Сложност на алгоритъм

Анализ на алгоритъм

- **Ефективност** – изпълнява ли задачата за която е предназначен;
- **Ефикасност** – колко изчислителни ресурси използва алгоритъмът.

Ефективност

- Удовлетворени ли са условията:
  - **Крайност** (анализът е алгоритмично нерешима задача);
  - **Коректност** на резултата (анализът може да бъде практически нерешима задача).

Ефикасност (изчислителна сложност)

- **Необходимо време за изпълнение (времева сложност);**
- Необходима памет за изпълнение (пространствена сложност);
- Точност на резултата;
- Използван мрежови трафик;
- Използвана мощност;
- …

Размер на задачата

- Размерът на задачата „n“ е мярка (най-често касаеща количеството входни/обработвани данни), спрямо която се анализира сложността на алгоритъма

Времева сложност

- T(n) – времето за изпълнение на задачата;
- Анализира се с броя операции за изпълнение на задачата;
- Показва скоростта на нарастване на броя операции спрямо размера на задачата.

Пространствена сложност

- S(n) – паметта за изпълнение на задачата;
- Анализира се с количеството памет, необходимо за изпълнение на задачата;
- Показва скоростта на нарастване на количеството памет спрямо размера на задачата.

Пр: Swap

```
public void Swap(int a, int b)
{
    int temp = a;   //          1
    a = b;          //          1
    b = temp;       //          1
                    // T(n) = 3
}
```

```
public void Swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
                    // a = 1, b = 1, temp = 1
                    // S(n) = 3
}
```

Пр: func

```
public double Func(int a, int b)
{
    return 5 * a + 16 / b;
}
```


```
public double Func(int a, int b)
{
    return 5 * a + 16 / b;
}
```

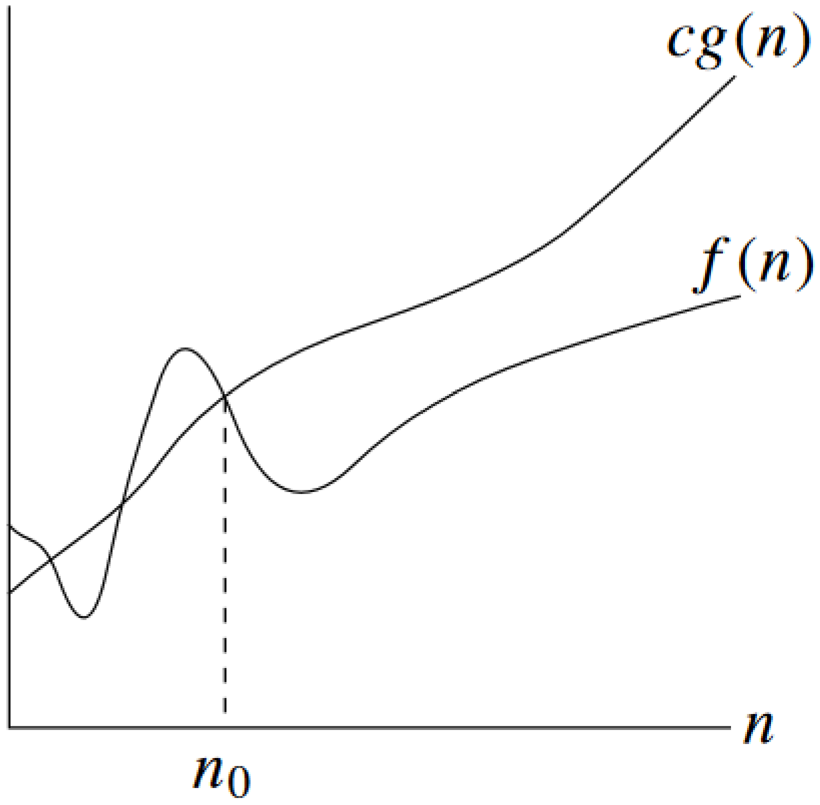

- T(n) = 3 или 1
- S(n) = 5 или 2

Асимптотична сложност - **горна граница**

- $O\big(g(n)\big) = \{f(n) | \exists c > 0, \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq f(n) \leq cg(n)\}$
- $O\big(g(n)\big)$ е множеството от функции f, за които съществува константа с (с>0), такава че $f(n) \leq cg(n)$, за всички достатъчно големи стойности на n , т.е. съществува константа n_0, за която горното неравенство е изпълнено за всяко $n > n_0$.
- $O(g)$ определя множеството на всички функции, които асимптотично нарастват **не по-бързо** от g

- о$\big(g(n)\big) = \{f(n) | \exists c > 0, \exists n_0 \in \mathbb{N}\ \forall n \geq n_0 : 0 \leq f(n) < cg(n)\}$
- о(g) определя множеството на всички функции, които асимптотично нарастват **по-бавно** от g.

Асимптотична сложност - горна граница



Горна граница - пример

- за $f(n) = 3n + 3$: $3n + 3 \leq 10n \Rightarrow O(n)$
- за $f(n) = 3n + 3$: $3n + 3 \leq 7n \Rightarrow O(n)$
- за $f(n) = 3n + 3$: $3n + 3 \leq 10n^2 \Rightarrow O(n^2)$

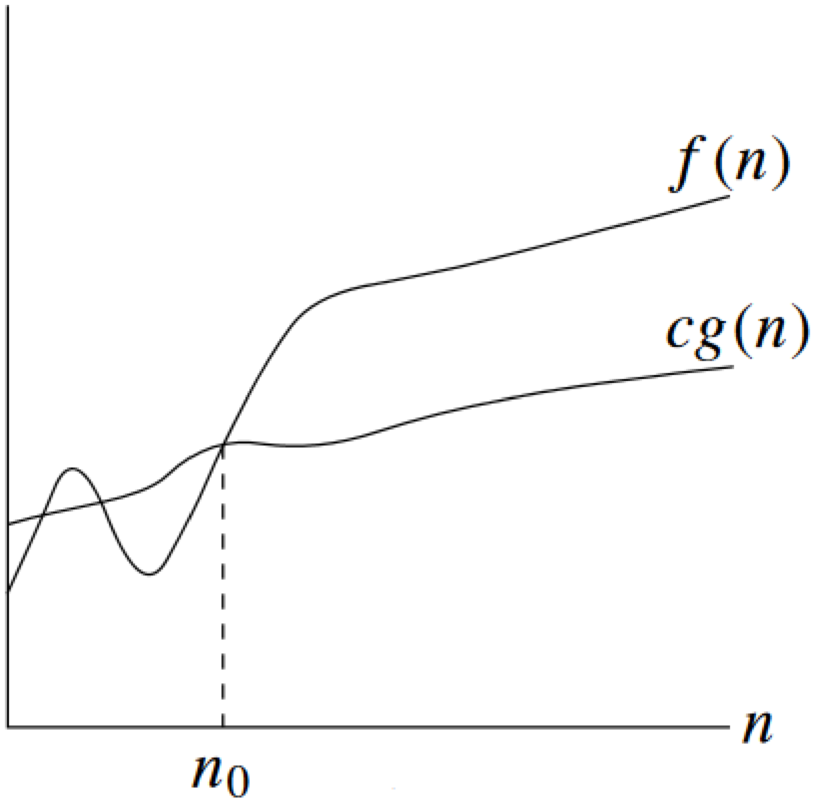
- ако $1 < \log(n) < \sqrt{n} < n < \mathrm{nlog(n)} < n^2 < n^3 < 2^n < 3^n < n^n$
- за $f(n) \to g(n) = n < \mathrm{nlog(n)} < n^2 < n^3 < 2^n < 3^n < n^n$

Асимптотична сложност – **долна граница**

- $\Omega\big(g(n)\big) = \{f(n) | \exists c(c > 0), \exists n_0 \in \mathbb{N}\ \forall n > n_0 : 0 \leq cg(n) \leq f(n)\}$
- $\Omega(g)$ включва всички функции, които асимптотично нарастват **не по-бавно** от g

- $\omega\big(g(n)\big) = \{f(n) | \exists c(c > 0), \exists n_0 \in \mathbb{N}\ \forall n > n_0 : 0 \leq cg(n) < f(n)\}$
- $\omega\ (g)$ включва всички функции, които асимптотично нарастват **по-бързо** от g

Асимптотична сложност – долна граница



Долна граница - пример

- за $f(n) = 3n + 3$: $3n + 3 \geq n \Rightarrow \Omega(n)$
- за $f(n) = 3n + 3$: $3n + 3 \geq 1 \Rightarrow \Omega(1)$

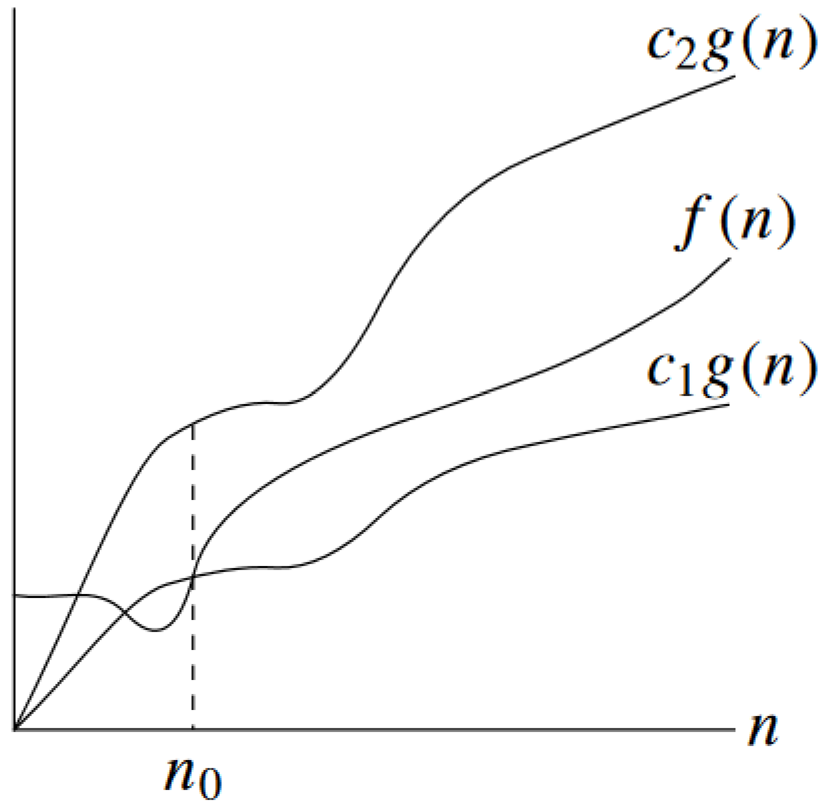
- ако $1 < \log(n) < \sqrt{n} < n < \mathrm{nlog(n)} < n^2 < n^3 < 2^n < 3^n < n^n$
- за $f(n) \rightarrow g(n) = 1 < \log(n) < \sqrt{n} < n$

Асимптотична сложност – **точна граница**

- $\Theta\big(g(n)\big) = \{f(n) | \exists c_1 (c_1 > 0), \exists c_2 (c_2 > 0), \exists n_0(c_1, c_2) \colon \forall n \geq n_0 \colon 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

- $\Theta(g)$ включва всички функции, които асимптотично нарастват **еднакво** с g

Асимптотична сложност – точна граница



Точна граница - пример

- за $f(n) = 3n + 3 \ \rightarrow n \leq 3n + 3 \leq 10n \Rightarrow \Theta\ (n)$

- ако $1 < \log(n) < \sqrt{n} < n < \mathrm{nlog(n)} < \ n^2 < n^3 < 2^n < 3^n < n^n$
- за $f(n) \rightarrow g(n) = n$

Пр: Swap – константна сложност

```
public void Swap(int a, int b)
{
    int temp = a;   // 1
    a = b;          // 1
    b = temp;       // 1
                    // O(1)
}
```

Пр: Sum(int[]) – времева сложност (спрямо всички операции)

```
public int Sum(int[] A)
{
    int s = 0;   1                                   // 1
    for (int i = 0; i < A.Length; i++)               // 2n + 2
                 1              n+1     n
        s += A[i];                                   // n
    return s;      1*n=n                             // 1
            1                                        // T(n) = 3n + 4
                                                     // O(n)
}
```

Пр: Sum(int[]) – пространствена сложност

```
public int Sum(int[] A)                       // A -> n
{
    int s = 0;                                // s -> 1
    for (int i = 0; i < A.Length; i++)        // i -> 1
        s += A[i];
    return s;
                                              // S(n) = n + 2
                                              // O(n)

}
```

Пр: Sum(int[,]) – времева сложност

```
public int[,] Sum(int[,] A, int[,] B, int n)
{
    int[,] C = new int[n, n];
    for (int i = 0; i < n; i++)
        for (int k = 0; k < n; k++)
            C[i, k] = A[i, k] + B[i, k];
    return C;


}
```

Пр: Sum(int[,]) – времева сложност (спрямо всички операции)

```
public int[,] Sum(int[,] A, int[,] B, int n)
{
    int[,] C = new int[n, n];                       // 1
    for (int i = 0; i < n; i++)                     // 2n + 2
        for (int k = 0; k < n; k++)                 // (2n + 2)*n
            C[i, k] = A[i, k] + B[i, k];            // n*n
    return C;                                       // 1
                                                    // T(n) = 3n^2 + 4n + 4
                                                    // O(n^2)

}
```

Пр: Sum(int[,]) – пространствена сложност

```
public int[,] Sum(int[,] A, int[,] B, int n)


{
    int[,] C = new int[n, n];
    for (int i = 0; i < n; i++)
        for (int k = 0; k < n; k++)
            C[i, k] = A[i, k] + B[i, k];
    return C;


}
```

Пр: Sum(int[,]) – пространствена сложност

```
public int[,] Sum(int[,] A, int[,] B, int n)// A -> n^2
                                            // B -> n^2
                                            // n -> 1

{
    int[,] C = new int[n, n];               // C -> n^2
    for (int i = 0; i < n; i++)             // i -> 1
        for (int k = 0; k < n; k++)         // k -> 1
            C[i, k] = A[i, k] + B[i, k];
    return C;
                                            // S(n) = 3n^2 + 3
                                            // O(n^2)

}
```

Пр: Mul(int[,]) – времева сложност

```
public int[,] Mul(int[,] A, int[,] B, int n)
{
    int[,] C = new int[n, n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            C[i, j] = 0;
            for (int k = 0; k < n; k++)
                C[i, j] += A[i, k] *
                           B[j, k];
        }
    return C;

}
```

Пр: Mul(int[,]) – времева сложност

```
public int[,] Mul(int[,] A, int[,] B, int n)
{
    int[,] C = new int[n, n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            C[i, j] = 0;
            for (int k = 0; k < n; k++)
                C[i, j] += A[i, k] *          // n*n*n
                           B[j, k];
        }
    return C;
                                              // O(n^3)
}
```

Пр: Mul(int[,]) – пространствена сложност

```
public int[,] Mul(int[,] A, int[,] B, int n)
{

    int[,] C = new int[n, n];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
        {
            C[i, j] = 0;
            for (int k = 0; k < n; k++)
                C[i, j] += A[i, k] *
                           B[j, k];
        }
    return C;

}
```

Пр: Mul(int[,]) – пространствена сложност

```
public int[,] Mul(int[,] A, int[,] B, int n)     // A->n^2
{                                                // B->n^2
                                                 // n->1
    int[,] C = new int[n, n];                    // C->n^2
    for (int i = 0; i < n; i++)                  // i->1
        for (int j = 0; j < n; j++)              // j->1
        {
            C[i, j] = 0;
            for (int k = 0; k < n; k++)          // k->1
                C[i, j] += A[i, k] *
                           B[j, k];
        }
    return C;
                                                 // O(n^2)
}
```

Пр:

```
for (int i = 0; i < n; i++)  // ignore
{
    //...                        // n
}

for (int i = 0; i < n; i+=2)
{
    //...                        // n/2
}
```

Пр:

```
for (int i = 0; i < n; i++)
{
    for (int k = 0; k < i; k++)
    {
        //..              // 1+2+3+..+n=(n(n+1))/2
    }                     // O(n^2)
}
```

i	k	X
0	0	0
1	0	1
	1	
2	0	2
	1	
	2	
3	0	3
	1	
	2	
	3	

Полезни суми

- $\sum_{i=1}^{n} 1 = n$
- $\sum_{i=a}^{b} 1 = b - a + 1$
- $\sum_{i=1, i=i+k}^{n} 1 \approx \frac{n}{k}$
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Пр:

```
for (int i = 1; i <= n; i *= 2)
    Print("a");
```

i
$1 = 2^0$
$2 = 2^1$
$4 = 2^2$
…
2^k

Нека при $i = 2^k$: $i \leq n$, а при $i = 2^{k+1}$: $i > n$.
Тогава блока ще се изпълни k пъти, но:
$i = 2^k \leq n \Rightarrow k \leq \log_2 n \Rightarrow$

$O(\log_2 n)$

Пр:

```
int p = 0;
for (int i = 1; p <= n; i++)
    p += i;
```

i	p
1	1
2	1+2
…	…
k	1+2+3+…+k

Нека при $i = k$: $p \leq n$, а при $i = k + 1$: $p > n$
Тогава блока ще се изпълни k пъти:

$$\mathrm{p} = \sum_{i=1}^{k} i = \frac{k(k+1)}{2} \leq n \Rightarrow \sqrt{k^2 + k} \leq \sqrt{2n} \Rightarrow$$

$$k \leq \sqrt{2n} \Rightarrow$$

$$O(\sqrt{n})$$

Пр:

```
for (int i = 0; i < n; i++)
{
    //..                        // n
}

for (int i = 0; i < n; i++)
{
    //..                        // n
}

                                // T(n) = 2n
                                // O(n)
```

Пр:

```
if (n > 5)  // 1
    //..    // n
else
    //..    // 2n
            //
            // T(n) = 2n+1
```

Търсене на елемент в масив (T_{min}, T_{max})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

Търсене на елемент в масив (T_{min}, T_{max})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

- $2, 3, 4, 5, 8, 10$
- $T_{min}(6) = 1$
- $T_{max}(6) = 6$

Търсене на елемент в масив (T_{min}, T_{max})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

- A[0],A[1],...,A[n-1]
- $T_{min}(n) = 1$
- $T_{max}(n) = n$

- $O(n)$

Търсене на елемент в масив (T_{av})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

Търсене на елемент в масив (T_{av})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

- $\mathbf{A[0]}, A[1], A[2] \rightarrow 1$

  $A[0], \mathbf{A[1]}, A[2] \rightarrow 2$

  $A[0], A[1], \mathbf{A[2]} \rightarrow 3$

- $T_{av}(3) = \frac{6}{3}$

Търсене на елемент в масив (T_{av})

```
public static int Find(int[] A, int x)
{
    for (int i = 0; i < A.Length; i++)
        if (A[i] == x) // T
            return i;

    return -1;
}
```

- $\boldsymbol{A[0]}, A[1], \dots, A[n-1] \to 1$
  $A[0], \boldsymbol{A[1]}, \dots, A[n-1] \to 2$
  …
  $A[0], A[1], \boldsymbol{\dots, A[n-1]} \to \mathrm{n}$

- $T_{av}(n) = \frac{1}{n}\sum_{i=1}^{n} i = \frac{1}{n}.\frac{n(n+1)}{2} = \frac{n+1}{2}$

Максимален елемент в масив (T_{min}, T_{max})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];        // T


    return res;
}
```

Максимален елемент в масив (T_{min}, T_{max})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];        // T


    return res;
}
```

- $A[2] < A[1] < A[0] \ \rightarrow 0$
- $T_{min}(3) = 0$

- $A[0] < A[1] < A[2] \ \rightarrow 2$
- $T_{max}(3) = 2$

Максимален елемент в масив (T_{min}, T_{max})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];         // T


    return res;
}
```

- $A[0] > A[1] > \cdots > A[n-1] \rightarrow 0$
- $T_{min}(n) = 0$

- $A[0] < A[1] < \cdots < A[n-1] \rightarrow n-1$
- $T_{max}(n) = n-1$

- $O(n)$

Максимален елемент в масив (T_{av})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];        // T


    return res;
}
```

Максимален елемент в масив (T_{av})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];        // T


    return res;
}
```

- $A[0] < A[1] < A[2] \rightarrow 2$

  $A[0] < A[2] < A[1] \rightarrow 1$

  $A[1] < A[0] < A[2] \rightarrow 1$

  $A[1] < A[2] < A[0] \rightarrow 0$

  $A[2] < A[0] < A[1] \rightarrow 1$

  $A[2] < A[1] < A[0] \rightarrow 0$

- $T_{av}(3) = \frac{5}{6}$

Максимален елемент в масив (T_{av})

```
public int Max(int[] A, int n)
{
    int res = A[0];
    for (int i = 1; i < n; i++)
        if (res < A[i])
            res = A[i];        // T

    return res;
}
```

- $\boldsymbol{A}[\boldsymbol{n}] < A[0] < A[1] < \cdots < A[n-1] \rightarrow\ +0$

  $A[0] < \boldsymbol{A}[\boldsymbol{n}] < A[1] < \cdots < A[n-1] \rightarrow\ +0$

  …

  $A[0] < A[1] < \cdots < A[n-1] < \boldsymbol{A}[\boldsymbol{n}] \rightarrow\ +1$

- $T_{av}(n) = T_{\text{cp}}(n-1) + \frac{1}{n}$
- $T_{av}(n) = 0 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \approx \ln(n)$