Tirgul 2

(some more)

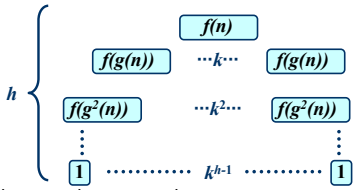
Asymptotic Analysis

Asymptotic Analysis

- Previously in asymptotic analysis:
  - Big O
  - Big Ω
  - Big Θ
  - Recurrence trees
- Today:
  - More recurrence
  - Summation

Recurrence

- Last week we looked at recurrence trees:



- There are two other popular approaches:
  - Guess the asymptotic complexity and prove by induction
  - The *Master* theorem (not today)

Recurrence back to Hanoi

- The input of the problem is: s, t, m, k
- The size of the input is k+3 ~ k (the number of disks).
- Denote the size of the problem k=n.
- Reminder:

```
H(s,t,m,k) {
  if (k > 1) {
    H(s,m,t,k-1)
    moveDisk(s,t)
    H(m,t,s,k-1)
  } else { moveDisk(s,t)}
}
```

- What is the running time of the "Towers of Hanoi"?

Recurrences

Guess and prove by induction

- Denote the run time of a recursive call to input with size n as $T(n)$:
  - `H(s,m,t,k-1)` takes $T(n\text{-}1)$ time
  - `moveDisk(s,t)` takes $T(1)$ time
  - `H(m,t,s,k-1)` takes $T(n\text{-}1)$ time
- We can express the running-time as a recurrence:
  $T(n) = 2T(n\text{-}1) + 1$
  $T(1) = 1$
- How do we solve this ?
- One method to solve recurrence is **guess** and prove by **induction.**

Step 1: "guessing" the solution

$T(n) = 2T(n\text{-}1) + 1$
$= 2[2T(n\text{-}2)+1] + 1 = 4T(n\text{-}2) + 3$
$= 4[2T(n\text{-}3)+1] + 3 = 8T(n\text{-}3) + 7$ ...

When repeating k times we get:
$T(n) = 2^kT(n\text{-}k) + (2^k - 1)$
Now take k=n-1. We'll get:
$T(n) = 2^{n-1}T(1) + 2^{n-1} - 1 = 2^n - 1$

Step 2: proving by induction

- If we guessed right, it will be easy to prove by induction that $T(n) = 2^n - 1$
- For n=1: $T(1)= 2\text{-}1=1$ (and indeed $T(1)=1$)
- Suppose $T(n\text{-}1) = 2^{n-1} - 1$. Then:
  $T(n) = 2T(n\text{-}1) + 1 = 2(2^{n-1} - 1) + 1$
  $= 2^n - 2 + 1 = 2^n - 1$
- So we conclude that: $T(n) = O(2^n)$

Recurrence: Another Example

$T(n) = 2\ T(n/2) + 1$
$T(1) = 1$
$T(n) = 2T(n/2) + 1$
$= 2\ (2T(n/4) + 1) + 1 = 4T(n/4) + 3$
$= 4\ (2T(n/8) + 1) + 3 = 8T(n/8) + 7$

- We get: $T(n) = k\ T(n/k)+(k\text{-}1)$
- For $k=n$ we get $T(n)= n\ T(1)+n\text{-}1=2n\text{-}1$
  Now proving by induction is very simple.
  $T(n) = O(?)$
- Can you think an algorithm that behaves like that?

Recurrence: Another Example

- Another way: "guess" right away $T(n) <= c\ n - b$ (for some b and c we don't know yet), and try to prove by induction:
- The base case:
  For n=1: $T(1)=c\text{-}b$, which is true when $c\text{-}b=1$
- The induction step:
  Assume $T(n/2)=c(n/2)\text{-}b$ and prove for $T(n)$.
  $T(n) <= 2\ (c(n/2) - b) + 1 = c\ n - 2b + 1 <= c\ n - b$
  (the last step is true if $b>=1$).
- Conclusion: $T(n) = O(n)$

Common mistakes

- Let us prove by induction that $2^n = O(n)$ (Wrong!)
- For $n=1$, $2^1=2=O(1)$ ✓
- Assume true for n, we will prove for $n+1$:
  $T(n+1) = 2^{n+1} = 2*2^n = 2T(n) = 2*O(n) = O(n)$
- What did we do wrong?
- Remember that the big-O notation is only a short hand for a definition. We should use the full definition

Common mistakes

- Let us use the full definition
- We want to prove that there exist c and n_0, such that for every $n > n_0$ it holds that $2^n \le cn$
- The induction step will be:
  $T(n+1) = 2^{n+1} = 2*2^n \le 2cn$
- But it is not true that $2cn \le c(n+1)$

Summations

(from Cormen, ex. 3.2-2., page 52)

- Find the asymptotic upper bound of the sum

$$\left(\lceil n/1\rceil+\lceil n/2\rceil+\lceil n/4\rceil+\lceil n/8\rceil+\ldots+\lceil 1\rceil\right)=\sum_{k=0}^{\lfloor \log(n)\rfloor}\left\lceil n/2^k\right\rceil$$

$$\sum_{k=0}^{\lfloor \log(n)\rfloor}\left\lceil n/2^k\right\rceil \le \sum_{k=0}^{\lfloor \log(n)\rfloor}\left(n/2^k+1\right)=\sum_{k=0}^{\lfloor \log(n)\rfloor}\left(n/2^k\right)+\sum_{k=0}^{\lfloor \log(n)\rfloor}(1)$$

$$\le \left(\log n+1\right)+n\sum_{k=0}^{\infty}1/2^k = 1+\log n+2n = O(n)$$

- note how we “got rid” of the integer rounding
- Is the sum also $\Omega(n)$?
  (yes)

Summations (example 2)

(Cormen, ex. 3.1-a., page 52)

- Find an asymptotic upper bound for the following expression:
- $f(n) = \sum_{k=1}^{n} k^r$ where r is a constant
- (remember that you already saw the case of r=1 in class)

$$f(n) = 1^r + 2^r + 3^r + \ldots + n^r \le n \cdot n^r = n^{r+1} = O(n^{r+1})$$

- Note that $n \cdot n^r \neq O(n^r)$
- Note that when a series increases polynomially the upper bound would be the last element but with an exponent increased by one.
- Can we find a tighter bound?

Example 2 (Cont.)

- A good way of finding tight bound is finding a lower bound that equals the upper bound.
- In this case we will find a lower bound for an even n and a lower bound for an odd n.
- For an even n:

$$f(n) = 1^r + 2^r + 3^r + \ldots + n^r \ge (n/2) \cdot (n/2)^r = \frac{n^{r+1}}{2^{r+1}} = \frac{1}{2^{r+1}} n^{r+1} = \Omega(n^{r+1})$$

- For an odd n:

$$f(n) \ge \left(\frac{n-1}{2}\right) \cdot \left(\frac{n+1}{2}\right)^r \ge \frac{n-1}{2}\left(\frac{n}{2}\right)^r = \frac{1}{2^{r+1}} n^r \cdot (n-1)$$

$$> \frac{1}{2^{r+1}} n^r \cdot \frac{n}{2} = \frac{1}{2^{r+2}} n^r = \Omega(n^{r+1})$$

Example 2 (Cont.)

- Thus: $f(n) = \Theta(n^{r+1})$
  so our upper bound was tight!
- Note that proving that $f(n) \neq O(n^r)$ is not enough! (why?)