Asimptotik Notasyon ve Temel Verimlilik Sınıfları

Hafta 2

(Büyüme Sırası) Order of growth

- En önemlisi : $n \to \infty$'a giderken algoritmanın performansı hangi sınırlarda bunu anlayabilmeks

- Örnek:
  - İki katı kadar hızlı bir bilgisayarda algoritma ne kadar hızlanıyor?
  - Girdi boyutu iki katına çıktığında algoritma ne kadar yavaşlıyor?

Values of some important functions as $n \to \infty$

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

Asimptotik Büyüme Dereceleri

Fonksiyonların büyüme hızlarını karşılaştırmak için kullanılan, sabit çarpanları ve küçük girdi boyutlarını yoksayan, bir yöntem.

- O($g(n)$): $g(n)$ fonksiyonundan daha hızlı büyümeyen $f(n)$ fonksiyonlarını kapsar
- Θ($g(n)$): $g(n)$ fonksiyonları ile aynı derecede büyüyen $f(n)$ fonksiyonlarını gösterir.
- Ω($g(n)$): en az $g(n)$ fonksiyonları kadar hızda büyüyen $f(n)$ fonksiyonlarını belirtmek için kullanılır.

Big-oh

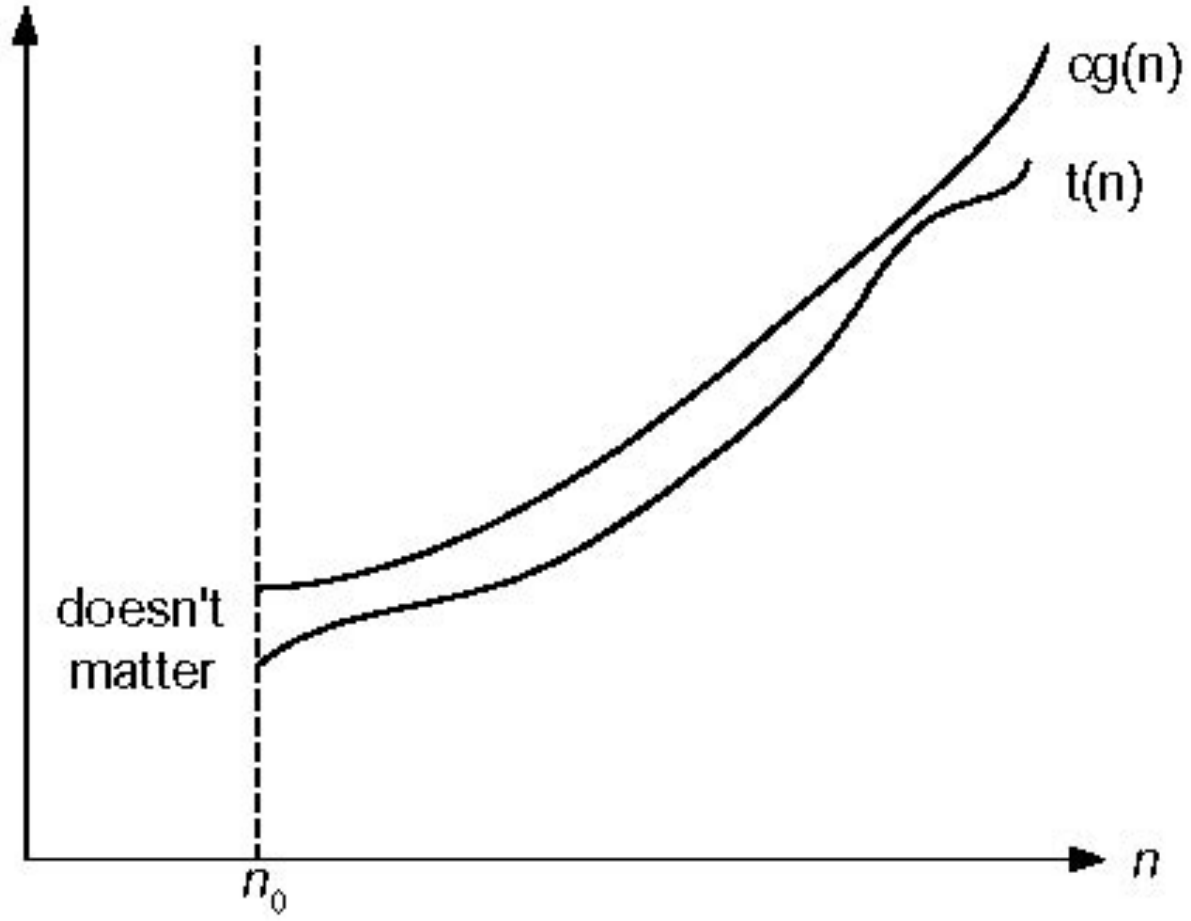


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

Big-omega

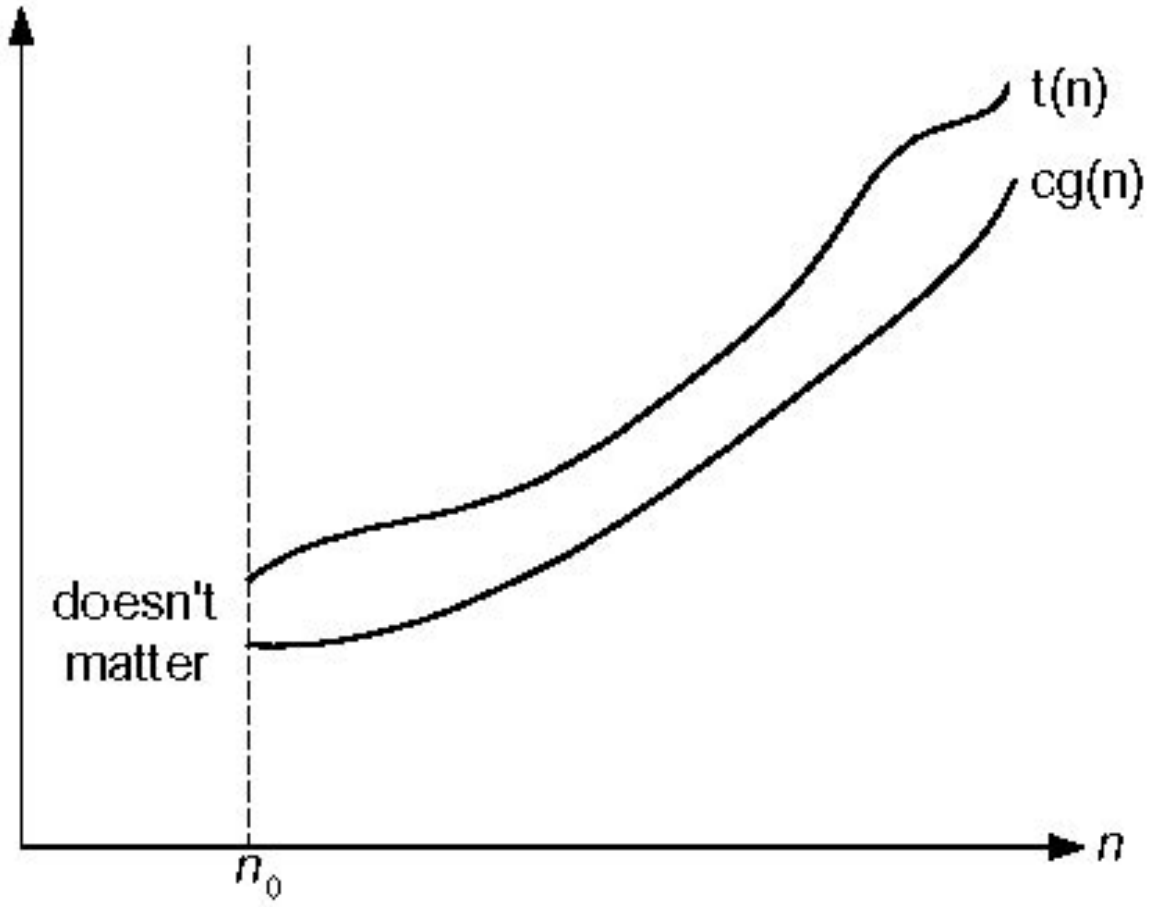


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

Big-theta

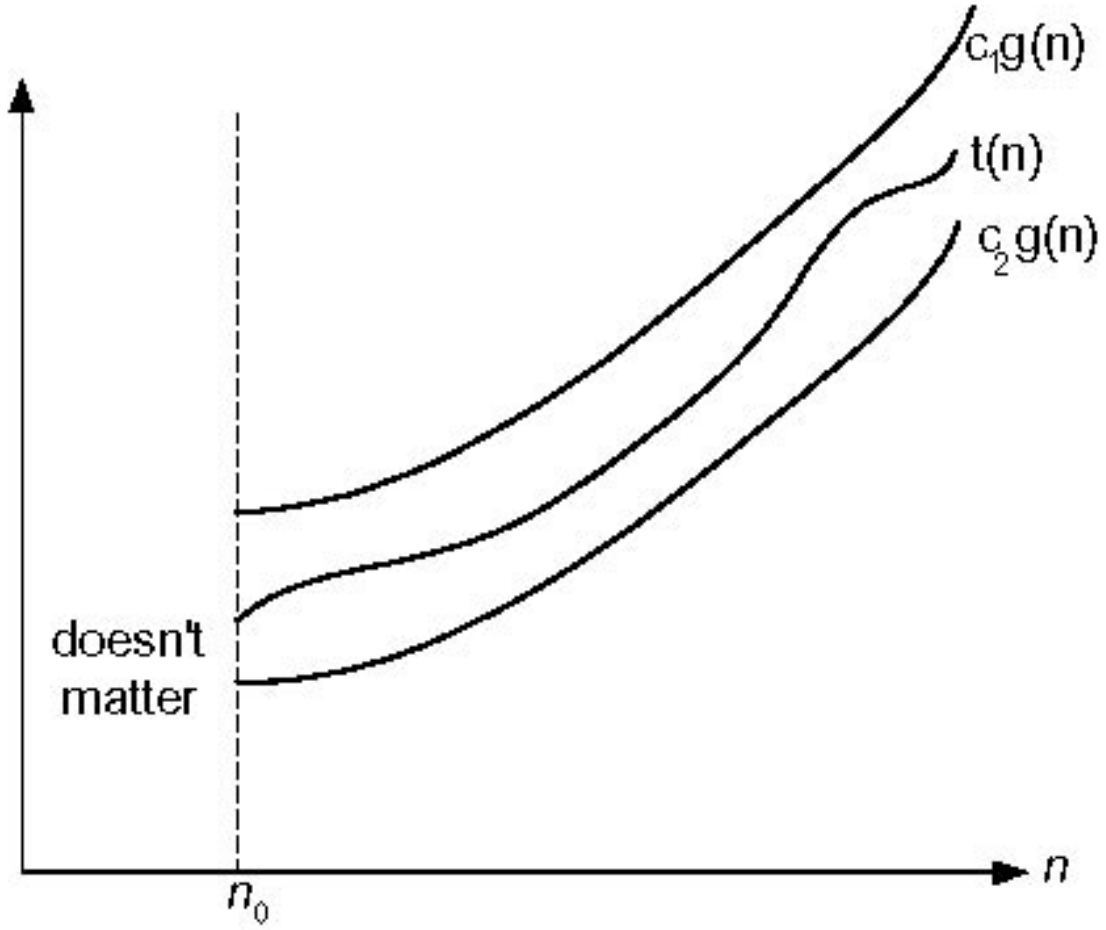


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Big *O* Formal Tanımı

Tanım: $f(n) \in O(g(n))$ ise, f(n) fonksiyonunun büyüme derecesi, $g(n)$'in büyüme sabit bir sayı ile çarpımının büyüme derecesinden küçüktür.

,

$$f(n) \leq c\, g(n)\ ,\ \forall\ \ n \geq n_0$$

Eşitsizliğini sağlayan pozitif bir sabit c ve pozitif bir tamsayı n_0 vardır

Örnekler:

- $10n$ is $O(n^2)$ $\quad$ $5n+20$ is $O(n)$

Big O'nun Özellikleri

- $f(n) \in O(f(n))$
- $f(n) \in O(g(n))$ iff $g(n) \in \Omega(f(n))$
- If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$

  $a \leq b$ eşitsizliğindekine benzer bir şekilde

- If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then

  $$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$$

Ω- Formal Tanımı

- **Tanım:** $f(n) \in \Omega(g(n))$ ise, f(n) fonksiyonunun büyüme derecesi, $g(n)$'in sabit bir sayı ile çarpımının büyüme derecesinden büyük veya eşittir.

$$f(n) \geq c\, g(n)\ ,\ \forall\ n \geq n_0$$

Eşitsizliğini sağlayan pozitif bir sabit c ve pozitif bir tamsayı n_0 vardır.

Örn: $n^3 \in \Omega(n^2)$

Θ Formal Tanımı

- **Tanım:** $f(n) \in \Theta(g(n))$ ise, f(n) fonksiyonunun büyüme derecesi, $g(n)$ fonksiyonun bir sabit katından yüksek aynı zamanda $g(n)$ fonksiyonun bir sabit katından da düşük olmaktadır.

$$c_1\, g(n) \leq f(n) \leq c_2\, g(n)\ ,\ \forall\ n \geq n_0$$

Eşitsizliğini sağlayan pozitif sabit c_1, c_2 *sayıları* ve pozitif bir tamsayı n_0 vardır.

Örn: $\frac{1}{2}n(n-1) \in \Theta(n^2)$

Limit Kullanarak Nüyüme Derecesi Belirleme

$\lim_{n\to\infty} T(n)/g(n) =$

0 order of growth of ***T(n)*** < order of growth of ***g(n)***

c* > 0** order of growth of ***T(n) = order of growth of ***g(n)***

∞ order of growth of ***T(n)*** > order of growth of ***g(n)***

Examples:

- **$10n$ vs. n^2**
- **$n(n+1)/2$ vs. n^2**

L'Hôpital's kuralı ve Stirling's formülü

L'Hôpital's rule: If $lim_{n \to \infty} f(n) = lim_{n \to \infty} g(n) = \infty$ and the derivatives f', g' exist, then

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$$

Example: $\log n$ vs. n

Stirling's formula: $n! \approx (2\pi n)^{1/2} (n/e)^n$

Example: 2^n vs. $n!$

Bazı Önemli Fonksiyonların Büyüme Dereceleri

- Tüm logaritmik fonksiyonlar $\log_a n$ aynı asimptotik sınıfa sahiptir.

  $\Theta(\log n)$ logarithmanın tabanı $a > 1$ önemli değil.

- Aynı derece *k'ye sahip olan tüm polinomlar* aynı asimptotik sınıfa sahiptir. :
- $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 \in \Theta(n^k)$

- Üstel fonksiyonlar a^n a değerine göre farklı büyüme sınıfına aittir.

- order $\log n$ < order n^α (α>0) < order a^n < order $n!$ < order n^n

Temel Asimptotik Verimlilik Sınıfları (Basic asymptotic efficiency classes)

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n or linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Recursive olmayan Algoritmaların Time Efficiency'sini Analiz Etmek için Genel Plan

- Girdi boyutu için düşünülen parametreye karar verme,
- Algoritmanın temel işlemini (*basic operation) belirle.*
- Ekötü(*worst)*, ortalama(*average)*, ve en iyi(*best)* durumlarına karar ver.
- Toplamda kaç farklı temel işlemin çalıştırıldığını say (toplam formülünü belirle)
- Toplamı kurallara göre basite indirge.

Kullanışlı Toplam Formülleri ve Kurallar

$\sum_{l \le i \le u} 1 = 1+1+ \cdots +1 = u - l + 1$

Örn: $\sum_{1 \le i \le n} 1 = n - 1 + 1 = n \in \Theta(n)$

$\sum_{1 \le i \le n} i = 1+2+ \cdots +n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$

$\sum_{1 \le i \le n} i^2 = 1^2+2^2+ \cdots +n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$

$\sum_{0 \le i \le n} a^i = 1 + a + \cdots + a^n = (a^{n+1} - 1)/(a - 1)$ for any $a \neq 1$

In particular, $\sum_{0 \le i \le n} 2^i = 2^0 + 2^1 + \cdots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$\sum(a_i \pm b_i) = \sum a_i \pm \sum b_i \qquad \sum c a_i = c \sum a_i \qquad \sum_{l \le i \le u} a_i = \sum_{l \le i \le m} a_i + \sum_{m+1 \le i \le u} a_i$

Example 1: Maximum element

ALGORITHM *MaxElement*$(A[0..n-1])$

```
//Determines the value of the largest element in a given array
//Input: An array A[0..n − 1] of real numbers
//Output: The value of the largest element in A
maxval ← A[0]
for i ← 1 to n − 1 do
    if A[i] > maxval
        maxval ← A[i]
return maxval
```

Example 1: Maximum element

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n).$$

Example 2: Element uniqueness problem

ALGORITHM *UniqueElements*$(A[0..n-1])$

```
//Determines whether all the elements in a given array are distinct
//Input: An array A[0..n − 1]
//Output: Returns "true" if all the elements in A are distinct
//         and "false" otherwise
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i] = A[j] return false
return true
```

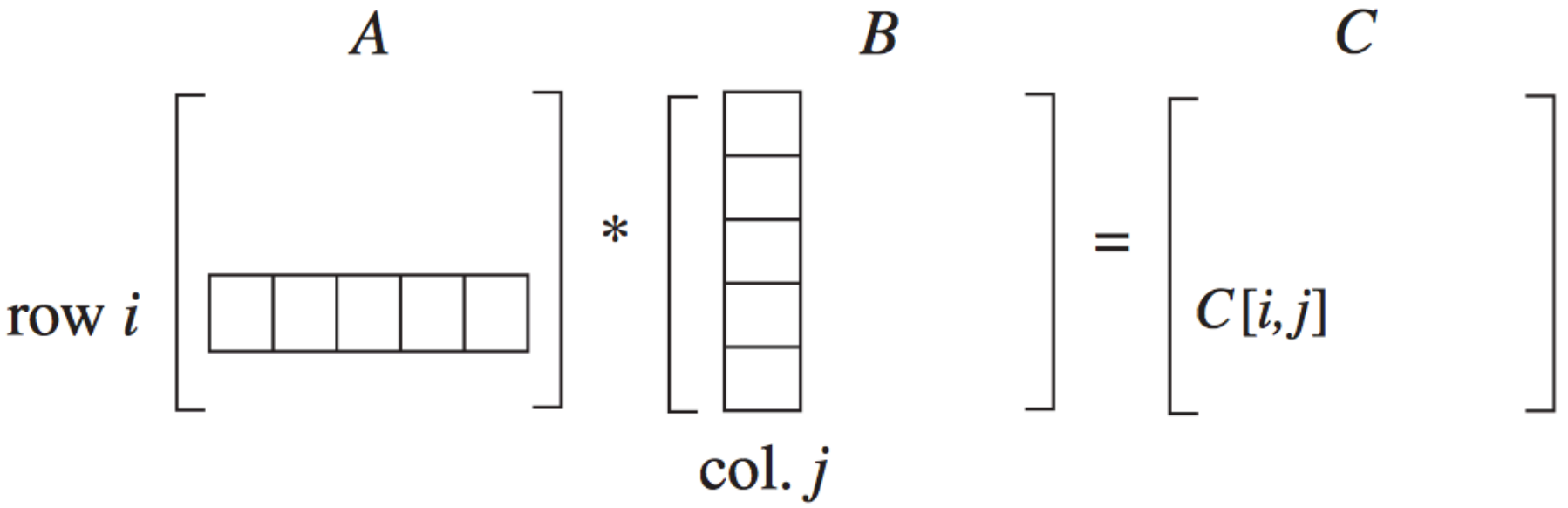
Example 2: Element uniqueness problem

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

Example 3: Matrix multiplication



Example 3: Matrix multiplication

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

//Multiplies two n-by-n matrices by the definition-based algorithm
//Input: Two n-by-n matrices A and B
//Output: Matrix $C = AB$
for $i \leftarrow 0$ **to** $n-1$ **do**
    for $j \leftarrow 0$ **to** $n-1$ **do**
        $C[i, j] \leftarrow 0.0$
        for $k \leftarrow 0$ **to** $n-1$ **do**
            $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$
return C

Example 3: Matrix multiplication

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

Example 4: Gaussian elimination

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix}, \quad \left[\begin{array}{cccc|c} a'_{11} & a'_{12} & \cdots & a'_{1k} & b'_1 \\ 0 & a'_{22} & \cdots & a'_{2k} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & a'_{kk} & b'_k \end{array}\right].$$

$$x_i = \frac{1}{a'_{ii}} \left(b'_i - \sum_{j=i+1}^{k} a'_{ij} x_j \right).$$

Example 4: Gaussian elimination

Algorithm *GaussianElimination*(A[0..n-1,0..n-*1*])

//Implements Gaussian elimination of an n-by-(n+1) matrix A

for $i \leftarrow$ 0 to n - 2 do
    for $j \leftarrow$ i + 1 to n - 1 do
        for $k \leftarrow$ i to n do
            $A[j,k] \leftarrow A[j,k] - A[i,k] * A[j,i] / A[i,i]$

Find the efficiency class and a constant factor improvement.

Example 5: Counting binary digits

ALGORITHM *Binary*(n)

```
//Input: A positive decimal integer n
//Output: The number of binary digits in n's binary representation
count ← 1
while n > 1 do
    count ← count + 1
    n ← ⌊n/2⌋
return count
```

It cannot be investigated the way the previous examples are.

Von Neumann’s Neighborhood