Examples of running-time analysis of recursive methods

Sherif Khattab

June 18, 2018

The following are a couple examples on analyzing the running-time of recursive methods and algorithms. For each example, two techniques are demonstrated: using proof by induction and using the recursion tree.

Example 1: Singly-recursive method for displaying the entries of an array

```java
public static void displayArray(int[] a, int end) {
    if(end > a.length-1) {
        throw new IllegalArgumentException();
    }
    if(end >= 0) { //non-empty range
        displayArray(a, end-1);
        System.out.println(a[end]);
    }
}
```

Method 1: Proof by induction

Assume $T(n)$ is the time it takes to display an array of size n using the `displayArray` method. $T(n)$ can be described by the recurrence:

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

$$T(0) = 1$$

This is because `displayArray` has one recursive call, each with an input that is equal to one less than the size of the input array (this is represented by $T(n-1)$ in the recurrence) and a constant number of operations besides the recursive call (these are represented by the constant (1) in the recurrence). If the array is empty, we have a constant number of operations as well ($T(0) = 1$).

From the recurrence:

$$T(1) = T(0) + 1 = 1 + 1 = 2$$

$$T(2) = T(1) + 1 = 2 + 1 = 3$$

$$T(3) = T(2) + 1 = 3 + 1 = 4$$

These values suggest that:

$$T(n) = n + 1 \ \forall n \geq 1$$

We then try to prove "our guess" by induction.

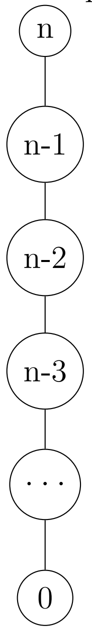
Proof. **Basis.** At $n = 1$, we try to prove that $T(1) = 1 + 1$. The Left Hand Side (LHS) = 2 (from the recurrence) = RHS.
Inductive step. Assume that $T(n) = n + 1 \ \forall 1 \leq n \leq k - 1$. We try to prove that $T(k) = k + 1$. The LHS = $T(k - 1) + 1$ (by the recurrence).

$$\begin{aligned} T(k) &= T(k-1) + 1 \\ &= (k - 1 + 1) + 1 \text{ (by the inductive hypothesis)} \\ &= k + 1 \end{aligned}$$

□

Method 2: Using the recursion tree

Each node (circle) in the recursion tree represents a recursive call. The labels indicate the size of the array that is input to the call. Because we have one recursive call, each node has one child.



The running-time is equal to:

$$\text{Running-time} = \text{number of nodes } \times \text{ time per node}$$

The time per node is the number of non-recursive operations, which is equal to a constant number of operations, that is, $O(1)$. The number of nodes is $n + 1$. So, the running time is:

$$\text{Running-time} = (n + 1) \times O(1) = O(n)$$

Example 2: Doubly-recursive method for displaying the entries of an array

```
public static void displayArray(int a[], int start, int end) {
    if((end > a.length-1) || (start < 0)){
        throw new IllegalArgumentException();
    }
    if (end >= start) { //non-empty range
        int mid = start + (end - start)/2; //to avoid integer overflow
        displayArray(a, start, mid-1);
        System.out.println(a[mid]);
        displayArray(a, mid+1, end);
    }
}
```

Method 1: Proof by induction

Assume $T(n)$ is the time it takes to display an array of size n using the `displayArray` method. $T(n)$ can be described by the recurrence:

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

$$T(0) = 1$$

This is because `displayArray` has two recursive calls, each with an input that is equal to half the array ($\frac{n}{2}$) (these are represented by $2T(\frac{n}{2})$ in the recurrence) and a constant number of operations besides the recursive calls (these are represented by the constant (1) in the recurrence above). If the array is empty, we have a constant number of operations as well ($T(0) = 1$).

From the recurrence:

$$T(1) = 2T(0) + 1 = 2 + 1 = 3$$

$$T(2) = 2T(1) + 1 = 6 + 1 = 7$$

$$T(4) = 2T(2) + 1 = 14 + 1 = 15$$

$$T(8) = 2T(4) + 1 = 30 + 1 = 31$$

$$T(16) = 2T(8) + 1 = 62 + 1 = 63$$

These values suggest that:

$$T(n) = 4n - 1 \; \forall n \geq 1$$

We then try to prove "our guess" by induction.

Proof. **Basis.** At $n = 1$, we try to prove that $T(1) = 4 * 1 - 1$. The Left Hand Side (LHS) = 3 (from the recurrence) = RHS.

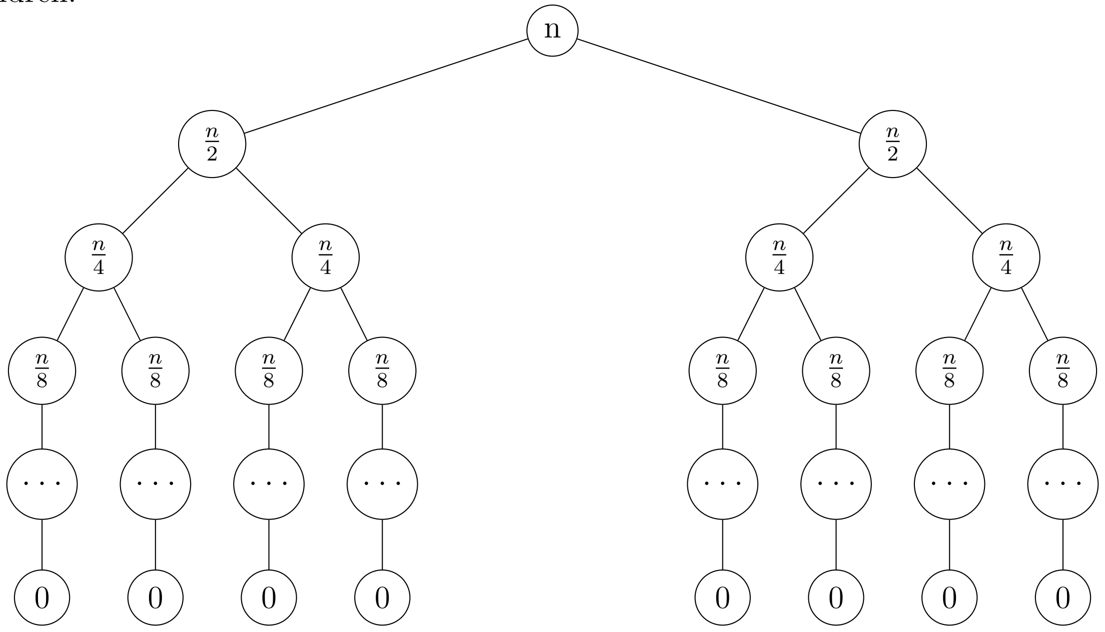
Inductive step. Assume that $T(n) = 4n - 1 \;\forall 1 \leq n \leq k-1$. We try to prove that $T(k) = 4k - 1$. The LHS $= 2T(\frac{k}{2}) + 1$ (by the recurrence).

$$\begin{aligned} T(k) &= 2T(\tfrac{k}{2}) + 1 \\ &= 2(4\tfrac{k}{2} - 1) + 1 \text{ (by the inductive hypothesis)} \\ &= 4k - 1 \end{aligned}$$

□

Method 2: Using the recursion tree

Each node (circle) in the recursion tree represents a recursive call. The label indicates the size of the array that is input to the call. Because we have two recursive calls, each node (except the leaves) has two children.



The running-time is equal to:

$$\text{Running-time} = \text{number of nodes} \;\times\; \text{time per node}$$

The time per node is the number of non-recursive operations, which is equal to a constant number of operations, that is, $O(1)$. The number of nodes is

$$\text{number of nodes} = \sum_{i=0}^{\text{number of levels}-1} \text{number of nodes at level } i$$

$$\begin{aligned} \text{number of nodes at level } 0 &= 1 = 2^0 \\ \text{number of nodes at level } 1 &= 2 = 2^1 \\ \text{number of nodes at level } 2 &= 4 = 2^2 \\ \text{number of nodes at level } i &= 2^i \end{aligned}$$

The number of levels of the above recursion tree is the number of times we can divide n by 2 before we reach 0. This is $log(n)$. So, the total number of nodes is:

$$\begin{aligned}\text{number of nodes} &= \textstyle\sum_{i=0}^{\text{number of levels}-1} \text{number of nodes at level } i\\ &= \textstyle\sum_{i=0}^{logn-1} 2^i\\ &= O(2^{logn-1})\\ &= O(2^{logn} \times 2^{-1}) = O(\frac{n}{2}) = O(n)\end{aligned}$$

So, the running time is:

$$\text{Running-time} = O(n) \times O(1) = O(n)$$