Part 3: Runtime Analysis

Contents

[document finalized]

Runtime Analysis of Iterative Algorithms

Example 1. Adding all numbers stored in an array

```
// assume n > 0
// input: array of n numbers
// output: sum of all numbers
function sum(A[0..n-1])
sum <- 0
for i <- 0 to n-1 do
  sum <- sum + A[i]  (*)
end
return sum
```

Runtime measured in number of steps in pseudo-code representation

Strategy: identify line that is executed most often

Likely candidates are located in bodies of innermost loops

Here: count how often line (*) is being executed: $C(n)$

Clearly, $C(n) = n \in \Theta(n)$

Finding an asymptotic runtime upper bound (O, o, Θ) also proves termination

Example 2: Check whether all elements in an array are unique

Idea: generate all pairs (i, j) with $i < j$ and return false iff (= if and only if) such a pair exist with `A[i] = A[j]`

```
// assume n > 0
function unique(A[0..n-1])
for i <- 0 to n-2 do
  for j <- i+1 to n-1 do
    if A[i] = A[j] then   (*)
      return false
    end
  end
end
return true
```

The runtime of this program can be measured as the number of comparisons $C(n)$ in line (*). All other instructions are executed at most that often. Then:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1$$

We want to derive a closed-form representation of $C(n)$

and start with the innermost sum:

$$\sum_{j=i+1}^{n-1} 1 = \underbrace{(n-1) - (i+1) + 1}_{\text{high-low+1 elements}} = n - 1 - i$$

So,

$$C(n) = \sum_{i=0}^{n-2} (n - 1 - i) \; =$$

$$\underbrace{(n-1)^2}_{(\text{high-low+1})(n-1)} - \underbrace{\frac{(n-2)(n-1)}{2}}_{\sum_{i=0}^{n-2} i}$$

$$\in \Theta(n^2)$$

because the quadratic terms don't cancel each other out.

$\Theta(n^2)$ is slow for large n.

Better algorithm: sort sequence (can be done in time $O(n \log n)$, as we will see later) and then check neighbors for equality ($O(n)$). Total runtime is $O(n \log n)$ which is much smaller than $\Theta(n^2)$ for large n.

Looking back at the derivation of $C(n)$ we see that having two nested loops that count up to n led to quadratic runtime. This is no coincidence and can be generalized: for k such nested loops the innermost statements will be executed $\Theta(n^k)$ times.

Runtime Analysis of Recursive Algorithms

Example 1: Compute $n! = 1 \cdot 2 \cdots n$ (read: "n factorial")

Recursive definition:

$0! := 1$

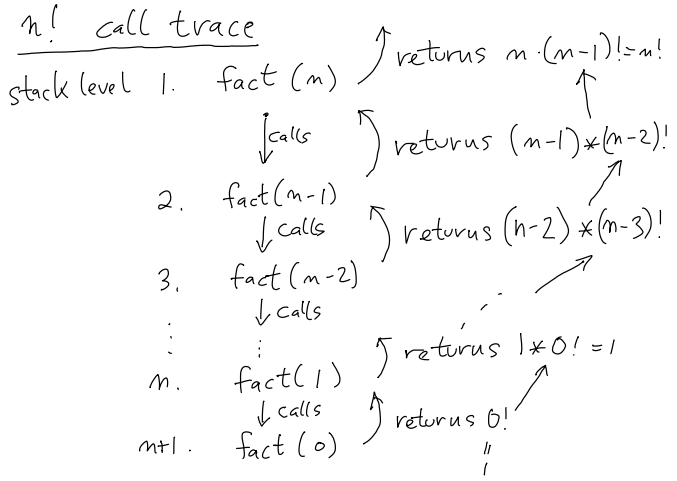
$n! := n \cdot (n-1)!$ for $n > 0$

Pseudo-code derived directly from the definition:

```
// assume n >= 0
function fact(n)
if n = 0 then
  return 1
end
return n * fact(n-1)  (*)
```


Each function call stores the return address on the system stack, incremenents the stack pointer, jumps to the function, and when it returns from it, decreases the stack pointer, and resumes execution right after the call.

Here is what happens when we call `fact(n)`:



Runtime Analysis:

$M(n) =$ number of multiplications executed in line (*)

$$M(0) = 0$$
$$M(n) = M(n-1) + 1, \quad n > 0$$

Want closed-form solution for this recurrence relation.

Apply it repeatedly to see an emerging pattern ("iterated substitution")

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

After i steps:

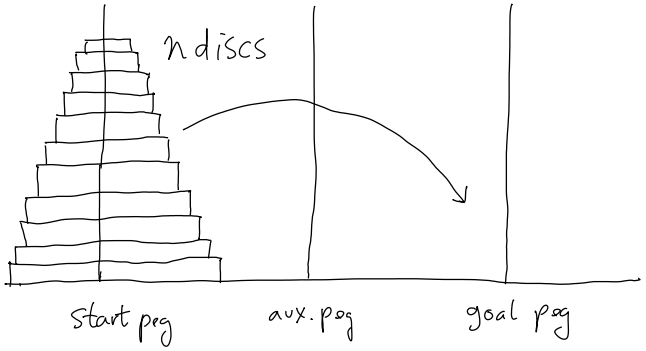
$$M(n) = M(n-i) + i$$

Recursion stops when $i = n \rightsquigarrow M(n) = M(0)+n = n$

Runtime of `fact` on a RAM is therefore $\Theta(n)$.

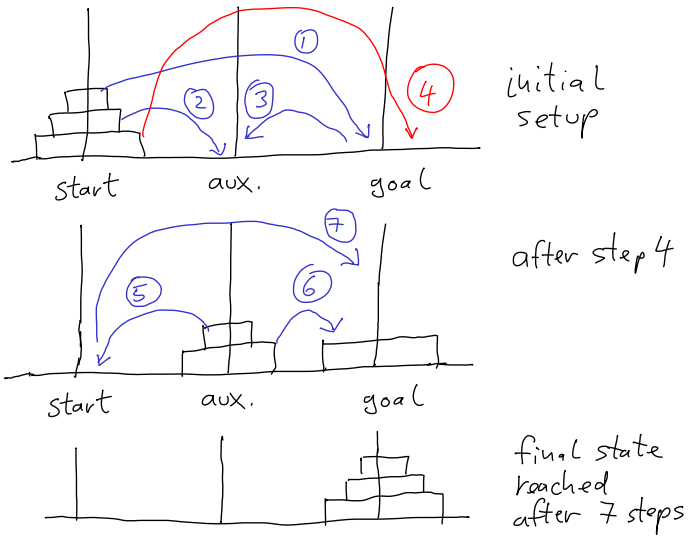
Note: $n!$ grows very quickly ($15! = 1307674368000$). So, assuming that each multiplication takes constant time vastly underestimates the runtime when executing function `fact` on a contemporary computer, which usually stores 64-bit values in each memory cell. In this case using the logarithmic-cost measure would be more appropriate.

Example 2: Towers of Hanoi

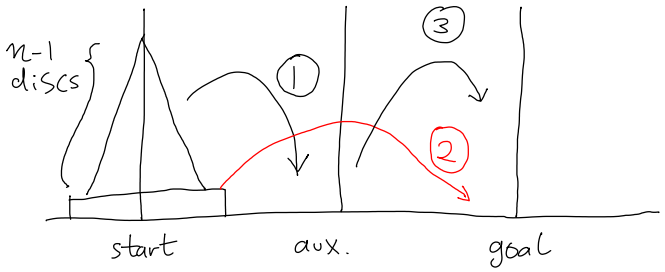


Three pegs, n discs of decreasing size located on one peg, smallest on top. Other pegs empty. Objective: move all discs to another peg subject to the constraint that no bigger disk can be placed on top of a smaller one.

Solving the 3-disc case:



Observation: can be solved recursively by first moving $n-1$ discs to another peg, then moving the biggest disk to the remaining peg, and then moving the $n-1$ smaller discs on top the biggest disc.



In pseudo-code:

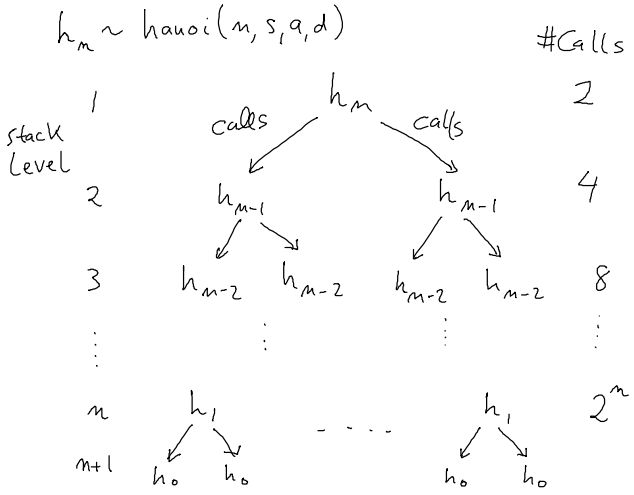
```
// move n disks from src to dst using aux
// assume n >= 0
function hanoi(n, src, aux, dst)
if n > 0 then
  hanoi(n-1, src, dst, aux)
  print "move " src " to " dst   (*)
  hanoi(n-1, aux, src, dst)
end

// Call to move n discs from peg 0 to 2:  hanoi(n, 0, 1, 2)
```

Lecture 9

Output of `hanoi(3,0,1,2)`:

```
move 0 to 2
move 0 to 1
move 2 to 1
move 0 to 2
move 1 to 0
move 1 to 2
move 0 to 2
```

Call trace:



Runtime measured in the number of times line (*) is executed

$$M(1) = 1$$
$$M(n) = M(n-1) + 1 + M(n-1),\ n > 1$$

Use iterated substitution to obtain hypothesis

$$\begin{aligned}
& M(n) \\
&= 2M(n-1) + 1 \\
&= 2(2M(n-2)+1) + 1 \\
&= 4M(n-2) + 2 + 1 \\
&= 8M(n-3) + 4 + 2 + 1 \\
&= 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 \\
&= 2^i M(n-i) + 2^i - 1 && \text{(geometric series)} \\
&= 2^{n-1}\, M(1) + 2^{n-1} - 1 && \text{(base case for } i = n-1) \\
&= 2^n - 1 && (M(1) = 1)
\end{aligned}$$

Prove hypothesis by mathematical induction:

Induction base $n = 1$: $M(1) = 1$ OK

Induction step: assume $M(n) = 2^n - 1$ (Induction Hypothesis (I.H.))

Then, using the recurrence relation and the I.H., we get

$M(n+1) = 2M(n)+1 =_{\text{(I.H.)}} 2(2^n-1)+1 = 2^{n+1}-1$

Thus, $M(n) = 2^n - 1$ for all $n \geq 1$.

Exponential runtime in n. Can we do better?

No. The $n-1$ stack needs to be moved to another peg before the biggest disk can be moved. The biggest disk needs to be moved at least once, and then the $n-1$ stack must be moved again.

Result: our algorithm is optimal!

Iterated Substitution

1. Play with recurrence, apply several times
2. Identify emerging pattern
3. Determine when base case is reached
4. Guess closed form or bound
5. Prove closed form or bound by mathematical induction

Example:

$$T(1) > 0$$
$$T(n) \ = \ 2T(\lfloor n/2 \rfloor) + n$$

For simplicity, we assume $n = 2^k$, so that we don't have to deal with rounding down ($\lfloor x \rfloor = \max\{y \in \mathbb{Z} \mid y \leq x\}$ — pronounced "floor of x". The floor's twin is called ceiling, denoted $\lceil x \rceil$, which rounds up.

Then $T(n) \ = \ 2T(n/2)+n \ = \ 2(2T(n/4)+n/2)+n$

$$= \ 4T(n/4) + n + n \ = \ 2^i T(n/2^i) \ + \ n \cdot i$$

[reaches base case when $n = 2^i$, i.e. $i = \log(n) = k$]

$$= 2^k T(1) + n\log(n) \ = \ n \cdot T(1) + n \log n$$

Guess: $T(n) \in O(n \log n)$, which we try to prove by showing

$$T(n) \leq An \log n + Bn$$

for all $n \geq 1$ and suitable constants $A > 0$ and B that we plan to determine along the way.

Induction Base $n = 1$: Plugging in $n = 1$ in above inequality yields

$$T(1) \leq A \cdot 1 \cdot \log(1) + B \cdot 1 = B$$

For this to hold $B \geq T(1)$ is required.

Induction step: Now $n > 1$ and we assume

$$T(k) \leq Ak \log k + Bk$$

for all $k < n$. This is the induction hypothesis (I.H.).

By using the recurrence relation and the I.H. we obtain:

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2\Big[A\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) + B\lfloor n/2 \rfloor\Big] + n \\ &\leq 2\Big[A(n/2)\log(n/2) + B(n/2)\Big] + n \end{aligned}$$

because $\lfloor x \rfloor \leq x$, and $\log$ is a monotonically increasing function, i.e. $x < y \Rightarrow \log(x) < \log(y)$. Therefore:

Lecture 10

$$\begin{aligned} T(n) &\leq An \log(n/2) + Bn + n \\ &= An(\log(n) - 1) + Bn + n \\ &= An \log(n) - An + Bn + n \end{aligned}$$

So, $T(n) \leq An \log(n) + Bn$ $(*)$ holds if

$$\begin{aligned} -An + Bn + n &\leq Bn \\ \Leftrightarrow \quad -A + B + 1 &\leq B \\ \Leftrightarrow \quad -A + 1 &\leq 0 \\ \Leftrightarrow \quad 1 &\leq A \end{aligned}$$

Thus, with $A = 1$ and $B = T(1)$ $(*)$ holds for all n.

So we know: $T(n) \in O(n \log n)$

$T(n) \in \Omega(n \log n)$ analogous $\rightsquigarrow$ $T(n) \in \Theta(n \log n)$ □

Frequent problem with iterated substitution:
Induction hypothesis not strong enough for the induction step

Example

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

We guess: $T(n) \in O(n)$, e.g. $T(n) \leq C \cdot n$

Induction step:

$$T(n) \leq C\lfloor n/2 \rfloor + C\lceil n/2 \rceil + 1 \;=\; Cn + 1$$

(Note: for all $n \in \mathbb{N}$ $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$)

Oops, that didn't work. We needed to show $T(n) \leq Cn$. But we can strengthen the induction hypothesis:

$$T(n) \leq Cn - B \quad (B > 0)$$

Then

$$\begin{aligned} T(n) &\leq C\lfloor n/2 \rfloor - B + C\lceil n/2 \rceil - B + 1 \\ &= C \cdot n - 2B + 1 \;\leq\; C \cdot n - B, \end{aligned}$$

if $-2B + 1 \leq -B$, which is equivalent to $B \geq 1$.

Sometimes changing variables works:

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n$$

Intimidating. Our plan is to transform this recurrence into something we know how to solve.

Try $n = 2^m$ $\Rightarrow$

$$T(2^m) = 2T(2^{m/2}) + m$$

and let

$$S(m) := T(2^m)$$

$\rightsquigarrow$ (using the recurrence for T)

$$S(m) = 2S(m/2) + m$$

$\rightsquigarrow$ (seen earlier)

$$S(m) \in O(m \log m)$$

$\rightsquigarrow$

$$\begin{aligned} T(n) = T(2^m) = S(m) &\in O(m \log m) \\ &= O(\log n \log \log n) \end{aligned}$$

using $m = \log n$, which follows from the assumption $n = 2^m$.

Master Theorem Finding a pattern using iterated substitution which is then followed by an induction proof is cumbersome. We want a tool to solve a class of frequently occurring recurrence relations like:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 3n + 1$$

Note that often both floor and ceiling operations occur.

The following theorem is based on the so-called "Master Theorem" (CLRS p.94, L p.483) which we will cite after the simplified version without proof (which is rather technical).

Master Theorem for Linear f

Suppose $a \geq 1, b \geq 1$ and $T(n) = aT(\{n/b\}) + f(n)$, with

- $T(0) > 0$
- $f(n)$ eventually > 0 (i.e. $\exists N\ \forall n \geq N\ f(n) > 0$), and
- $\{n/b\}$ denoting $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$ or combinations thereof.

If $f(n) = An + B$ with $A > 0$ then,

1. If $a > b$, then $T(n) \in \Theta(n^{\log_b a})$

   "total size of parts exceeds n"

2. If $a = b$, then $T(n) \in \Theta(n \log n)$

   "total size of parts equals n"

3. If $a < b$, then $T(n) = \Theta(n)$.

   "total size of parts smaller than n"

Examples in which the theorem applies (f is linear):

Case 1. $T(n) = 4T(\lfloor n/3 \rfloor) + 5T(\lceil n/3 \rceil) + 4n + 1$

The combined coefficient for all terms of form $\{n/3\}$ is 9 $(a = 9)$,
and $b = 3$.

Therefore, $a > b$ and $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

Case 2. $T(n) = T(\lfloor n/3 \rfloor) + 2T(\lceil n/3 \rceil) + 5n$
$\rightsquigarrow a = 3,\ b = 3$

Therefore, $a = b$ and $T(n) = \Theta(n \log n)$

Case 3. $T(n) = 2T(\lceil n/3 \rceil) + 4n + 1$
$\rightsquigarrow a = 2,\ b = 3$

Therefore, $a < b$ and $T(n) = \Theta(n)$

Master Theorem (CLRS p.94, L p.483)

Suppose $a \geq 1, b \geq 1$ and $T(n) = aT(\{n/b\}) + f(n)$, with

- $T(0) > 0$
- $f(n)$ eventually > 0 (i.e. $\exists N\ \forall n \geq N\ f(n) > 0$), and
- $\{n/b\}$ denoting $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$ or combinations thereof.

For $\lambda = \log_b(a)$

1. If $f(n) \in O(n^\gamma)$ for a $\gamma < \lambda$, then $T(n) \in \Theta(n^\lambda)$

2. If $f(n) \in \Theta(n^\lambda)$, then $T(n) \in \Theta(n^\lambda \log n)$

3. If $f(n) \in \Omega(n^\gamma)$ for a $\gamma > \lambda$, and the smoothness condition

   $$af(n/b) \leq cf(n)$$

   holds for a $c < 1$ and all sufficiently large n, then $T(n) \in \Theta(f(n))$

Examples

Case 1: $T(n) = 9T(\lfloor n/3 \rfloor) + \sqrt{n}$

$$a = 9,\ b = 3,\ \lambda = \log_3 9 = 2,\ f(n) = \sqrt{n}$$

Try to find a $\gamma < \lambda = 2$ such that $f(n) \in O(n^\gamma)$

$$\rightsquigarrow\ f(n) \in O(n^1),\ 1 < 2.\ \text{OK}$$

$$\Rightarrow T(n) \in \Theta(n^\lambda) = \Theta(n^2)$$

Lecture 11

Case 2: $T(n) = T(\lceil 2n/3 \rceil) + 1$

$$a = 1,\ b = 3/2,\ \lambda = \log_{3/2} 1 = 0,\ f(n) = 1$$

$$\rightsquigarrow\ f(n) \in \Theta(n^\lambda) = \Theta(1)$$

$$\Rightarrow T(n) \in \Theta(n^\lambda \log n) = \Theta(\log n)$$

Case 3:

$$T(n) = T(\lfloor n/4 \rfloor) + 2T(\lceil n/4 \rceil) + n \log n$$

$$a = 3\ (3\ \text{“}n/4\text{”}\ \text{cases}),\ b = 4,\ \lambda = \log_4 3 \approx 0.793$$

$$f(n) = n \log n$$

Try to find a $\gamma > \lambda$ such that $f(n) \in \Omega(n^\gamma)$

$$f(n) \in \Omega(n^1),\ 1 > 0.793\ \text{OK}$$

$$\Rightarrow\ \ T(n) = \Theta(f(n)) = \Theta(n \log n)\ \ (*)$$

IF

$$af(n/b) \le cf(n)$$

for large enough n and a $c < 1$. Let's plug in $f(n)$ and check:

$$3(n/4)\log(n/4) = (3/4)n \log n - (3n/4) \cdot 2$$
$$\stackrel{?}{\le} cn \log n$$

Yes, for $c = 3/4 < 1$ and $n \ge 1$. Therefore $(*)$ holds.

Recurrence relation to which the theorem does not apply:

$$T(n) \ = \ 2T(\lfloor n/2 \rfloor) + n \log n$$

$$a = b = 2,\ \lambda = \log_b a = 1$$

Case 1? No:

$$\forall \gamma < 1 :\ n \log n \notin O(n^\gamma)$$

Case 2? No:

$$n \log n \notin \Theta(n^1)$$

Case 3? No:

$$\forall \gamma > 1 :\ n \log n \notin \Omega(n^\gamma)$$

But with the iterated substitution method we can show: $T(n) \in O(n(\log n)^2)$ — exercise.

Proof of Master Theorem for linear f from full version: also exercise.