Analysis of Algorithms

- Why to analyze algorithms?
- Analyzing Control Structure
  - Sequencing
    - Let P1 and P2 be two fragments of an algorithm
    - Sequencing Rule
    - Maximum Rule
  - Can we always consider P1 and P2 independent?

Analysis of Algorithms

- Analyzing Control Structure
  - For Loop

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

  - If we neglect the time needed for loop control and assume that P(i) does not actually depend on i and each time it is performed at a cost of t.

Analysis of Algorithms

- Analyzing Control Structure
  - For Loop

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

  - If we do not neglect the time needed for loop control and assume that P(i) does not actually depend on i.

$$
\begin{array}{l}
i \leftarrow 1 \\
\textbf{while } i \le m \textbf{ do} \\
\quad P(i) \\
\quad i \leftarrow i + 1
\end{array}
$$

In most situations, it is reasonable to count at unit cost the test $i \le m$, the instructions $i \leftarrow 1$ and $i \leftarrow i + 1$, and the sequencing operations (**go to**) implicit in the **while** loop. Let c be an upper bound on the time required by each of these operations. The time ℓ taken by the loop is thus bounded above by

$$
\begin{array}{lllll}
\ell & \le & & c & \text{for } i \leftarrow 1 \\
 & & + & (m+1)c & \text{for the tests } i \le m \\
 & & + & mt & \text{for the executions of } P(i) \\
 & & + & mc & \text{for the executions of } i \leftarrow i+1 \\
 & & + & mc & \text{for the sequencing operations} \\
 & \le & & (t+3c)m + 2c. &
\end{array}
$$

Analysis of Algorithms

- Analyzing Control Structure
  - For Loop

$$\textbf{for } i \leftarrow 1 \textbf{ to } m \textbf{ do } P(i)$$

  - If we neglect the time needed for loop control and assume that P(i) depends on i and cost of executing P(i) each time is t(i).

$$\sum_{i=1}^{m} t(i).$$

Analysis of Algorithms

- Analyzing Control Structure
  - For Loop
    - Iterative Fibonacci

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,.........

function *Fibiter*(n)
  $i \leftarrow 1$; $j \leftarrow 0$
  for $k \leftarrow 1$ **to** n **do** $j \leftarrow i + j$
    $i \leftarrow j - i$
  return j

- If we count all arithmetic operations at unit cost and not taking loop control into account
- If we does not count additions involved in the computation of Fibonacci sequence at unit cost

$$\begin{cases} f_0 = 0; f_1 = 1 & \text{and} \\ f_n = f_{n-1} + f_{n-2} & \text{for } n \geq 2. \end{cases}$$

$$f_n = \frac{1}{\sqrt{5}}[\phi^n - (-\phi)^{-n}],$$

where $\phi = (1 + \sqrt{5})/2$ is the *golden ratio*. Since $0 < \phi^{-1} < 1$, the term $(-\phi)^{-n}$ can be neglected when n is large. Hence the value of f_n is roughly $\phi^n/\sqrt{5}$, which is exponential in n.

Analysis of Algorithms

- Analyzing Control Structure
  - For Loop
    - Iterative Fibonacci

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

Analysis of Algorithms

- Analyzing Control Structure
  - Recursive Calls

function $Fibrec(n)$
    if $n < 2$ **then return** n
    else return $Fibrec(n-1) + Fibrec(n-2)$

$$T(n) = \begin{cases} a & \text{if } n = 0 \text{ or } n = 1 \\ T(n-1) + T(n-2) + h(n) & \text{otherwise} \end{cases}$$

- Addition at unit cost or not at the unit cost.
- Running time

Analysis of Algorithms

- Analyzing Control Structure
  - While Loops
    - Why are they harder to analyze than for loops?
    - Identifying and using a function of variables whose value decreases each time around
    - Example of Binary_Search

```
function Binary_Search(T[1..n], x)
    {This algorithm assumes that x appears in T}
    i ← 1; j ← n
    while i < j do
        { T[i] ≤ x ≤ T[j] }
        k ← (i + j) ÷ 2
        case x < T[k]: j ← k − 1
             x = T[k]: i, j ← k {return k}
             x > T[k]: i ← k + 1
    return i
```

Analysis of Algorithms

- Analyzing Control Structure
  - While Loops
    - In binary search the number of elements of array T under consideration continuously decreases. Let this number be d and d = j – i + 1
    - Let d and $\hat{d}$ be the value of j – i + 1 before and after the iteration under consideration.

We use i, j, $\hat{\imath}$ and $\hat{\jmath}$ similarly. If $x < T[k]$, the instruction

Analysis of Algorithms

- Analyzing Control Structure
  - While Loops

If $x < T[k]$, the instruction "$j \leftarrow k - 1$" is executed and thus $\hat{\imath} = i$ and $\hat{\jmath} = [(i + j) \div 2] - 1$. Therefore,

$$\hat{d} = \hat{\jmath} - \hat{\imath} + 1 = (i + j) \div 2 - i \le (i + j)/2 - i < (j - i + 1)/2 = d/2.$$

Similarly, if $x > T[k]$, the instruction "$i \leftarrow k + 1$" is executed and thus

$$\hat{\imath} = [(i + j) \div 2] + 1 \text{ and } \hat{\jmath} = j.$$

Therefore,

$$\hat{d} = \hat{\jmath} - \hat{\imath} + 1 = j - (i + j) \div 2 \le j - (i + j - 1)/2 = (j - i + 1)/2 = d/2.$$

Finally, if $x = T[k]$, then i and j are set to the same value and thus $\hat{d} = 1$; but d was at least 2 since otherwise the loop would not have been reentered. We conclude that $\hat{d} \le d/2$ whichever case happens, which means that the value of d is at least halved each time round the loop. Since we stop when $d \le 1$, the process must eventually stop, but how much time does it take?

Analysis of Algorithms

- Analyzing Control Structure
  - While Loops

To determine an upper bound on the running time of binary search, let d_ℓ denote the value of $j - i + 1$ at the end of the ℓ–th trip round the loop for $\ell \geq 1$ and let $d_0 = n$. Since $d_{\ell-1}$ is the value of $j - i + 1$ *before* starting the ℓ–th iteration, we have proved that $d_\ell \leq d_{\ell-1}/2$ for all $\ell \geq 1$. It follows immediately by mathematical induction that $d_\ell \leq n/2^\ell$. But the loop terminates when $d \leq 1$, which happens at the latest when $\ell = \lceil \lg n \rceil$. We conclude that the loop is entered at most $\lceil \lg n \rceil$ times. Since each trip round the loop takes constant time, binary search takes a time in $O(\log n)$.

Analysis of Algorithms

- Analyzing Control Structure
  - While Loops
    - Alternative Approach : Treating while loops like recursive algorithms
    - Let t(d) be the maximum time needed to terminate the while loop when there are at most d elements under consideration
    - Let b be the constant time required to go round the loop once
    - Let c be the time to determine that the loop is finished when d=1 and return, then

$$t(d) \le \begin{cases} c & \text{if } d = 1 \\ b + t(d \div 2) & \text{otherwise} \end{cases}$$

Analysis of Algorithms

- Using a barometer
  - What is a barometer instruction?
  - Advantages
  - What should we do in case of nested for loops?
    - Sorting Example: T[1…n], U[1…s]

```
procedure pigeonhole(T[1..n])
    {Sorts integers between 1 and 10000}
    array U[1..10000]
    for k ← 1 to 10000 do U[k] ← 0
    for i ← 1 to n do
        k ← T[i]
        U[k] ← U[k]+1
    i ← 0
    for k ← 1 to s do
        while U[k] ≠ 0 do
            i ← i + 1
            T[i] ← k
            U[k] ← U[k]−1
```

Analysis of Algorithms

- Using a barometer
  - What should we do in case of nested for loops?
    - Sorting Example: T[1…n], U[1…s]
      - If we select any of the instructions in the inner loop as barometer then
      - Total number of times they are executed is therefore $\sum_{k=1}^{s} U[k]$
      - Is it true?
        - Let a be the time needed for the test U[k] ≠ 0 and let b be the time taken by one execution of the instructions in the inner loop including implicit sequencing operation.
        - To execute the inner loop completely for given k $t_k = (1 + U[k])a + U[k]b$
        - The complete process takes

```
i ← 0
for k ← 1 to s do
    while U[k] ≠ 0 do
        i ← i + 1
        T[i] ← k
        U[k] ← U[k] − 1
```

$c + \sum_{k=1}^{s}(d + t_k)$, where c and d are new constants to take account of the time needed to initialize and control the outer loop, respectively. When simplified, this expression yields $c + (a + d)s + (a + b)n$. We conclude that the process takes a time in $\Theta(n + s)$

Analysis of Algorithms

- Using a barometer
  - What should we do in case of nested for loops?
    - Sorting Example: T[1…n], U[1…s]
    - Shall we use barometer to analyze this pigeon-hole sorting?
      - $U[k] \neq 0$