Algorithm 1:

Computing L and U factors of A → می دانیم پیچیدگی زمانی این مرحله برابر با $\frac{2}{3}n^3$ و در واقع $O(n^3)$ است.

m بار تکرار می‌شود:

```
for i=1 to m
    use L and U factors to solve Ax_i=b_i
end
```

$Ax_i=b_i$ * → این قسمت در آخر توضیح داده شده که چرا $6n^2$ است

پس می‌توان نتیجه گرفت تعداد عملیات لازم برای اجرای این الگوریتم $6mn^2+O(n^3)$ است یعنی $c_2=6$ و $c_1=0$

Algorithm 2:

Computing L and U → $O(n^3)$

n بار تکرار می‌شود:

```
for i=1 to n
    use L and U factors to solve Ay_i=e_i
end
```

$Ay_i=e_i$ * → $6n^2$ (در آخر توضیح داده شده)

m بار تکرار:

```
for i=1 to m
    x_i = A^{-1}.b_i
end
```

$x_i = A^{-1}\cdot b_i$ ** → این ضرب یک ماتریس $n\times n$ در یک ماتریس $n\times 1$ است و از مرتبه $2n^2$ است. (در آخر توضیح می‌دهم)

پس این الگوریتم دارای تعداد عملیات $6n^3+2mn^2+O(n^3)$ است.
$c_1=6$ و $c_2=2$

اولی → $O(n^3)+6mn^2$

دومی → $O(n^3)+2mn^2+6n^3$

نتیجه می‌گیریم الگوریتم اول بهتر است چون دومی از مرتبه n^3 است (البته به شرط این که m کوچک باشد)

مقایسه → $O(n^3)+6mn^2-\left(O(n^3)+2mn^2+6n^3\right)$

$$= 4mn^2-6n^3 = 2n^2(2m-3n)$$

اگر $2m-3n<0$ باشد اولی بهتر است.

★ use L and U to solve $Ax_i = b_i$

// change L to I

for j = 1 to n-1

  for i = 2 to n

    $b_i = b_i - b_j \times A_{ij}$ (یک ضرب، یک تفریق)

  } $2(n-1)^2$

// change U to I

for j = n to 2

  for i = n-1 to 1

    $k = \frac{A_{ij}}{A_{jj}}$ (یک تقسیم)

    $A_{ij} = 0$ ← این را هم یک عملیات درنظر می‌گیریم

    $b_i = b_i - b_j \times k$ (یک ضرب، یک تفریق)

  end

end

} $4 \times (n-1)(n-1) = 4(n-1)^2$

for i = 1 to n

  $b_i = \frac{b_i}{A_{ii}}$

  $A_{ii} = 1$

end

} $2n$

در مجموع $6(n-1)^2 + 2n$ عملیات لازم است که تقریبا می‌شود $6n^2$

ضرب کردن

```
for i = 1 to n
    for j = 1 to n
        X_i1 = X_i1 + A_ij × b_j1
```

$$X_{i1} = X_{i1} + A_{ij} \times b_{j1}$$

ضرب ($A_{ij} \times b_{j1}$)

جمع ($X_{i1} + A_{ij} \times b_{j1}$)

$2n^2$ = عملیات