Section 1.2, if you haven't already. (Exercises 1.(6 & 7) and the quiz should have been done before class)

▲□▶ ▲圖▶ ▲필▶ ▲필▶ 三里

```
Due Fri, Jan 21:
Read Sections 1.(3 & 4)
Do Exercises 1.(27, 28, 42, 43)
Take quiz
```

Due Tues, Jan 25: Read Section 2.1 Do Exercise 1.11 Take quiz