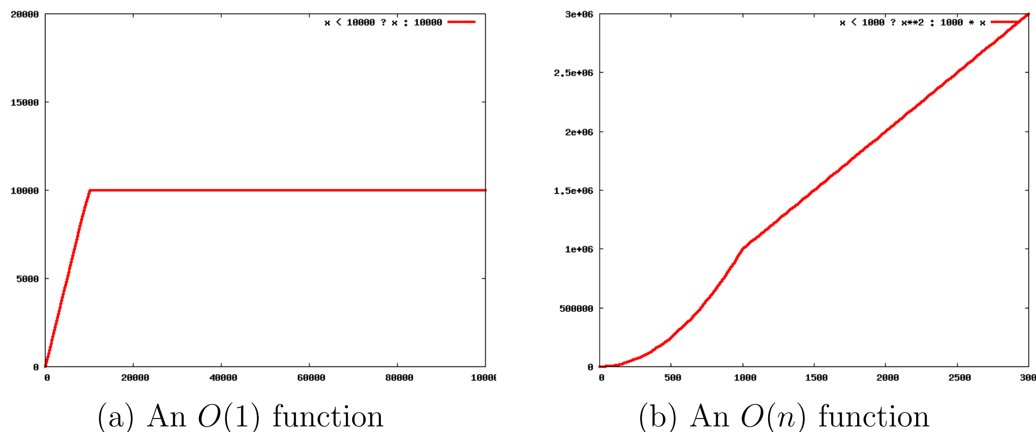
CS1114: Notes on big-O notation



(a) An $O(1)$ function (b) An $O(n)$ function

Figure 1: With big-O notation, what matters is how the amount of work grows for "large" n

In class we went over the running time of several different algorithms on different inputs. This document explains how the running time (in terms of big-O notation) of each can be arrived at. The two algorithms we will consider are the "repeated find biggest" algorithm, which repeatedly finds and removes the largest element of the array until the desired element is found, and the quickselect algorithm, described in class. For each problem we assume that the input is an array of length n. Remember that:

- When expressed in terms of big-O notation, the constant of proportionality doesn't matter. For instance, $1000n^2$ and $0.001n^2$ are both $O(n^2)$.
- We only care about the growth in the amount of work for "large" input sizes n. If a amount of work starts growing linearly for small n, but eventually "flattens out" to a constant amount of work as n grows, then the amount of work is $O(1)$ (constant). See Figure 1.

1. **What is the running time of finding the median using "repeated find biggest"?**

   Answer: $O(n^2)$. The first time we remove the biggest, we examine all n elements of the array. The next time, we look at $n - 1$, and so on, until we have removed 50% ($0.5n$) of the elements (during the last iteration, we examine the remaining $0.5n$ elements). Let $T(n)$ be the total number of elements examined. Then $T(n)$ is:

$$T(n) = n + n - 1 + n - 2 + \ldots + 0.5n$$

Each of these terms is at least $0.5n$, and there are $0.5n$ terms, so we can say that

$$T(n) \geq (0.5n)(0.5n) = 0.25n^2$$

(We can also work the sum out exactly—the result is still an expression proportional to n^2.) Thus, $T(n)$ grows in proportion to n^2, so $T(n) = O(n^2)$. Since the number of elements we examine is $O(n^2)$, the algorithm takes $O(n^2)$ time to run.

2. **What is the running time of finding the median using quickselect?**

   Answer: $O(n)$ (expected). Quickselect can find an element with any rank in expected linear time (including the median). (The worst case running time is $O(n^2)$, however.)

3. **What is the running time of finding the 2nd-largest element using "repeated find biggest"? How about the 3rd-largest element?**

   Answer: $O(n)$. The first time we remove the biggest element, we examine n elements. The second time, we examine $n-1$ elements. Thus, the total number of elements examined is $2n-1$. The amount of work done is thus proportion to n, so for finding the 2nd largest element, "repeated find biggest" runs in $O(n)$ time. Similarly, the number of elements examined when finding the 3rd-largest element is $n+n-1+n-2 = 3n-3$, which is also $O(n)$.

4. **What is the running time of finding the 50,000th-largest element using "repeated find biggest"?**

   Answer: Still $O(n)$! The number of elements examined is now:

$$n + n - 1 + n - 2 + \ldots + n - 50000$$

   which is equal to

$$50000n - (1 + 2 + 3 + \ldots 50000)$$

   i.e., 50000n minus a large constant. This is still proportional to n, though with a very large constant. Thus, this approach runs in $O(n)$ time.

5. **What is the running time of finding the 2nd-largest element using quickselect?**

   Answer: $O(n)$ expected. See question 2.

6. **What is the running time of finding the element larger than all but 5% using "repeated find biggest"?**

   Answer: $O(n^2)$. To see why, let's count the number of elements we examine. We will continue to remove the biggest element until we have removed 5% of the elements (and have 95% remaining). Thus, the total number of elements examined is:

$$n + n - 1 + n - 2 + \ldots + 0.95n$$

There are $0.05n$ elements in this summation, and all of them are at least as big as $0.95n$. Thus, the total number of elements examined is at least:

$$(0.95n)(0.05n) = 0.0475n^2$$

which is proportional to n^2. Thus, the running time is $O(n^2)$.

7. **What is the running time of finding the biggest element *just in the first 50,000 elements of the array*?**

   Answer: $O(1)$ (constant time)! The reason is that, for arrays larger than size $n = 50,000$, we only need to examine the first 50,000 elements. In other words, we only examine 50,000 elements total no matter how large the array is. Thus, we can do this in constant time.