Lecture 4: Analysis of Algorithms

1. Introduction: An essential tool to design an efficient and suitable algorithm is the "Analysis of Algorithms". There is no magic formula it is simply a matter of judgment, intuition and experience. Nevertheless, theses are some basic techniques that are often useful, such as knowing how to deal with control structures and recursive equations.
2. Control Structures Analysis: Eventually analysis of algorithms proceeds from the inside out. Determine fist, the time required by individual instructions, then combine these tines according to the control systems that combine the instructions in the program.
   2.1. Sequencing: Let P_1 and P_2 be two fragments of an algorithm. They way be single instruction. They maybe single instructions or complicated sub-algorithms. Let t_1 and t_2 be the times taken by P_1 and P_2. t_1 and t_2 may depend on various parameters, such as the instance size. The sequencing rule says that the time required to compute "P_1 and P_2" is simply $t_1 + t_2$. By the maximum rule, this time is in $\theta(\max(t_1, t_2))$. Despite its simplicity, applying this rule is sometimes less obvious than it may appear. It could happen that one of the parameters that control t_2 depends on the results of the computation performed P_1.
   2.2. "For" Loops: They are the easiest loops to analyze.
   For $i \leftarrow 1$ to m do P(i)
   By a convention we'll adopt: m=0 means P(i) is not executed at all, (not an error). P(i) could depend on the size. Of course, the easiest case is when it doesn't. Let t be the time required to compute P(i) and the total time required is $l=mt$. Usually this approach is adequate, but there is a potential pitfall: We didn't consider the time for the "loop control". After all our for loops is shorthand for something like the following while loop.

```
i ← 1
while i ≤ m do
    P(i)
    i ←i + 1
```

   In the worst situations it is reasonable to count the test $i \le m$ at unit cost and the same thing with the instructions $i \leftarrow i + 1$ and the sequencing operations "go to" implicit in the while loop. Let "c" be the upper bound on the time required by each of the operations:

l	$\le$	c	for $i \leftarrow 1$
	$+$	$(m+1)c$	tests $i \le m$
	$+$	mt	execution of P(i)
	$+$	mc	execution of $i \leftarrow i + 1$
	$+$	mc	sequencing operations
l	$\le$	$(t+3c)m+2c$	

   This time is clearly bounded below by mt. If c is negligible compared to t, our previous estimate that l is roughly equal to mt is justified.

   The analysis of for loop is more interesting when the time t(i) required for P(i) runs as a function of I and or also on size n.

So: for i $\leftarrow$1 to m do P(i) takes a time given by, $\sum t(i)$ (ignoring the time taken by the loop control).

Example: Computing the Fibonacci Sequence,

```
Function Fibitir(n)
 i ←1, for j ←0
for k ←1 to n do
        j ←i+j, j ←i-j
return j
```

If we count all arithmetic operations at unit cost, the instructions inside the "for" loop take constant time. Let this time be bounded by the same constant c. Not taking control loop into account, the time taken by the "for" loop is bounded by: nc, we conclude that the algorithm takes a time in O(n). Similar, reasoning yields that this time is in $\Omega(n)$, thus $\theta(n)$.
This is not reasonable. j $\leftarrow$i+j is increasingly expensive each time around the loop. To know the approximate time, we need to know the length of the integers involved at any given kth trip around the loop. j and i are respectively kth and k-1st values of the Fibonacci sequence, sometimes denoted f_k anf f_{k-1} where f_k can be expressed as:

$$f_k = \frac{1}{\sqrt{5}}(\Phi^k - \hat{\Phi}^k)$$

Where $\Phi = \frac{(1+\sqrt{5})}{2}$ is the golden ratio, and $\hat{\Phi} = \frac{(1-\sqrt{5})}{2}$

When k is large, $\hat{\Phi}^k$ is negligible and $f_k \approx \phi^k$ and the size of f_k is in the order of k. Thus: $f_k \in \theta(k)$ and the kth iteration takes a times $\theta(k-1) + \theta(k)$, which is of course $\theta(k)$.

Let c be a constant such that this time is bounded above by ck for all $k \geq 1$. If we again neglect the time required by the loop control, for the instructions before and after the loop we conclude:

$$\sum_{k=1}^{n} ck = c\sum_{k=1}^{n} k = \frac{cn(n+1)}{2} \in O(n^2)$$

Similar reasoning yields that this time is in $\Omega(n^2)$ and therefore it is in $\theta(\ n^2)$. And this shows the crucial difference whether or not we count arithmetic operations at unit cost.

2.3. Recursive Calls:

The analysis of recursive algorithms is straight forward to a certain point. Simple inspection of the algorithm often gives rise to a recurrence equation that “mimics” the flow of control in the algorithm. General techniques on how to solve or how to transform the equation into simpler non-recursive equations will be seen later.

2.4. “While” and “Repeat” loops:

Theses two types of loops are usually harder to analyze than “for” loops because there is no obvious a priori way to know how many times we shall have to go around the loop. The standard technique for analyzing these loops is to find a function of the variables involved where value decreases each time around. To determine how many times the loop is repeated, however, we need to understand better how the value of this function decreases. An alternative approach to the analysis of “while” loops consist of treating them like recursive algorithms. We illustrate both techniques with the same example, The analysis of “repeat” loops is carried out similarly.

Left of on page 31