Chapter 6

Running time of programs

For any program P and any input x, let $t_P(x)$ denote the number of "steps" P takes on input x. We need to specify what we mean by a "step." A "step" typically corresponds to machine instructions being executed, or some indication of time or resources expended.

Consider the following (somewhat arbitrary) accounting for common program steps:

method call: 1 step + steps to evaluate each argument, + steps to execute the method.

return statement: 1 step + steps to evaluate return value.

if statement: 1 step + steps to evaluate condition.

assignment statement: 1 step + steps to evaluate each side.

arithmetic, comparison, boolean operators: 1 step + steps to evaluate each operand.

array access: 1 step + steps to evaluate index.

member access: 2 steps.

constant, variable evaluation: 1 step.

Linear search

Let's use linear search as an example.

// A is an array, x is an element to search for.
// Return an index i such that $A[i] = x$;
// if there is no such index, return -1.
// Convention: array indices start at 0

```
   LS (A, x) {
1.   i = 0;                     // 3 steps (evaluate variable, constant, assignment)
2.   while (i < A.length) {     // 5 steps (while, A.length (2 steps), i, <)
3.      if (A[i] == x) {        // 5 steps (A[i], ==, x, if)
4.         return i;            // 2 steps (return, i)
5.      }
6.      i = i + 1;              // 5 steps (i, assignment, i, +, 1)
7.   }
8.   return -1;                 // 2 steps (return, −1)
9. }
```

Let's trace a function call, `LS([2,4,6,8],4)`:

Line 1: 3 steps ($i = 0$)

Line 2: 5 steps ($0 < 4$)

Line 3: 5 steps ($A[0] == 4$)

Line 6: 5 steps ($i = 1$)

Line 2: 5 steps ($1 < 4$)

Line 3: 5 steps ($A[1] == 4$)

Line 4: 2 (return 1)

So $t_{LS}([2,4,6,8],4) = 30$. Notice that if the first index where x is found is j, then $t_{LS}(A,x)$ will count lines 2, 3, and 6 once for each index from 0 to $j-1$ (j indices), and then count lines 2, 3, 4 for index j, and so $t_{LS}(A,x)$ will be $3 + 15j + 12 = 15(j+1)$.

If x does not appear in A, then $t_{LS}(A,x) = 3 + 15A.\mathit{length} + 7 = 15A.\mathit{length} + 10$, because line 1 executes once, lines 2,3, and 6 execute for each index from 0 to $A.\mathit{length} - 1$ ($A.\mathit{length}$ indices), and then lines 2 and 8 execute.

We want a measure that depends on the size of the input, not the particular input. There are three standard ways. Let P be a program, and let I be the set of all inputs for P. Then:

BEST-CASE COMPLEXITY: $\min(t_P(x))$, where x is an input of size n.
In other words, $B_P(n) = \min\{t_P(x) \mid x \in I \wedge \mathit{size}(x) = n\}$.

WORST-CASE COMPLEXITY: $\max(t_P(x))$, where x is an input of size n.
In other words, $W_P(n) = \max\{t_P(x) \mid x \in I \wedge \mathit{size}(x) = n\}$.

AVERAGE-CASE COMPLEXITY: the weighted average over all possible inputs of size n.
Assuming all the inputs are equally likely,

$$A_P(n) = \frac{\sum_{x \text{ of size } n} t_P(x)}{\text{number of inputs of size } n}$$

Best-case: mostly useless. Average-case: difficult to compute. Worst-case: easier to compute, and gives a performance guarantee.

What is meant by "input size"? This depends on the algorithm. For linear search, the number of elements in the array is a reasonable parameter. Technically (in CSC363, for example), the size is the number of bits required to represent the input in binary. In practice we use the number of elements of input (length of array, number of nodes in a tree, etc.) Remember that if we're using asymptotic notation to bound run times, multiplicative constants usually don't matter too much (they'll get absorbed into the c).

What is the best-case for linear search?[1] What about the worst-case for linear search?[2] The average-case for linear search?[3] Once we've answered these questions, we can use the machinery of Big-O. Suppose U is an upper bound on the worst-case running time of some program P, denoted $T_P(n)$:

$t_P \in O(U)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T_P(n) \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \max\{t_P(x) \mid x \in I \wedge \mathit{size}(x) = n\} \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \forall x \in I, \mathit{size}(x) = n \Rightarrow t_P(x) \leq cU(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall x \in I, \mathit{size}(x) \geq B \Rightarrow t_P(x) \leq cU(\mathit{size}(x))$

So to show that $T_P \in O(U(n))$, you need to find constants c and B and show that for an arbitrary input x of size n, P takes at most $c \cdot U(n)$ steps.

In the other direction, suppose L is a lower bound on the worst-case running time of algorithm P:

$T_P \in \Omega(L)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \max\{t_P(x) \mid x \in I \wedge size(x) = n\} \geq cL(n)$
$\Leftrightarrow$
$\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow \exists x \in I, size(x) = n \wedge t_P(x) \geq cL(n)$

So, to prove that $T_p \in \Omega(L)$, we have to find constants c, B and for arbitrary n, find an input x of size n, for which we can show that P takes at least $cL(n)$ steps on input x

Insertion sort example

Here is an intuitive[4] sorting algorithm:

```
// A is an array of comparable elements
// that will be rearranged (sorted) in non-decreasing order
   IS (A) {
1.   i = 1;
2.   while (i < A.length) {
3.      t = A[i];
4.      j = i;
5.      while (j > 0 && A[j-1] > t) {
6.         A[j] = A[j-1];
7.         j = j-1;
8.      }
9.      A[j] = t;
10.  i = i+1;
11.}
```

Since we last computed running time, we got lazier. We could use the list from last time for the number of steps of each expression, and we'd find that there are between 3 and 11 steps for the lines in the program above. Since we are interested in big-O comparisons, that four-fold difference in steps will be absorbed into our multiplicative constants, so a better use of our time would be to count each line as one step.

Let's find an upper bound for $T_{IS}(n)$, the maximum number of steps to Insertion Sort an array of size n. We'll use the proof format to prove and find the bound simultaneously — during the course of the proof we can fill in the necessary values for c and B.

We show that $T_{IS}(n) \in O(n^2)$ (where $n = A.length$):

> Let $c =$ ____. Let $B =$ ____.
>> Then $c \in \mathbb{R}^+$ and $B \in \mathbb{N}$.
>> Let $n \in \mathbb{N}$, and let A be an array of length n, and assume $n \geq B$.
>>> So lines 5–7 execute at most n times, for n steps, plus 1 step for the last loop test.
>>> So lines 2–11 take no more than $n^2 + 5n + 1$ steps.
>>> So $n^2 + 5n + 1 \leq cn^2$ (fill in the values of c and B that makes this so — setting $c = B = 6$ should do).
>>
>> So $n \geq B \Rightarrow T_{IS}(n) \leq cn^2$.
>> Since n is the length of an arbitrary array A and a natural number, $\forall n \in \mathbb{N}, n \geq B \Rightarrow T_{IS}(n) \leq cn^2$ (so long as $B \geq 1$).
>
> Since c is a positive real number and B is a natural number, $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T_{IS}(n) \leq cn^2$.
> So $T_{IS} \in O(n^2)$ (by definition of $O(n^2)$).

Similarly, we prove a lower bound. Specifically, $T_{IS} \in \Omega(n^2)$:

> Let $c =$ ____. Let $B =$ ____.
>> Then $c \in \mathbb{R}^+$ and $B \in \mathbb{N}$.
>> Let $n \in \mathbb{N}$, and let $A = [n-1, \ldots, 1, 0]$ (notice that this means $n \geq 1$). Assume $n \geq B$.
>>> Note that at any point during the outside loop, $A[0..(i-1)]$ contains the same elements as before but sorted (i.e., no element from $A[(i+1)..(n-1)]$ has been examined yet). Since the value $A[i]$ is less than all the values $A[0..(i-1)]$, by construction of the array, the inner while loop makes i iterations, at a cost of 3 steps per iteration, plus 1 for the final loop check. This is strictly greater than $2i+1$, so (since the outer loop varies from $i = 1..i = n-1$ and we have $n-1$ iterations of lines 3 and 4, plus one iteration of line 1), we have that $t_{IS}(n) \geq 1 + 3 + 5 + \cdots + (2n-1) + (2n+1) = n^2$ (the sum of the first n odd numbers).
>>
>> So $n \geq B \Rightarrow T_{IS}(n) \geq cn^2$ (setting $B = c = 1$ will do).
>> So there is some array A of size n such that $t_{IS}(A) \geq cn^2$.
>> Since n was an arbitrary natural number, $\forall n \in \mathbb{N}, n \geq B \Rightarrow T_{IS}(n) \geq cn^2$.
>
> Since $c \in \mathbb{R}^+$ and B is a natural number, $\exists c \in \mathbb{R}^+, \exists B \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq B \Rightarrow T_{IS}(n) \geq cn^2$.
> So $T_{IS} \in \Omega(n^2)$ (definition of $\Omega(n^2)$).

From these proofs, we conclude that $T_{IS} \in \Theta(n^2)$.

Exercises

1. We determined that the running time for linear search $LS(A, x)$ looking for x in an array A of length n, denoted $t_{LS}(A, x)$, was:

   - $t_{LS}(A, x) = 15$ if $A[0] = x$ (best case).
   - $t_{LS}(A, x) = 15n + 10$, if x is not in the array (worst case).

   Let $T_{LS}(n) = \max\{t_{LS}(A, x) \mid A \text{ has length } n\}$.

   (a) Is $T_{LS}(n) \in O(n)$?[5]
   (b) Is $T_{LS}(n) \in O(\frac{1}{n})$?[6]
   (c) Is $T_{LS}(n) \in O(n^2)$?
   (d) Is $T_{LS}(n) \in O(2^n)$?
   (e) Is $T_{LS}(n) \in \Omega(\frac{1}{n+1})$?

Chapter 6 Notes

[1]15 steps, when $A[0] == x$.

[2]$15n + 10$, where $n = A.length$.

[3]Inputs at index 0 through $n - 1$, plus missing value equally likely $(n + 1)$ input categories, so

$$\begin{aligned}\frac{\left(\sum_{i=0}^{n-1} 15(i+1)\right) + 15n + 10}{n+1} &= \frac{15n(n+1)/2 + 15n + 10}{n+1} \\ &= \frac{7.5n(n+1) + 15n + 10}{n+1} \\ &= \frac{7.5n(n+1) + 15n + 10}{n+1} = 7.5n + \frac{15n+10}{n+1}.\end{aligned}$$

[4]but not particularly efficient...

[5]Yes. If $n \geq 10$ and $c = 16$, then $T_{LS}(n) = 15n + 10 \leq 16n$.

[6]No. There is no constant that will make $15n + 10 \leq c/n$, once n is somewhat greater than $16c$.