School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Complexity examples

Overview

- Analysis of recursive algorithms: simple methods in this course
- Examples for finding Complexity
  - Fibonacci
  - GCD

Complexity Analysis

- How to prove things:
  - If we want to prove Big-oh (and other) notation, the only thing we can rely on is the formal definition.
  - Sometimes, if we want to disprove some statement, at least one counterexample will work.
- We will see examples later.

Complexity Analysis

- How to analyze recursive functions
- It can be quite complicated.
- If the recursion is really just a thinly veiled loop, the analysis is usually trivial

```
int fac(int n){

  if(n <= 1){
    return 1;
  }else{

    return n*fac(n-1);
  }
}
```

Complexity Analysis - Recursion

- However, when more than one recursive call is done in the function, it is difficult to convert the recursion into a simple loop structure.
- Recursive Fibonacci has a growth rate of:
- $1/\sqrt{5}\ (\ ((1+\sqrt{5})/2)^n - (1-\sqrt{5}/2)^n)$
- We can show that it is in $O(2^n)$ and $\Omega(2^{n/2})$

```
int fib(int n){

  if(n<=1)
    return 1;
  else
    return fib(n-1)+fib(n-2);
}
```

Euclid's Algorithm

- Computing the greatest common divisor

```
int recursiveGCD(int a, int b) {
   if (b==0) return a;
   return gcd(b, a%b);
 }
```

```
int gcd(int a, int b){

  while(b != 0){
    int reminder = a % b;
    a = b;
    b = reminder;
  }

  return a;
}
```

GCD

```
int gcd(int a, int b){

  while(b != 0){
    int reminder = a % b;
    a = b;
    b = reminder;
  }

  return a;
}
```

- The number of iterations depends on the values of a and b.
- Values of a and b are monotonically decreasing.
- After 1 iteration we have $a = \min\{a,b\}$.
- We can prove that after two iterations, the value of a is at most half of what it has been before.
  - Therefore, the complexity is $O(\log \min\{a,b\})$.

GCD

Theorem: Let a and b, $a >= b$, be inputs to gcd(int a, int b). Then after at most two iterations of the while-loop we obtain a^* where $a^* <= a/2$.

Sketch of proof by case distinction:

- Value of a is monotonically decreasing and we always have $a >= b$.
- Assume that $b > a/2$ then $b' = a \% b <= a/2$ and $a' = b$ holds in the next iteration, and $a^* = a' \% b' = b \% b' <=$ $a/2$ after two iterations due to % operation.
- Assume $b <= a/2$ then $a^* = b <= a/2$ after one iteration.

THE UNIVERSITY of ADELAIDE