Introduction to Data Structures and Algorithms

Chapter: Calculating Fibonacci Numbers

Friedrich-Alexander-Universität
Erlangen-Nürnberg

Lehrstuhl Informatik 7 (Prof. Dr.-Ing. Reinhard German)
Martensstraße 3, 91058 Erlangen

Iteration vs. Recursion

- **Example** $\sum_{i=m}^{n} i^2 = m^2 + (m+1)^2 + \cdots + n^2$ where $\mathrm{m} \le \mathrm{n}, \quad \mathrm{n}, \ \mathrm{m} \in \mathrm{N}$

- <u>Iterative sum of squares</u>

```
ItSum_Squares(m,n)
  Sum:=0
   for i = m to n do
   Sum:= Sum + i * i
```

- <u>Recursion sum of squares</u>

```
RecSum_Squares(m,n)
  if m < n
   then
     Sum:= m * m + RecSum_Squares(m+1,n)
   Sum:= m * m
```

Iteration vs. Recursion

- **Example** $\sum_{i=m}^{n} i^2 = m^2 + (m+1)^2 + \cdots + n^2 \text{ where } \mathrm{m} \leq \mathrm{n}, \quad \mathrm{n}, \mathrm{m} \in \mathrm{N}$

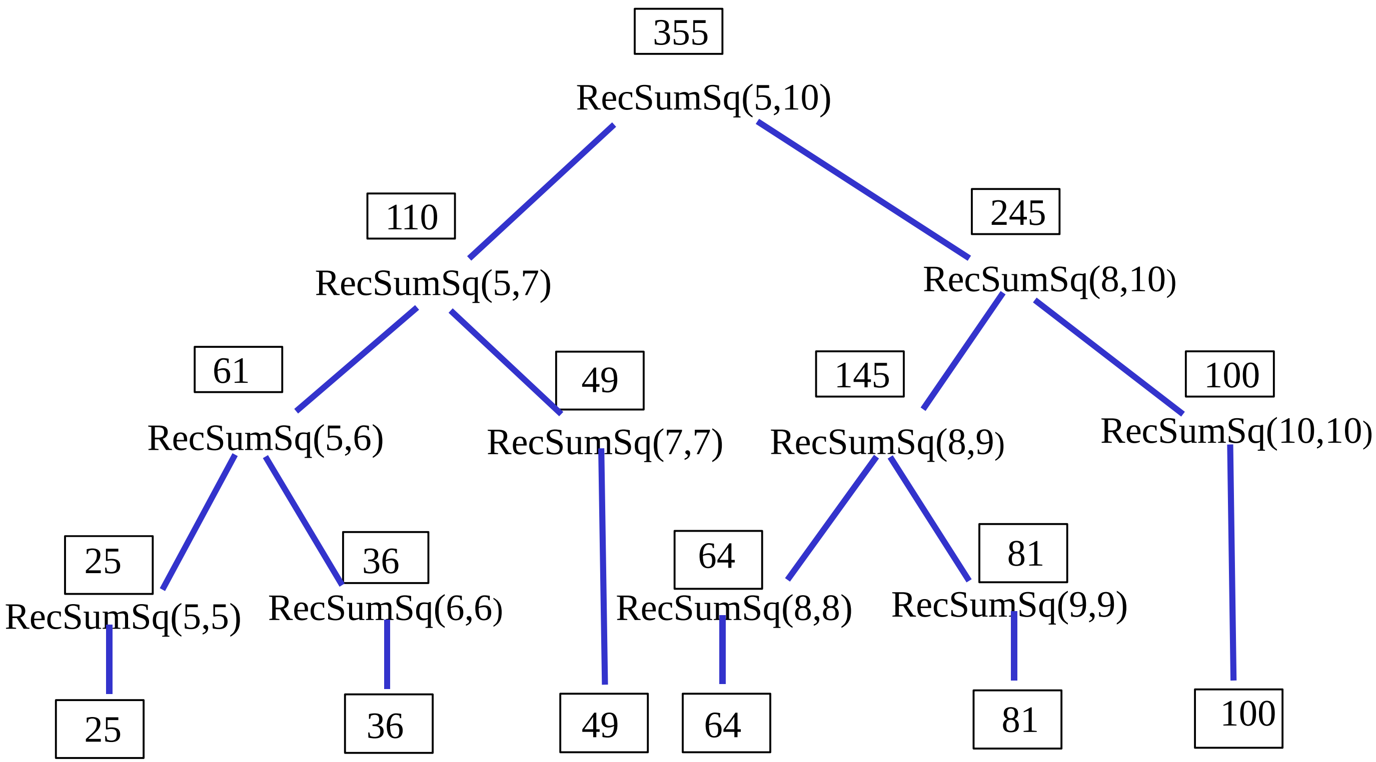
- <u>**Recursion combining two half-solutions**</u>

(method Divide-and-conquer)

```
RecSumSq(m,n)
  if m = n
   then
     RecSumSq:= m * m
  else
     mid:= floor((m + n)/2)
     RecSumSq:= RecSumSq(m,mid)+ RecSumSq(mid+1,n)
```

Iteration vs. Recursion

- <u>Call Tree for Recursion RecSumSc (5,10)</u>



Calculating Fibonacci Numbers

- **Fibonacci numbers** f_i are defined by the following recurrence

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 1 \\
f_i &= f_{i-1} + f_{i-2} \quad \text{for} \quad i \geq 2
\end{aligned}
$$

- This leads to the sequence

i	0	1	2	3	4	5	6	7	8	9	...
f_i	0	1	1	2	3	5	8	13	21	34	...

Calculating Fibonacci Numbers

- Problem:

  How can we compute the i-th Fibonacci number f_i ?

- We will present 3 sample solutions
  - Recursive algorithm (*fibrec*)
  - Iterative algorithm (*fibiter*)
  - Iterative squaring algorithm (*fibisq*)

- We will have to investigate
  - how good these algorithms perform and
  - which of these solutions is the “best solution”!

Calculating Fibonacci Numbers

1) A recursive algorithm *fibrec*

- What does “recursive algorithm” mean?
  - Again the recursive definition:

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_i &= f_{i-1} + f_{i-2} \quad \text{for} \quad i \geq 2 \end{aligned}$$

- For i=0, 1, ... compute

```
fibrec(i)
   if i=0 then return 0
   if i=1 then return 1
   return fibrec(i-1)+fibrec(i-2)
```

Calculating Fibonacci Numbers

Analysis of the recursive algorithm

- We could find out
  the exact runtime for each operation and use these values

- Instead we typically procede as follows:
  We count the number of arithmetic operations performed
  when executing *fibrec* and consider this value as runtime
  (or cost or complexity) of *fibrec*:
  - Arithmetic operations: additions, subtractions, multiplications, divisions

- For simplicity here we assume
  that all operations need the same amount of time for execution

Calculating Fibonacci Numbers

Analysis of the recursive algorithm

- Let $C_{rec}(i)$ be the cost of computing f_i using *fibrec*(i)
  - i.e. $C_{rec}(i)$ =
    number of arithmetic operations when computing f_i using *fibrec*
- Then

$$
\begin{aligned}
C_{rec}(0) &= 0 \\
C_{rec}(1) &= 0 \\
C_{rec}(i) &= 3 + C_{rec}(i-1) + C_{rec}(i-2) \qquad \textbf{for} \qquad i \geq 2
\end{aligned}
$$

- The values of $C_{rec}(i)$ for small values (i=0, 1, 2, ...) are

i	0	1	2	3	4	5	...
$C_{rec}(i)$	0	0	3	6	12	21	...

Calculating Fibonacci Numbers

Analysis of the recursive algorithm

- A closed-form expression for the cost $C_{rec}(i)$ would be fine!
- "Someone guesses" that (for i=0, 1, ...)

$$\boxed{C_{rec}(i) = 3f_{i+1} - 3}$$

- Try to proof this assumption!

  (Which method of proof would be your favorite choice?)

Calculating Fibonacci Numbers

Analysis of the recursive algorithm

- Use this result
  to find information about f_i
  - Find lower and upper bounds
  - Hint: Show that for $i > 0$
    - f_i is positive
    - f_i is monotonically increasing

- Result: For $i > 2$:

$$\boxed{2^{(i-2)/2} \leq f_i \leq 2^{i-2}}$$

⇨ f_i growing exponentially

⇨ **$C_{rec}(i)$ growing exponentially**

$$\text{Expl}: \quad 10^{13} \leq C_{rec}(100) \leq 10^{30}$$

i	Lower bound (f_i)	Upper bound (f_i)
2	1	1
3	1.4	2
4	2.0	4
10	16	256
20	512	262144

Calculating Fibonacci Numbers

2) An iterative algorithm *fibiter*

- For i=0, 1, ... compute

```
fibiter(i)

   if i=0 then return 0
   if i=1 then return 1

   ppred := 0       ▻ pre-predecessor
   pred  := 1       ▻ predecessor

   for j := 2 to i do
       curr := pred+ppred      ▻ current
       ppred := pred
       pred  := curr

   return curr
```

- Claim: *fibiter*(i) = f_i for i=0, 1, ...

Calculating Fibonacci Numbers

Analysis of the iterative algorithm

- Let $C_{iter}(i)$ be the cost of computing f_i using *fibiter*(i)

  (remember: cost = “number of arithmetic operations”)

- Then
$$\begin{aligned} C_{iter}(0) &= 0 \\ C_{iter}(1) &= 0 \\ \boxed{C_{iter}(i) = i-1 \quad \text{for} \quad i \geq 2} \end{aligned}$$

- This means: **Linear complexity of *fibiter*** !
  - Expl: $i = 100 \Rightarrow C_{iter}(100) = 99 \quad \text{or} \quad i = 200 \Rightarrow C_{iter}(200) = 199$

- So *fibiter* is much better than *fibrec*

- But: Is it possible to find an even better algorithm than *fibiter* ?

Calculating Fibonacci Numbers

3) An iterative squaring algorithm *fibisq*

- Remark: We will see later what “iterative squaring” means

- Let us start with some preparations:

  Prove that the following holds for i=2, 3, ...

$$\begin{pmatrix} f_{i-1} \\ f_i \end{pmatrix} = A^{(i-1)} * \begin{pmatrix} f_0 \\ f_1 \end{pmatrix}$$

  where

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

Calculating Fibonacci Numbers

- Function pow implements the *iterative squaring*:

  pow(a, n) = a^n for arbitrary a and n=1, 2, ...

```
pow(a,n)

   if n=1 then return a

   if n even then
      x := pow(a,n/2)
      return x*x

   if n odd then
      x := pow(a,(n-1)/2)
      return x*x*a
```

Calculating Fibonacci Numbers

Analysis of the runtime of *pow*

- (Note: For $x \in \mathbb{R}$, $\lfloor x \rfloor$ (to be read "floor of x") denotes the largest integer that is $\leq x$.)

- For given n there are $\lfloor \log_2 n \rfloor + 1$ recursice calls to function *pow*

- Each call involves at most
  - 1 integer subtraction *(only if odd)*
  - 1 integer division
  - 2 multiplications (of values of type(a)) (only 1 multiplication, if n even)

$$\boxed{Cost_{pow} \leq 4 \cdot (\lfloor \log_2(n) \rfloor + 1) + 1}$$

Calculating Fibonacci Numbers

The iterative squaring algorithm fibisq

- For i=0, 1, ... compute

```
fibisq(i)

    if i=0 then return 0
    if i=1 then return 1
```

$$A := \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$$

```
    P := pow(A,i-1)

    return P[2,2]
```

$f_i = p_{22}$ of matrix P with

$$P = \begin{pmatrix} p_{11} & p_{12} \\ p_{21} & p_{22} \end{pmatrix}$$

- Claim: *fibisq* (i) = f_i for i=0, 1, ...

Calculating Fibonacci Numbers

Analysis of *fibisq*:

- The only arithmetic operations involved are in the call pow(A, i-1)
- There are $\lfloor \log_2(i-1) \rfloor + 1$ such calls, each involving <u>at most</u>
  - 1 integer subtraction
  - 1 integer division
  - 2 multiplications of (2x2) matrices
    (each involving 8 integer multiplications and 4 integer additions)

  summing up to at most 26 arithmetic operations per call

- So the cost $C_{isq}(i)$ of *fibisq* is:

$$\boxed{C_{isq}(i) \leq 26 \cdot (\lfloor \log_2(i-1) \rfloor + 1) + 1}$$

- ⇨ **Logarithmic complexity of *fibisq***!   (“<u>*Divide and Conquer*</u>”)

Calculating Fibonacci Numbers

Interpretation

- Cost comparison of $C_{rec}(i)$ vs. $C_{iter}(i)$ vs. $C_{isq}(i)$ for calculating the i-th Fibonacci number

i = 50: $10^7 \le C_{rec}(50) \le 10^{16}$ vs. $C_{iter}(50) = 49$ vs. $C_{isq}(50) \le 157$

i = 100: $10^{14} \le C_{rec}(100) \le 10^{30}$ vs. $C_{iter}(100) = 99$ vs. $C_{isq}(100) \le 183$

i = 200: $10^{29} \le C_{rec}(200) \le 10^{61}$ vs. $C_{iter}(200) = 199$ vs. $C_{isq}(200) \le 209$

i = 300: $10^{44} \le C_{rec}(300) \le 10^{91}$ vs. $C_{iter}(300) = 299$ vs. $C_{isq}(300) \le \underline{235}$

i = 400: $10^{59} \le C_{rec}(400) \le 10^{121}$ vs. $C_{iter}(400) = 399$ vs. $C_{isq}(400) \le \underline{235}$

Calculating Fibonacci Numbers

Interpretation

- The runtimes of the three algorithms computing the same function (Fibonacci number) differ significantly!
- If we assume that arithmetic operation takes 1 microsecond, the following table gives the maximal value i for which f_i can be computed in a given time (1 ms, …, 1 h) using the respective method (based on measurements):

	1ms	1s	1min	1h
recursive	14	28	37	45
iterative	500	$5 \cdot 10^5$	$3 \cdot 10^7$	$2 \cdot 10^9$
iterative squaring	10^{12}	$10^{12,000}$	$10^{700,000}$	10^{10^6}

Calculating Fibonacci Numbers

Is our analysis realistic?

- We only counted arithmetic operations
- We assumed that all arithmetic operations take the same time
- The time to perform one arithmetic operation may depend on the value of the arguments involved

  (it takes longer to add two 1000-digit numbers
  than to add two 10-digit numbers)

- But the trend for large values of i is very clear!