Ex. A linear search or sequential search is a method for finding an element within a list.

This program searches an array A[0...n-1] to find element x believed to be in the arrays.

1. \( i = 0 \)
2. while (\( x \neq A[i] \))
3. \( i++ \)

Sample Statement:

```
Sample Statement  
\( i = 0 \)  
```

At most n times around

```
\( O(1) \)  
\( O(1 \times n) = O(n) \)  
\( O(1) \)  
```
The running time of the entire program fragment is $O(n)$

Note: This $O(n)$ program can be replaced by $O(\log n)$ program using the so-called Binary Search.

**Sorting Algorithm** - puts elements in list in a certain order

Ex. Selection sort is a sorting algorithm (in-place sort)

<table>
<thead>
<tr>
<th>UNSORTED SUBLIST</th>
<th>SMALLEST ELEMENT IN LIST</th>
<th>SORTED SUBLIST</th>
</tr>
</thead>
<tbody>
<tr>
<td>(11, 25, 12, 22, 64)</td>
<td>11</td>
<td>()</td>
</tr>
<tr>
<td>(25, 12, 22, 64)</td>
<td>12</td>
<td>(11)</td>
</tr>
<tr>
<td>(25, 22, 64)</td>
<td>22</td>
<td>(11, 12)</td>
</tr>
<tr>
<td>(25, 64)</td>
<td>25</td>
<td>(11, 22)</td>
</tr>
<tr>
<td>(64)</td>
<td>64</td>
<td>(11, 22, 25)</td>
</tr>
<tr>
<td>()</td>
<td></td>
<td>(11, 22, 25, 64)</td>
</tr>
</tbody>
</table>
Selection-sort program fragment

```c
for (i = 0; i < n-1; i++)
    small = i
    for (j = i+1; j < n; j++)
        if (A[j] < A[small])
            small = j;
    swap = A[small];
    A[small] = A[i];
    A[i] = swap
```

Running time = two loops

Structure Tree

\[ O(n) \cdot O(n-1) = O(n^2) \]
Conclusions:

1) Linear search has linear complexity.
2) Selection/sort has quadratic complexity.
Analyzing Programs w| Function Calls

Recursive Function - calls itself
Non-recursive - does NOT call itself

1. Evaluate running times of functions that do NOT call any other function

2. Evaluate running times of ONLY functions whose running time we've already determined

3. Repeat

Homework: