Practice

Time Complexity

$f(n)$	$g(n)$	$O/\Omega/\Theta$
$n - 100$	$n - 200$	Θ
$n^{1/2}$	$n^{2/3}$	O
$100n + \log n$	$n + (\log n)^2$	Θ (a)
$\log 2n$	$\log 3n$	Θ (b)
$10 \log n$	$\log n^2$	Θ (c)
$n^{1/2}$	$5^{\log_2 n}$	O (d)
2^n	2^{n+1}	Θ (e)

(a): $n \; dominates \; (\log n)^c \rightarrow n + (\log n)^2 = \Theta(n)$

(b): $\log ab = \log a + \log b$

(c): $\log a^b = b \log a$

(d): $5 = 2^{2x} \; where \; x > 0 \rightarrow 5^{\log_2 n} = (2^{2x})^{\log_2 n} = (2^{\log_2 n})^{2x} = \Omega(n^{1/2})$

(e): $2^{n+1} = 2 \times 2^n$

Fibonacci 1

$$F_n = \begin{cases} 0, if \; n = 0 \\ 1, if \; n = 1 \\ F_{n-1} + F_{n-2}, otherwise \end{cases}$$

Prove: $F_n = \Omega(\sqrt{2^n})$.

By trial and error: It appears that $F(n) \geq 2^{n/2}$ for all $n \geq 7$

To prove: For all positive integers $n \geq 7 \rightarrow F_n \geq 2^{n/2}$

By induction on n. Base case: $n = 7$

Step, assume: Indeed, true that for all $i = 7, 8, \dots, k \rightarrow F_i \geq 2^{i/2}$

To prove: $F_{k+1} \geq 2^{k+1/2}$

Paolo Torres

Practice

LHS: $F_{k+1} = F_k + F_{k-1} \geq 2^{k/2} + 2^{(k-1)/2}$

Suffices to prove: $2^{k/2} + 2^{(k-1)/2} \geq 2^{(k+1)/2}$

$2^{1/2} + 1 \geq 2^{2/2}$ (by dividing the above by $2^{(k-1)/2}$)

It is indeed true that $2^{1/2} + 1 \geq 2^{2/2} = 1$

Multiplication

Figure 1.1 Multiplication à la Français.

```
function multiply(x, y)
Input: Two n-bit integers x and y, where y ≥ 0
Output: Their product

if y = 0: return 0
z = multiply(x, ⌊y/2⌋)
if y is even:
    return 2z
else:
    return x + 2z
```

Suppose instead of both x and y being n-bit, x is n-bit and y is m-bit. What is the worst-case time efficiency of $multiply$?

Proposed: $O(nm)$

Time Efficiency:

- $\#\ recursive\ calls\ x\ time/call$
- $\#\ worst\ case\ recursive\ calls = O(m)$
- $Worst\ case\ time/call =$
    - $2z\ is\ at\ worst\ O(n+m) \rightarrow because\ very\ last\ addition\ is\ 2z = xy - x$
    - $x\ is\ n\ bits$
    - $So, addition's\ time: O(max\ \{n, n+m\}) = O(max\ \{n, m\})$

So, final answer: $O(m \times max\ \{n, m\})$

Fibonacci 2

Paolo Torres

Practice

Let F_n be the nth Fibonacci number, Prove $F_n = O(2^n)$.

- Somewhere, we have shown: $F_n = \Omega(\sqrt{2}^n)$

- But here, seek to show: There exists positive real $F_n \leq c \cdot 2^n$, for all n in N

- Natural proof strategy for "there exists" – construction (i.e., propose some concrete c, and show that it works)

- Try some small values for n, and see what c would work

    - $n = 0, F_0 = 0, 2^0 = 1 \rightarrow c = 1 \; works$

    - $n = 1, F_1 = 1, 2^1 = 2 \rightarrow c = 1 \; works$

    - $n = 2, F_2 = 1, 2^2 = 4 \rightarrow c = 1 \; works$

    - $n = 3, F_3 = 2, 2^3 = 8 \rightarrow c = 1 \; works$

    - $n = 4, F_4 = 3, 2^4 = 16 \rightarrow c = 1 \; works$

- Appears that $c = 1$ works. Adopt it and check if proof goes through. Now, proof by induction with $c = 1$

- Base case, $n = 1, F_1 = 1, 2^1, 1 \leq 2 \rightarrow True$

- Step: Seek to show $F_n \leq 2^n$ given that $F_k \leq 2^k$ for all $k = 1, 2, \ldots, n - 1$

- $F_n = F_{n-1} + F_{n-2} \leq 2^{n-1} + 2^{n-2}$ by induction assumption

- $F_n = 2^{n-2}(2 + 1) = 3 \times 2^{n-2} \leq 2^n = 2^2 \times 2^{n-2} = 4 \times 2^{n-2} \rightarrow Done$

Fibonacci 3

Let F_n be the nth Fibonacci number, Prove $F_n \neq O(n^2)$.

- Recall from logic: not (there exists an egg-laying mammal) = for all mammals m, m is not egg-laying

- Here, $f = O(g)$: There exists positive real c, for all natural n, $f(n) \leq c \cdot g(n)$

- So here, need to prove: Given any positive real c, it is true that there exists n such that $F_n > c \cdot n^2$

- By contradiction: Suppose that there exists positive real c, such that, for all natural n, $F_n \leq c \cdot n^2$

- Then: $F_n = F_{n-1} + F_{n-2} \leq c(n - 1)^2 + c(n - 2)^2 = c(n^2 - 2n + 1 + n^2 - 4n + 4) = c(2n^2 - 6n + 5) \leq cn^2$

- $2n^2 - 6n + 5 \leq n^2$

Paolo Torres

Practice

- $2 - \frac{1}{n^2}(6n - 5) \le 1$

- This is true only if $\frac{1}{n^2}(6n - 5)$ is "large" compared to $2n^2$

- What is large? We need $\frac{1}{n^2}(6n - 5) \ge 1 \rightarrow true\ for\ n = 1$

- Try $n = 2$: $\frac{1}{4}(12 - 5) = \frac{7}{4} \ge 1$

- Try $n = 3$: $\frac{1}{8}(18 - 5) = \frac{13}{8} \ge 1$

- Try $n = 4$: $\frac{1}{16}(24 - 5) = \frac{19}{16} \ge 1$

- Try $n = 5$: $\frac{1}{25}(30 - 5) = 1$

- Try $n = 6$: $\frac{1}{36}(36 - 5) < 1$

- Try $n = 7$: $\frac{1}{49}(42 - 5) < 1$

- Prove by induction: $6n - 5 < n^2$ for all natural $n > 5$

- Base case $n = 6$: See above

- Step: $6(n - 1) - 5 \le (n - 1)^2 \rightarrow from\ induction\ assumption$

- $6n - 5 - 6 < n^2 - 2n + 1$

- $6n - 5 \le n^2 - (2n - 7) \le n^2\ whenever\ 2n - 7 \ge 0 \rightarrow which\ it\ is\ for\ n \ge 6$

- So far: We have shown that indeed, for $n \ge 6$, $F_n < cn^2 \rightarrow Done$

Selection Sort

$$SELECTIONSORT(A[1, ..., n])$$
$$foreach\ i\ from\ 1\ to\ n\ do$$
$$m \leftarrow i - 1 + INDEXOFMIN(A[i, ..., n])$$
$$if\ i \ne m\ then\ swap\ A[i], A[m]$$

$$INDEXOFMIN(B[1, ..., m])$$
$$min \leftarrow B[1], idx \leftarrow 1$$
$$foreach\ j\ from\ 2\ to\ m\ do$$
$$if\ B[j] < min\ then$$
$$min \leftarrow B[j], idx \leftarrow j$$
$$return\ idx$$

What is a meaningful characterization of the time efficiency of $SELECTIONSORT$?

Paolo Torres

Practice

- Suppose we invoke $INDEXOFMIN(A[5, ..., 13])$. In $INDEXOFMIN: B[1, ..., 9]$.
  Suppose now, min is at index 3 in $B[1, ..., 9]$. This $\rightarrow$ index of a min in $A[5, ..., 13]$ is at index $(5 - 1) + 3 = 7$

- Suppose on input: $A[1, ..., 5] = [13, -23, 45, -23, 1]$. Then A evolves in $SELECTIONSORT$ as follows:
  - $i = 1, m = 2, [-23, 13, 45, -23, 1]$
  - $i = 2, m = 4, [-23, -23, 45, 13, 1]$
  - $i = 3, m = 4, [-23, -23, 13, 45, 1]$

- For time efficiency: Need to make meaningful assumption(s)

- Customary Assumptions: (1) n is unbounded, (ii) each $A[i]$ is bounded

- What should we count? Suppose we all agree that counting # swaps is a meaningful measure for time efficiency

- Then: $Worst\ case\ \#\ swaps = n - 1 = \Theta(n)$

- Now, let's say we want to get a bit more fine-grained. Incorporate (worst case) time for each swap x # swaps

- So now, time efficiency: $(n - 1) + (n - 2) + \cdots + 1 = \Theta(n^2)$

Modular Simplification

1. Is $6^6 \equiv 5^3 \ (mod\ 31)$?

$$6 \times 6 = 36 \equiv 5 \ (mod\ 31)$$

So: $(6^2)^3 \equiv (5)^3 \ (mod\ 31)$

2. $2^{125} \equiv ? \ (mod\ 127)$

$$2^7 = 128 = 127 + 1$$

So: $128 \ mod\ 127 = 1$

Now: $125/7 = 17 + 6/7$

So: $2^{125} = 2^{17 \times 7 + 6} = 2^{17 \times 7} \times 2^6$

So: $2^{125} \equiv 2^{17 \times 7} \times 2^6 \equiv (2^7)^{17} \times 2^6 \equiv 1^{17} \times 2^6 \equiv 64 \ (mod\ 127)$

Paolo Torres

Practice

3. Is $4^{1536} - 9^{4824}$ divisible by 35?

$$4^{1536} \equiv 9^{4824} \ (mod \ 35)$$

Trick: Keep exponentiating until numbers start to repeat.

Suppose we repeatedly exponentiate 4:

$$4$$

$$\rightarrow 16$$

$$\rightarrow 64 \equiv 29 \ (mod \ 35)$$

$$\rightarrow 116 = 35 \times 3 + 11 = 11 \ (mod \ 35)$$

$$\rightarrow 9 \ (mod \ 35)$$

$$\rightarrow 36 \equiv 1 \ (mod \ 35)$$

So: $4^6 \equiv 1 \ (mod \ 35)$. And $1536 = 6 \times 256$. So $4^{1536} \equiv 1 \ (mod \ 35)$

Now check whether 1536 is divisible by 4. Indeed: $1536 = 4 \times 384$

Repeat with 9. Repeated exponentiation of 9:

$$9$$

$$\rightarrow 81 \equiv 11 \ (mod \ 35)$$

$$\rightarrow 99 \equiv 29 \ (mod \ 35)$$

$$\rightarrow 261 = 7 \times 35 + 16 \equiv 16 \ (mod \ 35)$$

$$\rightarrow 144 \equiv 4 \times 35 + 4 \equiv 4 \ (mod \ 35)$$

$$\rightarrow 36 \equiv 1 \ (mod \ 35)$$

So: $9^6 \equiv 1 \ (mod \ 35)$

Now: $9^{4824} = 9^{804 \times 6} \equiv 1 \ (mod \ 35)$.

$\therefore$ *It is divisible by* 35.

4. $2^{2^{2006}} \ (mod \ 3) = ?$

Paolo Torres

Practice

$$2^{2^{2006}} = (2^2)^{2^{2005}} = 4^{2^{2005}} \equiv 1 \ (mod \ 3)$$

5. Is $5^{30000} - 6^{123456}$ a multiple of 31?

   31 is prime. And $5^{30000} = (5^{30})^{1000} \equiv 1 \ (mod \ 31)$.

   Compare with $6^{123456} = 6^{123450} \times 6^6$:

$$1 \times 6^6 \equiv 5^3 \equiv 125 \equiv 31 \times 4 + 1 \ (mod \ 31) \equiv 1 \ (mod \ 31)$$

   $\therefore It \ is \ a \ multiple \ of \ 31.$

Show that if a has a multiplicative inverse modulo N, then this inverse is unique (modulo N).

Let's assume $a \in \{1, \dots, N - 1\}$.

Suppose $b, c \in \{1, \dots, N - 1\}$ are both multiplicative inverses of $a \ modulo \ N$. Then:

$$ab \equiv 1 \ (mod \ N)$$
$$ac \equiv 1 \ (mod \ N)$$

$$ab \equiv ac \ (mod \ N)$$
$$ab \cdot b \equiv ac \cdot b \ (mod \ N) \ (1)$$

(1): Substitution Rule:

$$x \equiv x', y \equiv y' (mod \ N)$$
$$xy \equiv x'y' (mod \ N)$$

Then:

$$(ab) \cdot b \equiv (ab) \cdot c \ (mod \ N) \ (2)$$

(2): Commutativity

$$1 \cdot b \equiv 1 \cdot c \ (mod \ N)$$
$$b \equiv c \ (mod \ N)$$
$$b = c$$

Paolo Torres

Practice

Suppose $p \equiv 3 \ (mod \ 4)$. Show that $(p+1)/4$ is an integer.

$$p \equiv 3 \ (mod \ 4)$$
$$p = 4k + 3 \ for \ some \ k \in \mathbb{Z}$$

So: $p + 1 = 4k + 4$, which is divisible by 4.

We say that x is a square root of y modulo a prime p if $y \equiv x^2 \ (mod \ p)$. Show that if $(i) \ p \equiv 3 \ (mod \ 4)$ and $(ii) \ y$ has a square root modulo p, then $y^{(p+1)/4}$ is such a square root.

Let x be the square root of $y \ modulo \ p$. Then: $y \equiv x^2 \ (mod \ p)$.

Write $p = 4k + 3$. Then, $\left(y^{\frac{p+1}{4}}\right)^2 = y^{2(p+1)/4} = y^{2(4k+3+1)/4} = y^{2k+2}$

Keep in mind: $(p+1)/4 = k + 1$.

Try plugging in x in the last expression:

Is $y^{2k+2} = x^{4k+4} \equiv x^2$?

So, we're asking: Is $x^{4k+4} - x^2 \equiv 0 \ (mod \ p)$?

$$x^{4k+4} - x^2 = (x^{2k+2} - x)(x^{2k+2} + x)$$

So at least one of: $x^{2k+2} - x$ or $x^{2k+2} + x$ must be $\equiv 0 \ (mod \ p)$.

- $\frac{(p+1)}{4} = \frac{(4k+3+1)}{4} = k + 1$
- $2 \cdot \frac{(p+1)}{4} = 2k + 2$
- $p - 1 = 4k + 2$

We know: There exists $x \in \{1, ..., p-1\}$ such that $y \equiv x^2 \ (mod \ p)$.

We seek to prove: $\left(y^{\frac{(p+1)}{4}}\right)^2 \equiv y \ (mod \ p)$. Sufficient condition for that to be true:

$\left(y^{\frac{(p+1)}{4}}\right)^2 \cdot y^{-1} \equiv 1 \ (mod \ p) \rightarrow$ is okay, because y is invertible modulo p

$$\Rightarrow (y^{2k+2}) \cdot y^{-1} \equiv 1 \ (mod \ p)$$
$$\Rightarrow y^{2k+1} \equiv 1 \ (mod \ p)$$

Paolo Torres

Practice

$$\Rightarrow (x^2)^{2k+1} \equiv 1 \ (mod \ p)$$

$$\Rightarrow x^{4k+2} \equiv 1 \ (mod \ p)$$

$$\Rightarrow x^{p-1} \equiv 1 \ (mod \ p)$$

$$\Rightarrow True \ (Fermat's \ little \ theorem)$$

Proving Recurrence 1

Suppose $x \in \mathbb{Z}^+, y \in \mathbb{Z}_0^+$. Prove recurrence correctness.

$$x^y = \begin{cases} 1, if \ y = 0 \\ (x^2)^{\lfloor y/2 \rfloor}, if \ y \ is \ even \\ x \cdot (x^2)^{\lfloor y/2 \rfloor}, otherwise \end{cases}$$

Case Analysis:

1. If $y = 0$, then $x^y = x^0$. So, the recurrence is correct for the case where $y = 0$

2. If $y \neq 0, y \ is \ even$: then $\lfloor y/2 \rfloor = y/2$. So $x^y = x^{2 \times y/2} = (x^2)^{y/2} = (x^2)^{\lfloor y/2 \rfloor}$

3. If $y \neq 0, y \ is \ odd$: then $\lfloor y/2 \rfloor = (y-1)/2$. So now:

$$x^y = x^{(2 \times (y-1)/2)+1} = x^{(2 \times \lfloor y/2 \rfloor)+1} = x \cdot x^{2 \times \lfloor y/2 \rfloor}$$

Proving Recurrence 2

Let $\langle q, r \rangle$ be the quotient and remainder of x/y and $\langle q', r' \rangle$ be the quotient and remainder of $(\lfloor x/2 \rfloor)/y$. Prove recurrence correctness.

$$\langle q, r \rangle = \begin{cases} \langle 0, 0 \rangle, if \ x = 0 \\ \langle 2q', 2r' \rangle, if \ x \ even \ and \ 2r' < y \\ \langle 2q', 2r' + 1 \rangle, if \ x \ odd \ and \ 2r' + 1 < y \\ \langle 2q' + 1, 2r' - y \rangle, if \ x \ even \ and \ 2r' \geq y \\ \langle 2q' + 1, 2r' + 1 - y \rangle, otherwise \end{cases}$$

To be absolutely clear, what are the quotient and remainder of x/y?

We call q the quotient, and r the remainder if and only if q and r are non-negative integers that satisfy:

$$x = q \cdot y + r, where \ r \in \{0, 1, ..., y - 1\}$$

Paolo Torres

Practice

Proof by case analysis:

1. If $x = 0$, then $x = 0 = 0 \cdot y + 0$. So, recurrence is correct for this case.

2. If x is even and $2r' < y$: then $\lfloor x/2 \rfloor = x/2$. So:

$$\lfloor x/2 \rfloor = x/2 = q' \cdot y + r'$$

$$x = (2q') \cdot y + 2r'$$

$$q = 2q', r = 2r'$$

Where we infer the last line from the facts that: (i) equation is of the form from definition for quotient and remainder, (ii) $r' \geq 0 \rightarrow 2r' \geq 0$, and (iii) we are given $2r' \leq y - 1$.

3. If x is odd and $2r' + 1 < y$: $\lfloor x/2 \rfloor = (x - 1)/2$

$$\lfloor x/2 \rfloor = (x - 1)/2 = q' \cdot y + r'$$

$$x - 1 = (2q') \cdot y + 2r'$$

$$x = (2q') \cdot y + (2r' + 1)$$

4. x is even, $2r' \geq y$: $\lfloor x/2 \rfloor = x/2$. So:

$$\lfloor x/2 \rfloor = x/2 = q' \cdot y + r'$$

$$x = (2q') \cdot y + 2r'$$

This is of the form of the definition of quotient and remainder, except that we need to confirm that $2r'$ indeed lies between 0 and $y - 1$. Which it does not necessarily. Actually, we are given that $2r' \geq y$ and therefore not between 0 and $y - 1$. Now we observe:

$$x = (2q') \cdot y + 2r'$$

$$x = (2q' + 1) \cdot y + (2r' - y)$$

Now only question that remains: is it the case that $2r' - y \in \{0, 1, ..., y - 1\}$?

- Is $2r' - y \geq 0$? Yes, because $2r' \geq y$
- Is $2r' - y \leq y - 1$? Yes, because:

$$r' \leq y - 1$$

$$2r' \leq 2y - 2$$

$$2r' - y \leq y - 2 \leq y - 1$$

Paolo Torres

Practice

5. x odd, $2r' + 1 \geq y$:

$$\lfloor x/2 \rfloor = (x - 1)/2 = q' \cdot y + r'$$

$$x = (2q') \cdot y + (2r' + 1)$$

$$x = (2q' + 1) \cdot y + (2r' + 1 - y)$$

Now:

- $2r' + 1 - y \geq 0$ because $2r' + 1 \geq y$.

- $2r' + 1 - y \leq y - 1$ because:

$$r' \leq y - 1$$

$$2r' + 1 \leq 2y - 1$$

$$2r' + 1 - y \leq y - 1$$

Proving Recurrence 3

Prove that $BinSearch$ is correct.

$BinSearch(A[1, \ldots, n], lo, hi, i)$

1. **if** $lo \leq hi$ **then**
2. $\quad mid \leftarrow \lfloor (lo + hi)/2 \rfloor$

3. $\quad$ **if** $A[mid] = i$ **then return** $true$
4. $\quad$ **if** $A[mid] < i$ **then return** $BinSearch(A, mid + 1, hi, i)$
5. $\quad$ **else return** $BinSearch(A, lo, mid - 1, i)$
6. **else return** $false$

Above is recursive version of binary search. Iterative version:

$BinSearch(A[1, \ldots, n], lo, hi, i)$

1. **while** $lo \leq hi$ **do**
2. $\quad mid \leftarrow \lfloor (lo + hi)/2 \rfloor$

3. $\quad$ **if** $A[mid] = i$ **then return** $true$
4. $\quad$ **if** $A[mid] < i$ **then** $lo \leftarrow mid + 1$

Paolo Torres

Practice

5.   **else** $hi \leftarrow mid - 1$

6. **else return** $false$

Typically, for iterative algorithms, towards correctness, we articulate a *loop invariant*:

Let $lo^{(in)}$ and $hi^{(in)}$ be the values of lo and hi respectively on input. Just before we successfully enter an iteration of the **while** loop of Line (1), it is true that:

$$i \in A\big[lo^{(in)}, \dots, hi^{(in)}\big] \to i \in A[lo, \dots, hi]$$

Going back to the recursive version, what is a correctness property?

Given $A[1, \dots, n]$ an array that is sorted, non-decreasing, lo, hi are each $\in \{1, \dots, n\}$ on input, $BinSearch(A, lo, hi, i)$ returns:

- $True \to (lo \leq hi) \; and \; (i \in A[lo, \dots, hi])$
- $False \to either \; (lo > hi) \; or \; (i \; is \; not \in A[lo, \dots, hi])$

Proof by case analysis:

Case 1: $lo > hi$ on input: then **if** condition of Line (1) evaluates to **false**, and we correctly return **false** in Line (6). Then, this is either from (a) Line (6) without making any recursive calls, or (b) as the return value from a recursive call from one of Lines (4) or (5).

For (b), we first observe that $lo \leq hi$ because the only recursive calls are within the **if** block of Line (1). So, all that remains to be proven is that indeed: $i \notin A[lo, \dots, hi]$.

We prove that by induction on $hi - lo + 1$. Base case: $hi - lo + 1 = 1$. We claim we return **$false$** within the first recursive invocation. That is, we claim: $(i) \; mid + 1 > hi$ and $lo > mid - 1$, $(ii) \; mid = lo = hi$, and $(iii) \; i \neq A[mid]$.

(ii) easy to prove:

$$hi - lo + 1 = 1$$
$$\Rightarrow lo = hi$$
$$\Rightarrow mid = \left\lfloor \frac{(lo + hi)}{2} \right\rfloor = \left\lfloor \frac{(lo + lo)}{2} \right\rfloor = \left\lfloor \frac{(2 \cdot lo)}{2} \right\rfloor = \frac{2 \cdot lo}{2} = lo = hi$$

Paolo Torres

Practice

(iii) is **true**, because then we would have returned **true** in Line (3).

To prove (i): we simply exploit: $mid = hi = lo$

$$mid = hi \Rightarrow mid + 1 > hi$$
$$mid = lo \Rightarrow mid - 1 < lo$$

So, the algorithm is correct if it returns **false**, and $hi - lo + 1 = 1$.

For the step, we know that on input $lo < hi$. So, we returned **false** in some recursive call. So, all we have to prove to appeal to induction assumption: $hi - (mid + 1) < hi - lo$ and $(mid - 1) - lo < hi - lo$.

Proving Master Theorem

Give a closed form solution for the following recurrence. Assume: $f: \mathbb{R}^+ \to \mathbb{R}^+$ is non-decreasing, $a > 0, b > 1, d \geq 0$.

$$f(n) = \begin{cases} \Theta(1), if\, n \leq 1 \\ a \cdot f\left(\dfrac{n}{b}\right) + \Theta(n^d), otherwise \end{cases}$$

Proposed approach: Inductive "rewriting" of the function f. But first: adopt concrete functions wherever we have $\Theta(\cdot)$, $O(\cdot)$ or $\Omega(\cdot)$. In this case: adopt 1 for $\Theta(1)$, and n^d for $\Theta(n^d)$. Now onto the rewriting:

$$f(n) = a \cdot f\left(\frac{n}{b}\right) + n^d$$

$$= a \cdot \left(a \cdot f\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^d\right) + n^d$$

$$= a^2 \cdot f\left(\frac{n}{b^2}\right) + a \cdot \left(\frac{n}{b}\right)^d + n^d$$

$$= a^2\left(a \cdot f\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^d\right) + a \cdot \left(\frac{n}{b}\right)^d + n^d$$

$$= a^3 \cdot f\left(\frac{n}{b^3}\right) + a^2 \cdot \left(\frac{n}{b^2}\right)^d + a \cdot \left(\frac{n}{b}\right)^d + n^d$$

$$= a^3 \cdot f\left(\frac{n}{b^3}\right) + n^d\left(\left(\frac{a}{b^d}\right)^2 + \left(\frac{a}{b^d}\right)^1 + \left(\frac{a}{b^d}\right)^0\right)$$

Paolo Torres

Practice

$$= a^4 \cdot f\left(\frac{n}{b^4}\right) + n^d \left(\left(\frac{a}{b^d}\right)^3 + \left(\frac{a}{b^d}\right)^2 + \left(\frac{a}{b^d}\right)^1 + \left(\frac{a}{b^d}\right)^0\right)$$

...

$$= a^{\log_b n} \cdot f(1) + n^d \cdot \left(\left(\frac{a}{b^d}\right)^{(\log_b n)-1} + \left(\frac{a}{b^d}\right)^{(\log_b n)-2} + \cdots + \left(\frac{a}{b^d}\right)^0\right)$$

$$= a^{\log_b n} + n^d \cdot \left(\left(\frac{a}{b^d}\right)^{(\log_b n)-1} + \left(\frac{a}{b^d}\right)^{(\log_b n)-2} + \cdots + \left(\frac{a}{b^d}\right)^0\right)$$

To figure out the power of a in that last term:

Power of a is the same as the power of b inside the $f\left(\frac{n}{b^x}\right)$. In other words, we're asking: what is the power of b, i.e., x for which $\frac{n}{b^x} = 1$? Answer: $\frac{n}{b^x} = 1 \Leftrightarrow n = b^x \Leftrightarrow x = \log_b n$.

Our next step: Simplify/figure out:

$$S = \left(\frac{a}{b^d}\right)^{(\log_b n)-1} + \left(\frac{a}{b^d}\right)^{(\log_b n)-2} + \cdots + \left(\frac{a}{b^d}\right)^0$$

Suppose:

$$T = r^{q-1} + r^{q-2} + \cdots + r^0$$

$$\Longrightarrow r \cdot T = r^q + r^{q-1} + \cdots + r$$

Now subtract one from the other:

$$\Longrightarrow T - r \cdot T = r^0 - r^q$$

$$\Longrightarrow (1 - r) \cdot T = 1 - r^q$$

$$\Longrightarrow T = \frac{1 - r^q}{1 - r}, provided\ r \neq 1$$

When $r = 1$, how do we figure out what T is? Answer: then, T is:

$$T = 1^{q-1} + 1^{q-2} + \cdots + 1^0$$

$$= 1 + 1 + \cdots + 1 \rightarrow q\ instances\ of\ 1$$

$$= q$$

So, going back to our S:

Paolo Torres

Practice

$$S = \left(\frac{a}{b^d}\right)^{(\log_b n)-1} + \left(\frac{a}{b^d}\right)^{(\log_b n)-2} + \cdots + \left(\frac{a}{b^d}\right)^0$$

$$\Rightarrow S = \frac{1 - \left(\frac{a}{b^d}\right)^{\log_b n}}{1 - \left(\frac{a}{b^d}\right)}, provided \; \frac{a}{b^d} \neq 1$$

And:

$$S = \log_b n, when \; \frac{a}{b^d} = 1$$

When is $\frac{a}{b^d} = 1$? Answer: $d = \log_b a$.

So, going back to our $f(n)$: first, the case that $d = \log_b a$.

But even before that: rewrite $a^{\log_b n} = n^{\log_b a}$. Because:

$$x = a^{\log_b n} \iff \log_b x = \log_b a \cdot \log_b n \iff x = n^{\log_b a}$$

$$f(n) = n^{\log_b a} + n^d \cdot S$$

So, when $d = \log_b a$, $S = \log_b n$. So, in this case:

$$f(n) = n^d + n^d \cdot \log_b n$$
$$= \Theta(n^d \cdot \log n)$$

Onto the other two cases: $d \neq \log_b a$.

$$f(n) = n^{\log_b a} + \cdots + n^d \cdot S$$

Before we continue: a closer look at $\left(\frac{a}{b^d}\right)^{\log_b n}$:

$$\left(\frac{a}{b^d}\right)^{\log_b n} = \frac{a^{\log_b n}}{(b^d)^{\log_b n}}$$

$$= \frac{n^{\log_b a}}{(b^{\log_b n})^d}$$

$$= \frac{n^{\log_b a}}{n^d}$$

So: when $d \neq \log_b a$

Paolo Torres

Practice

$$S = \frac{1 - \dfrac{n^{\log_b a}}{n^d}}{1 - \left(\dfrac{a}{b^d}\right)}$$

So, going back to $f(n)$:

$$f(n) = n^{\log_b a} + n^d \cdot S$$

$$= n^{\log_b a} + \frac{1}{1 - \left(\dfrac{a}{b^d}\right)} \cdot \left(n^d - n^{\log_b a}\right)$$

$$= c \cdot n^{\log_b a} + c' \cdot n^d, \text{ for positive constants } c, c'$$

So, if $d > \log_b a$: $f(n) = \Theta(n^d)$

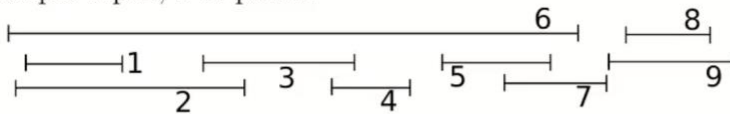And if $d < \log_b a$: $f(n) = \Theta\left(n^{\log_b a}\right)$

Proving Greediness

Given as input n meeting requests, $\langle s_1, f_1 \rangle, \langle s_2, f_2 \rangle, \ldots, \langle s_n, f_n \rangle$, where each $s_i, f_i \in \mathbb{Z}^+$ is a start- and finish-time and $s_i < f_i$. We want a subset of those requests that is of maximum size that are pairwise conflict-free.

Two requests $\langle s_i, f_i \rangle, \langle s_j, f_j \rangle$ are in conflict if $s_i \leq f_j$, and $s_j \leq f_i$, or vice versa.

Example input, 9 requests:

Request 5 is in conflict with each Request 6 and 7. But is conflict-free with Request 2.

An optimal (maximum-sized) conflict-free set: $\{1, 3, 5, 9\}$. Another: $\{1, 4, 7, 8\}$.

Prove: this problem possesses a greedy choice.

Candidate greedy choice: request with earliest finish time.

Proof strategy: "cut and paste."

For this problem, we prove two claims in order:

Claim 1: *Suppose for some input of n requests, $O = \{o_1, \ldots, o_k\}$ is an optimal (maximum-sized) set of requests which are pairwise conflict-free ordered in increasing finish time. Suppose our*

Paolo Torres

Practice

greedy algorithm outputs $G = \{g_1, \dots, g_l\}$, *ordered in increasing finish time. Then, it is true that: for every* $i = 1, 2, \dots, l, f(g_i) \leq f(o_i)$.

Proof. Note: it must be the case that $l \leq k$. And therefore, $k = l$, i.e., greedy is optimal.

Proof by induction on i. Base case: $i = 1$. In our greedy algorithm, we first pick exactly a meeting that finishes earliest amongst all requests. Therefore, immaterial of what o_1 is, $f(g_1) \leq f(o_1)$.

Induction assumption: for $i = j - 1$, it is true that $f(g_i) \leq f(o_i)$.

Step: to prove that $f(g_j) \leq f(o_j)$. We observe:

- $f(o_{j-1}) \leq s(o_j)$ – because the set O is conflict-free requests, ordered in increasing finish, and therefore, start times.
- $f(g_{j-1}) \leq f(o_{j-1})$ – induction assumption.
- Therefore, $f(g_{j-1}) \leq s(o_j)$. Therefore $f(g_j) \leq f(o_j)$ – because after we greedily choose g_{j-1} and eliminate all requests that are in conflict, o_j still remains. And our greedy choice is exactly to pick a request that remains that finishes earliest, and we happened to pick g_j.

Claim 2: *Given sets* O, G *as in Claim 1,* o_{l+1} *cannot exist in* O.

Proof. By Claim 1, $f(g_l) \leq f(o_l)$. And because the O set is all conflict-free, $f(o_l) \leq s(o_{l+1})$. Therefore, $f(g_l) \leq s(o_{l+1})$. So, o_{l+1} not in conflict with g_l, and so was available to be chosen after g_l was chosen and all conflicts were eliminated.

Contradiction to the assumption that greedy algorithm terminates only when no more requests available to choose from.

Graph Algorithm 1

Given an undirected graph $G = \langle V, E \rangle$ encoded as an adjacency list, define an array snd[·] as: for each $u \in V$, snd[u] is the sum of the degrees of the neighbours of u.

Devise an algorithm that given input G, computes and outputs an array snd.

Paolo Torres

Practice

$SNDStraightForward(G = \langle V, E \rangle)$

1. $snd \leftarrow new\ array\ of\ |V|\ entries$

2. $\boldsymbol{foreach}\ u \in V\ \boldsymbol{do}\ snd[u] \leftarrow 0$

3. $\boldsymbol{foreach}\ u \in V\ \boldsymbol{do}$

4. $\boldsymbol{foreach}\ v \in Adj[u]\ \boldsymbol{do}$

5. $degreev \leftarrow 0$

6. $\boldsymbol{foreach}\ w \in Adj[v]\ \boldsymbol{do}\ degreev \leftarrow degreev + 1$

7. $snd[u] \leftarrow snd[u] + degreev$

8. $\boldsymbol{return}\ snd$

Time efficiency of $SNDStraightForward$: $O(|V| \cdot (|E|)^2)$

Perhaps a better (more efficiency) approach:

- Visit each vertex as though it is someone's neighbor.

- Measure its degree.

- Walk its adj list again and inform each neighbor of the degree so they can update their snd.

$SNDLinearTime(G = \langle V, E \rangle)$

1. $snd \leftarrow new\ array\ of\ |V|\ entries$

2. $\boldsymbol{foreach}\ u \in V\ \boldsymbol{do}\ snd[u] \leftarrow 0$

3. $deg \leftarrow new\ array\ of\ |V|\ entries$

4. $\boldsymbol{foreach}\ u \in V\ \boldsymbol{do}\ deg[u] \leftarrow 0$

5. $\boldsymbol{foreach}\ u \in V\ \boldsymbol{do}$

5. $deg[u] \leftarrow 0$

6. $\boldsymbol{foreach}\ v \in Adj[u]\ \boldsymbol{do}\ deg[u] \leftarrow deg[u] + 1$

7. $\boldsymbol{foreach}\ v \in Adj[u]\ \boldsymbol{do}\ snd[v] \leftarrow snd[v] + deg[u]$

8. $\boldsymbol{return}\ snd$

Time efficiency:

- We visit each vertex once – Line (4) $foreach$ loop.

- We visit each edge four times – Line (6) and Line (7), we walk each adj list twice.

- So total time: $O(|V| + |E|)$.

Paolo Torres

Practice

Graph Algorithm 2

Given an undirected graph G as an adjacency list and an edge e in it, devise a linear-time algorithm to determine whether there is a cycle in G that contains e.

"Go-to" linear time algorithms for graphs: DFS and BFS.

Idea:

- DFS, check if back edge results in DFS tree.
- In fact, edit the explore routine as follows:
  - Keep track of parent in DFS tree.
  - Every time we hit a vertex, check if edge to root of DFS tree, and root is not parent in DFS tree.
  - If yes, immediately output **true**.

$$HasCycle(G = \langle V, E \rangle, e = \langle u, v \rangle)$$

1. **foreach** $u \in V$ **do**
2. $visited(u) \leftarrow false$
3. $\pi(u) \leftarrow NIL$
4. **return** $ExploreModified(\langle V, E \rangle, u, u)$

$ExploreModified(\langle V, E \rangle, \langle u, v \rangle, x)$

1. $visited(x) \leftarrow true$
2. **foreach** $y \in Adj[x]$ **do**
3.   **if** $visited(y) = false$ **then**
4.   **if** $(x \neq u)$ or $(x = u$ and $y = v)$ **then**
5. $\pi(y) \leftarrow x$
6. $ret \leftarrow ExploreModified(\langle V, E \rangle, \langle u, v \rangle, y)$
7.   **if** $ret = true$ **then**
8.   **return** $true$
9.   **else**
10.   **if** $y = r$ and $\pi(x) \neq u$ **then**
11.   **return** $true$
12. **return** $false$

Paolo Torres