Recursion: Computational Complexity

Recursively Move Smallest to Front

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Recursion.minToFront(t.subList(1, t.size()));
    int first = t.get(0);
    int second = t.get(1);
    if (second < first) {
      t.set(0, second);
      t.set(1, first);
    }
  }
}
```

size of problem, n, is the number of elements in the list `t`

Estimating complexity

- the basic strategy for estimating complexity:
  1. for each line of code, estimate its number of elementary instructions
  2. for each line of code, determine how often it is executed
  3. determine the total number of elementary instructions

Elementary instructions

- what is an elementary instruction?
  - for our purposes, any expression that can be computed in a constant amount of time
- examples:
  - declaring a variable
  - assignment (=)
  - arithmetic (+, -, *, /, %)
  - comparison (<, >, ==, !=)
  - Boolean expressions (||, &&, !)
  - if, else
  - return statement

Estimating complexity

- count the number of elementary operations in each line of **`minToFront`**
  - assume that the following are all elementary operations:
    - **`t.size()`**
    - **`t.get(0)`**
    - **`t.get(1)`**
    - **`t.set(0, ...)`**
    - **`t.set(1, ...)`**
    - **`t.subList(x, y)`**
  - leave the line with the recursive call blank for now

Recursively Move Smallest to Front

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Recursion.minToFront(t.subList(1, t.size()));
    int first = t.get(0);
    int second = t.get(1);
    if (second < first) {
      t.set(0, second);
      t.set(1, first);
    }
  }
}
```

number of elementary ops

3
1
3
3
2
1
1

Estimating complexity

- for each line of code, determine how often it is executed

Recursively Move Smallest to Front

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {                                   1
      return;                                             1 or 0
    }
    Recursion.minToFront(t.subList(1, t.size()));         1 or 0
    int first = t.get(0);                                 1 or 0
    int second = t.get(1);                                1 or 0
    if (second < first) {                                 1 or 0
      t.set(0, second);                                   1 or 0
      t.set(1, first);                                    1 or 0
    }
  }
}
```

Total number of operations

- before we can determine the total number of elementary operations, we need to count the number of elementary operations arising from the recursive call
- let $T(n)$ be the total number of elementary operations required by **minToFront(t)**

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {



    Recursion.minToFront(t.subList(1, t.size()));

                                    1 elementary operation



  }
}
```

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {



    Recursion.minToFront(t.subList(1, t.size()));

                                  1 elementary operation



  }
}
```

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {

    Recursion.minToFront(t.subList(1, t.size()));

  }
}
```

$T(n-1)$ elementary operations

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {



    Recursion.minToFront(t.subList(1, t.size()));

  }
}
```

$T(n-1)$ elementary operations

1 elementary operation

1 elementary operation

$$= T(n-1) + 2$$

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Recursion.minToFront(t.subList(1, t.size()));
    int first = t.get(0);
    int second = t.get(1);
    if (second < first) {
      t.set(0, second);
      t.set(1, first);
    }
  }
}
```

these lines run if the base case is true

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {                                   3 * 1
      return;                                             1 * 1
    }
    Recursion.minToFront(t.subList(1, t.size()));         (T(n-1) + 2) * 1
    int first = t.get(0);                                 3 * 1
    int second = t.get(1);                                3 * 1
    if (second < first) {                                 2 * 1
      t.set(0, second);                                   1 * 1
      t.set(1, first);                                    1 * 1
    }
  }
}
```

Total number of operations

- base cases
  - $T(0) = T(1) = 4$

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {
      return;
    }
    Recursion.minToFront(t.subList(1, t.size()));
    int first = t.get(0);
    int second = t.get(1);
    if (second < first) {
      t.set(0, second);
      t.set(1, first);
    }
  }
}
```

this line runs if the base case is not true

these lines run if the base case is not true

these lines might run if the base case is not true

Total number of operations

- when counting the total number of operations, we often consider the worst case scenario
  - let's assume that the lines that might run always run

Total number of operations

```
public class Recursion {

  public static void minToFront(List<Integer> t) {
    if (t.size() < 2) {                                   3 * 1
      return;                                             1 * 1
    }
    Recursion.minToFront(t.subList(1, t.size()));         (T(n-1) + 2) * 1
    int first = t.get(0);                                 3 * 1
    int second = t.get(1);                                3 * 1
    if (second < first) {                                 2 * 1
      t.set(0, second);                                   1 * 1
      t.set(1, first);                                    1 * 1
    }
  }
}
```

Total number of operations

- base cases
  - $T(0) = T(1) = 4$
- recursive case
  - $T(n) = T(n-1) + 15$

- the two equations above are called the *recurrence relation* for **minToFront**

Selection Sort

```
public class Recursion {

  // minToFront not shown

  public static void selectionSort(List<Integer> t) {
    if (t.size() > 1) {
      Recursion.minToFront(t);
      Recursion.selectionSort(t.subList(1, t.size()));
    }
  }


}
```

number of elementary ops?

Total number of operations

- base cases
  - $T(0) = T(1) = 4$
- recursive case
  - $T(n) = T(n-1) + 15$

- the two equations above are called the *recurrence relation* for **minToFront**
- let's try to solve the recurrence relation

Solving the recurrence relation

$T(0) = 4$
$T(1) = 4$
$T(n) = T(n-1) + 15$

- if we knew $T(n-1)$ we could solve for $T(n)$

$$T(n) = T(n-1) + 15$$
$$= (T(n-2) + 15) + 15$$
$$= T(n-2) + 2(15)$$

$$T(n-1) = T(n-2) + 15$$

Solving the recurrence relation

$T(0) = 4$
$T(1) = 4$
$T(n) = T(n-1) + 15$

- if we knew $T(n-2)$ we could solve for $T(n)$

$$
\begin{aligned}
T(n) &= T(n-1) + 15 \\
&= (T(n-2) + 15) + 15 \\
&= T(n-2) + 2(15) \\
&= (T(n-3) + 15) + 2(15) \\
&= T(n-3) + 3(15)
\end{aligned}
$$

$T(n-1) = T(n-2) + 15$

$T(n-2) = T(n-3) + 15$

Solving the recurrence relation

$T(0) = 4$
$T(1) = 4$
$T(n) = T(n-1) + 15$

- if we knew $T(n-3)$ we could solve for $T(n)$

$$
\begin{aligned}
T(n) &= T(n-1) + 15 \\
&= (T(n-2) + 15) + 15 \\
&= T(n-2) + 2(15) \\
&= (T(n-3) + 15) + 2(15) \\
&= T(n-3) + 3(15) \\
&= (T(n-4) + 15) + 3(15) \\
&= T(n-4) + 4(15)
\end{aligned}
$$

$T(n-1) = T(n-2) + 15$

$T(n-2) = T(n-3) + 15$

$T(n-3) = T(n-4) + 15$

Solving the recurrence relation

$T(0) = 4$
$T(1) = 4$
$T(n) = T(n-1) + 15$

- there is clearly a pattern

$T(n) = T(n-k) + k(15)$

Solving the recurrence relation

$$T(0) = 4$$
$$T(1) = 4$$
$$T(n) = T(n-1) + 15$$

- substitute $k = n - 1$ so that we reach a base case

$$\begin{aligned} T(n) &= T(n-k) + k(15) \\ &= T\big(n-(n-1)\big) + (n-1)(15) \\ &= T(1) + 15n - 15 \\ &= 4 + 15n - 15 \\ &= 15n - 11 \in O(n) \end{aligned}$$

Big-O notation

- Proof: $f(n) = 15n - 11, g(n) = n$

For $n \geq 1$, $f(n) > 0$ and $g(n) \geq 0$; therefore, we do not need to consider the absolute values. We need to find M and m such that the following is true:

$$15n - 11 < Mn \text{ for all } n > m$$

For $n > 0$ we have:

$$\frac{15n - 11}{n} < \frac{15n}{n} = 15$$

$\therefore 15n - 11 < 15n$ for all $n > 0$ and $T(n) \in O(n)$

Try to solve the recurrence relation

$T(1) = 1$
$T(n) = T(n-1) + 3n$

Try to solve the recurrence relation

$$T(1) = 7$$
$$T(n) = T\left(\frac{n}{2}\right) + 1$$

Try to solve the recurrence relation

$T(0) = 3$
$T(1) = 3$
$T(n) = T(n-1) + Mn + 5$