COMP 3804 : Design and Analysis of Algorithms I

algorithms

COMP 3803 ~~2805~~ definition: Turing machine

this course: "well-defined computational procedure that transforms an input into an output"

Example: input: road map, two locations A and B
output: shortest path from A to B.

Our task: discover the algorithm

Focus in this course:

* correctness of algorithms
* does it terminate?
* efficient (= fast):
  - estimate $\underbrace{\text{running time}}_{\text{what does this mean?}}$
  - count the number of $\underbrace{\text{steps}}_{\text{what is a "step"?}}$
* limits of efficiency: some problems cannot be solved efficiently
* pseudocode, no programming

Example to illustrate this:

Fibonacci numbers

$F_0 = 0$, $F_1 = 1$, for $n \geq 2$: $F_n = F_{n-1} + F_{n-2}$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

This definition leads to an algorithm that returns F_n for any given input $n \geq 0$:

```
Algorithm fib(n):
    if n ≤ 1 : return n
    else: return fib(n-1) + fib(n-2)
```

* correct? yes
* terminate? yes

* efficient?
  how does the running time depend on the input n?

<u>Define</u> $T(n)$ = # of steps when running fib(n).

for $n=0$ or $n=1$:

$$\left.\begin{array}{l}\text{comparison "}n \le 1\text{"} \\ \text{return value}\end{array}\right\} \text{2 steps}$$

for $n \ge 2$:

comparison "$n \le 1$"	: 1 step
compute $n-1$	: 1 step
call fib$(n-1)$	: $T(n-1)$ steps
compute $n-2$	: 1 step
call fib$(n-2)$	: $T(n-2)$ steps
compute sum of 2 results	: 1 step

$T(0) = 2$

$T(1) = 2$

for $n \geq 2$: $T(n) = T(n-1) + T(n-2) + 4$

Exercise: Prove by induction that

$$T(n) \geq F_n.$$

$\therefore$ running time of algorithm fib(n) is

$\geq F_n$.

But: F_n is very large:

Exercise: Prove by induction that

* $F_n \geq 2^{n/2}$ for $n \geq 6$
* $F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$

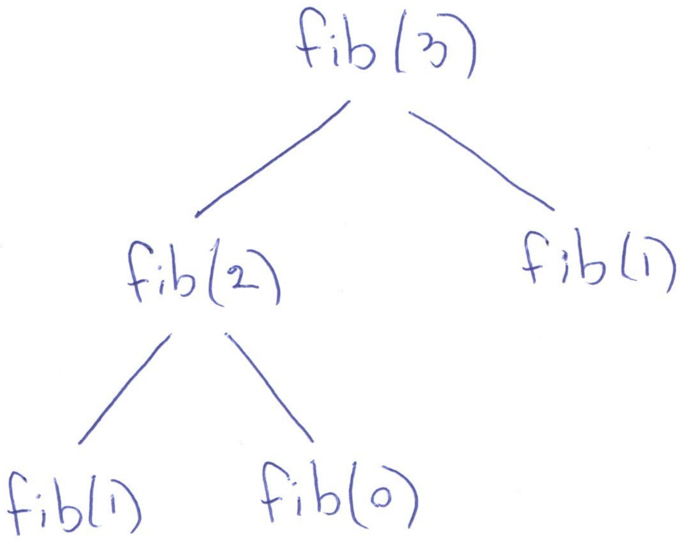
  for $n \geq 0$

∴ fib(n) takes at least exponential time

fib(200) will not terminate during our lifetime.

Why is fib(n) so slow?

Recursion tree for fib(3):



∴ fib(1) is called twice

Better algorithm :

Algorithm fib′(n):

if $n \leq 1$: return n

else: initialize array f[0..n];

f[0] = 0;

f[1] = 1;

for $i = 2$ to n: $f[i] = f[i-1] + f[i-2]$;

return f[n]

correct ? yes

terminate ? yes

running time : <u>linear</u> in n.

<u>Conclusion</u> :

fib(n) : exponential (very slow)

fib′(n) : linear (very fast)

But: is it realistic to say that fib'(n) takes a linear number of steps?

In our analysis: one step is:

comparison
addition
subtraction

of very large numbers

In kindergarden, you learned:
two n-bit numbers can be added in a linear number of bit-operations

When running fib'(n):

$\approx$ n additions of numbers

each of these numbers is $\leq F_n$

each of these numbers has $\approx$ n bits

$\therefore$ fib'(n) makes a quadratic number of bit-operations.

Exercise: Convince yourself that fib(n) makes a number of bit-operations that is proportional to $n \cdot F_n$.

Asymptotic notation:

$f(n) = O(g(n))$:

$$\exists c \; \exists n_0 \; \forall n \geq n_0 : \; f(n) \leq c \cdot g(n)$$

$f(n) = \Omega(g(n))$:

$$\exists c \; \exists n_0 \; \forall n \geq n_0 : \; f(n) \geq c \cdot g(n)$$

$f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$:

$$\exists c \; \exists c' \; \exists n_0 \; \forall n \geq n_0 :$$

$$c \cdot g(n) \leq f(n) \leq c' \cdot g(n)$$

$O / \Omega / \Theta$:

* ignore constant factors
* focus on the dependency on n
* consider large values of n

Running time, in bit-operations, of

* fib (n) : $O(n \cdot F_n)$: exponential
* fib'(n) : $O(n^2)$: quadratic

I assume you know:

* induction
* O, Ω, Θ
* sorting algorithms:

  $\left.\begin{array}{l}\text{insertion sort}\\ \text{selection sort}\\ \text{bubble sort}\end{array}\right\} O(n^2)$

  merge sort : $O(n \log n)$

  quicksort : $\begin{cases} O(n^2) & \text{worst case,} \\ O(n \log n) & \text{"always"} \end{cases}$

* comparison-based sorting: $\Omega(n \log n)$ for <u>every</u> algorithm
* balanced binary search trees (AVL, red-black, ...)

  search, insert, delete : $O(\log n)$