Invariant (Outer loop of `selection_sort`)

(a) *The range* $[0, i)$ *in* `sequence` *is sorted.*

(b) *All the elements in range* $[0, i)$ *in* `sequence` *are less than or equal to all the elements in the range* $[i, n)$.

(c) $0 \le i \le n$.

Invariant (Inner loop of `selection_sort`)

(a) $\texttt{sequence}[\texttt{min\_pos}] = \texttt{min}$.

(b) *`min` is the smallest element in the range* $[i, j)$. *(Formally:* $\forall\ k \in [i, j), \texttt{min} \le \texttt{sequence}[k]$.*)*

(c) $i \le \texttt{min\_pos} < j \le n$.

Correctness claim (`selection_sort`.)

After n iterations, `sequence` *is sorted and* `selection_sort` *returns.*

```
def bounded_linear_search(sequence, P):
    found = False
    i = 0
    while not found and i < len(sequence) :
        found = P(sequence[i])
        i = i + 1
    if found :
        return i - 1
    else
        return -1
```

a_1

$a_2(n+1)$

$a_3 n$

a_4

a_5

a_6

$$
\begin{aligned}
T_{bls}(n) &= a_1 + a_2(n+1) + a_3 n + a_4 + \max(a_5, a_6) \\
&= b_0 + b_1 n
\end{aligned}
$$

```
def binary_search(sequence, TO, item):
    low = 0                                               c1
    high = len(sequence)
    while high - low > 1 :                                c2
        mid = (low + high) / 2                            c3
        compar = TO(item, sequence[mid])
        if compare < 0:
            high = mid                                    c4
        elif compar > 0:                                  c5
            low = mid + 1                                 c6
        else:
            low = mid                                     c7
            high = mid + 1
    if low < high and TO(item, sequence[low]) == 0:       c8
        return low                                        c9
    else:
        return -1                                         c10
```

$$
\begin{aligned}
T_{bs}(n) &= c_1 + c_2(\lg n + 1) + (c_3 + \max(c_4, c_5 + c_6, c_5 + c_7)) \lg n \\
&\quad + c_8 + \max(c_9, c_{10}) \\
&= d_0 + d_1 \lg n
\end{aligned}
$$

```
def selection_sort(sequence, TO):
    for i in range(len(sequence)):
        min_pos = i
        min = sequence[i]
        for j in range(i + 1, len(sequence)):
            if TO(sequence[j], min) < 0:
                min = sequence[j]
                min_pos = j
        sequence[min_pos] = sequence[i]
        sequence[i] = min
```

$e_1 + e_2 n$

$e_3 n$

$e_4 n + e_5 \sum_{i=0}^{n-1}(n-i)$

$e_6 \sum_{i=0}^{n-1}(n-i-1)$

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

Every time you run a program, you are performing a scientific experiment that relates the program to the natural world and answers one of our core questions: How long will my program take? The running time [of many programs] is relatively insensitive to the input itself; it depends primarily on the problem size.

Sedgewick, pg 173

$g(n) \sim f(n)$ means the functions are asymptotically *equal*, that is, that $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 1$. Thus $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} \sim \frac{n^3}{6}$.

$g(n) = O(f(n))$, which really should be written $g(n) \in O(f(n))$, means that a scaled version of $f(n)$ asymptotically *bounds* g above. It means there exists a c such that when n is large enough, $g(n) \leq cf(n)$. Thus $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(\frac{n^3}{6})$ but also $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(n^3)$ and $\frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3} = O(n^4)$.

With big-oh, you can throw away the lower ordered terms *and* throw away the constant factor of the highest order term *and* overshoot.

With tilde, you only can throw away the lower ordered terms.