Algoritmy a dátové štruktúry

Rasťo Královič

kralovic@dcs.fmph.uniba.sk
http://foja.dcs.fmph.uniba.sk/ads

o časovej zložitosti

... z minula

algoritmické úlohy	exaktne opísaný vstup a výstup
poskytovať záruky	program musí fungovať **pre všetky** vstupy
veľkosť vstupu	parameter n (súčasť definície alebo "veľkosť súboru")
časová zložitosť	ako rastie čas s rastúcim n

reklama

CODEFORCES Sponsored by TON	https://codeforces.com/
Sphere online judge	https://www.spoj.com/
Online Judge	https://onlinejudge.org/
CODECHEF	https://www.codechef.com/
K → S → P	https://testovac.ksp.sk/tasks/
Algorithms for Competitive Programming	https://cp-algorithms.com/

dôkazy

```
int mx = a[0];
for (i = 1; i < n; i++) {
    if (a[i] > mx)
        mx = a[i];
}
```

dôkazy

```
int mx = a[0];
for (i = 1; i < n; i++) {
    if (a[i] > mx)
        mx = a[i];
}
```

$\Leftarrow \; mx_i = \max\{a_0, \ldots, a_{i-1}\}$

dôkazy

```
int mx = a[0];
for (i = 1; i < n; i++) {
    if (a[i] > mx)
        mx = a[i];
}
```

$\Leftarrow \; mx_i = \max\{a_0, \ldots, a_{i-1}\}$

```
cnt = 0;
int i, j;
for (i = 0; a[i] != 2; i = j) {
  while (a[i] == 0) i++;
  for (j = i; a[j] == 1; j++)
    cnt++;
}
```

dôkazy

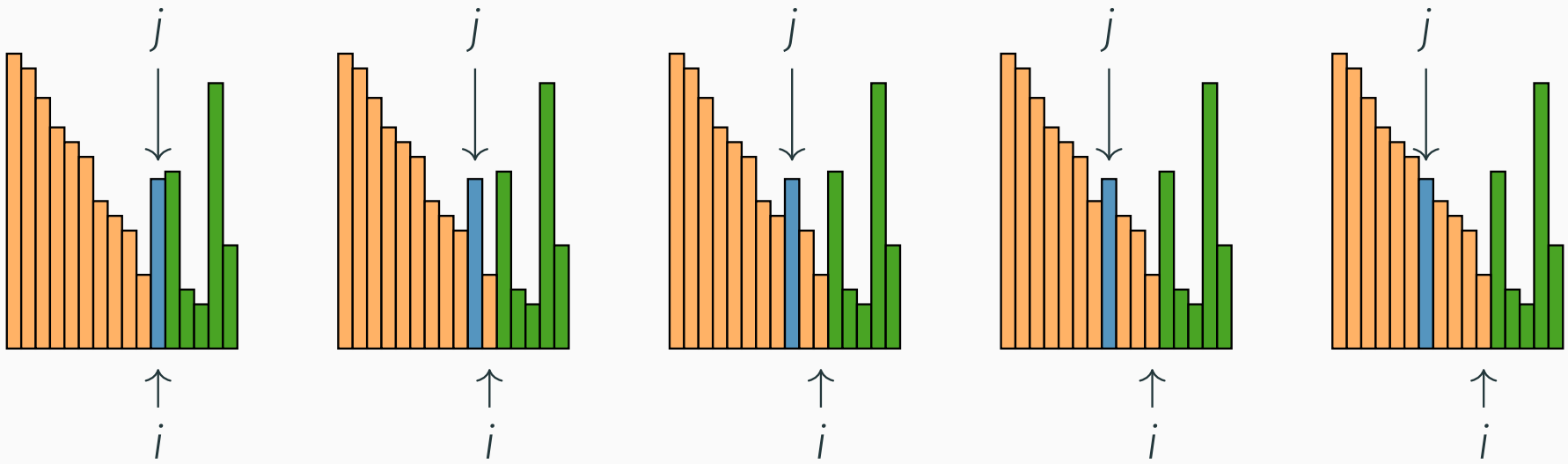
```
int mx = a[0];
for (i = 1; i < n; i++) {
    if (a[i] > mx)
        mx = a[i];
}
```

$\Leftarrow$ $mx_i = \max\{a_0, \ldots, a_{i-1}\}$

```
cnt = 0;
int i, j;
for (i = 0; a[i] != 2; i = j) {
  while (a[i] == 0) i++;
  for (j = i; a[j] == 1; j++)
    cnt++;
}
```

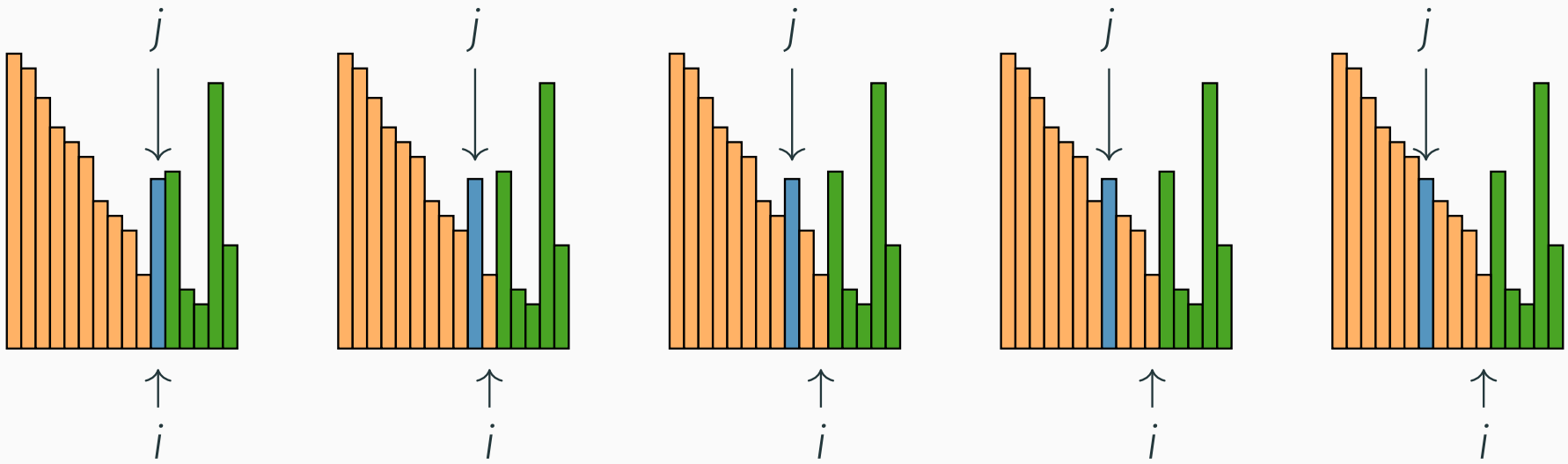
$\Leftarrow$ i je začiatok úseku jednotiek

príklad: triedim čísla z rozsahu 0 . . . 999




```
void vymen(int i, int j) {
  int x; x = a[i]; a[i] = a[j]; a[j] = x;
}

void InsertSort(int n) {
  int i, j;
  for (i = 1; i < n; i++)

    for (j = i; j > 0 && a[j - 1] < a[j]; j--)
      vymen(j - 1, j);
}
```

príklad: triedim čísla z rozsahu 0 . . . 999



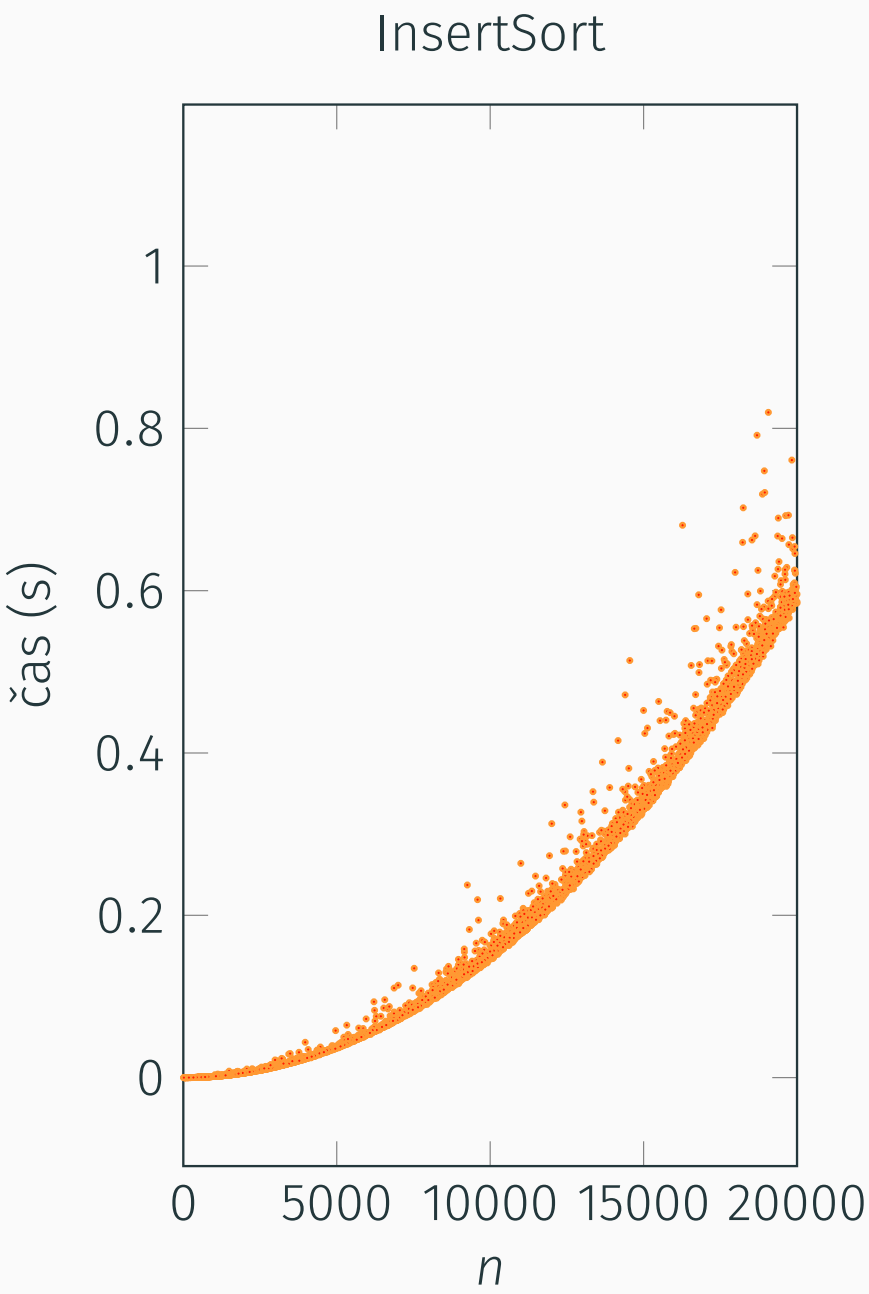

```
void vymen(int i, int j) {
  int x; x = a[i]; a[i] = a[j]; a[j] = x;
}

void InsertSort(int n) {
  int i, j;
  for (i = 1; i < n; i++)
    for (j = i; j > 0 && a[j - 1] < a[j]; j--)
      vymen(j - 1, j);
}
```

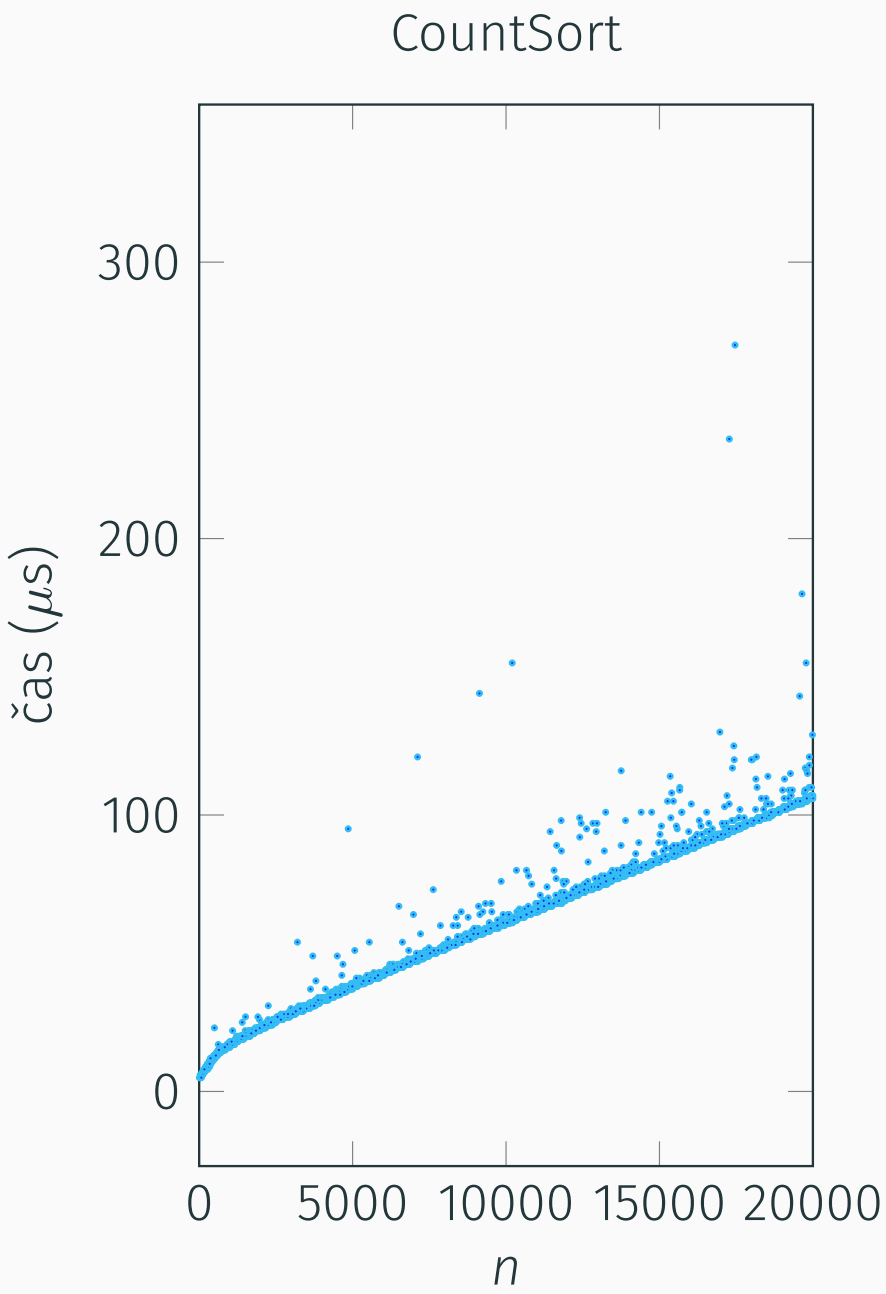
⇐ začiatok poľa je utriedený

príklad: triedim čísla z rozsahu 0 . . . 999

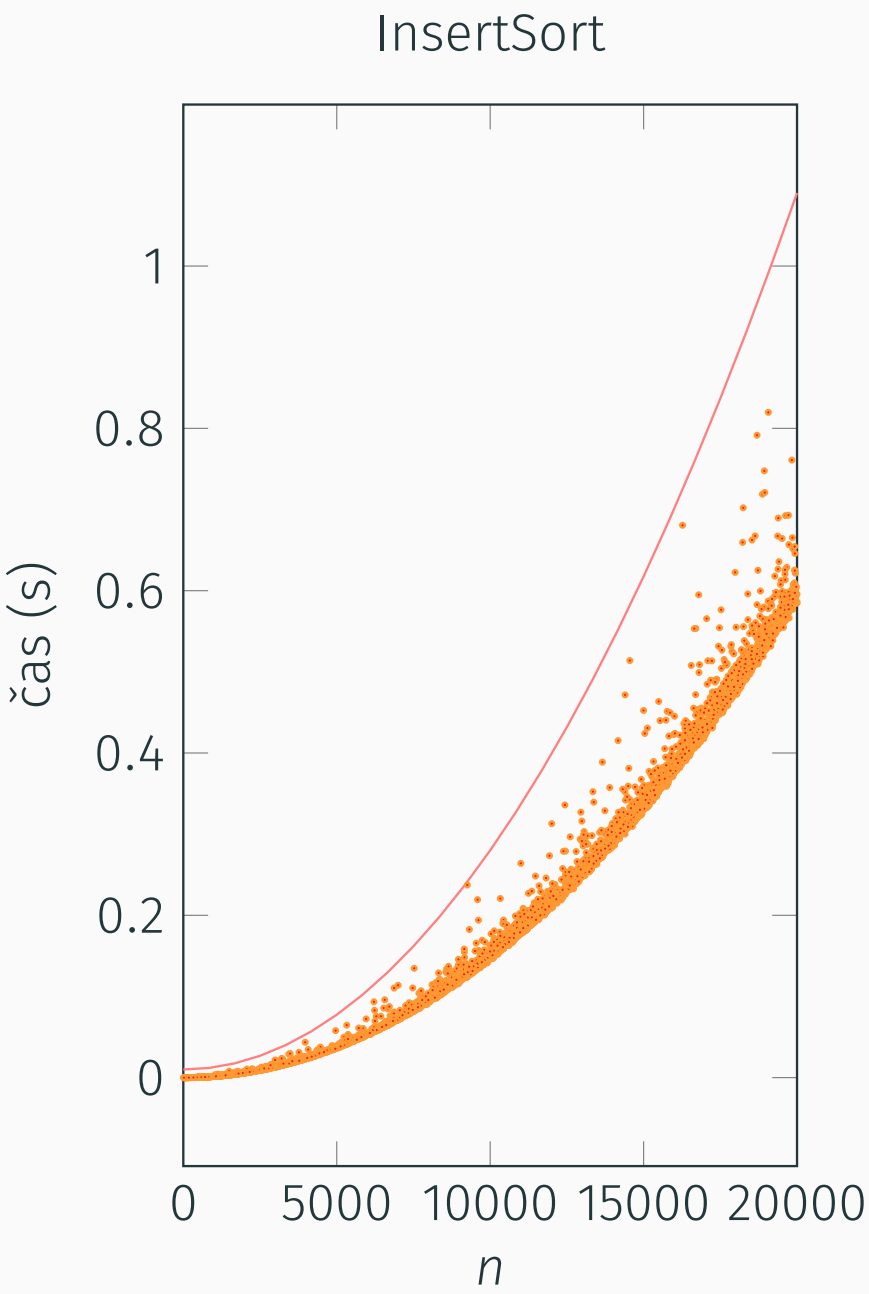
```
void CountSort(int n) {
  int pocty[1000];
  int i, c;

  for (i = 0; i < 1000; i++) pocty[i] = 0;
  for (i = 0; i < n; i++) pocty[a[i]]++;
  i = 0;
  for (c = 999; c>=0 ; c--)
    while (pocty[c] > 0) {
      a[i] = c;
      i++;
      pocty[c]--;
    }
}
```

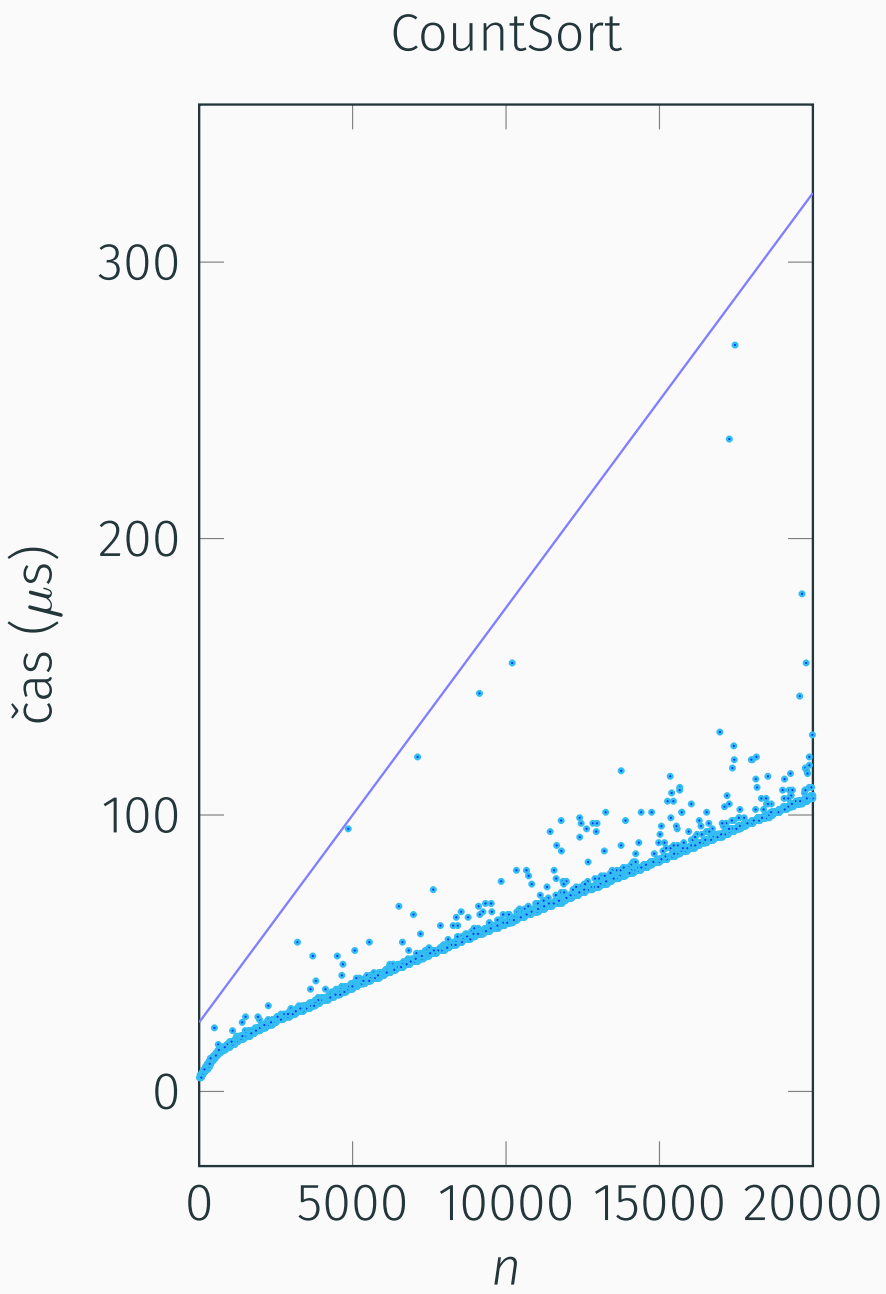
kvadratický vs lineárny



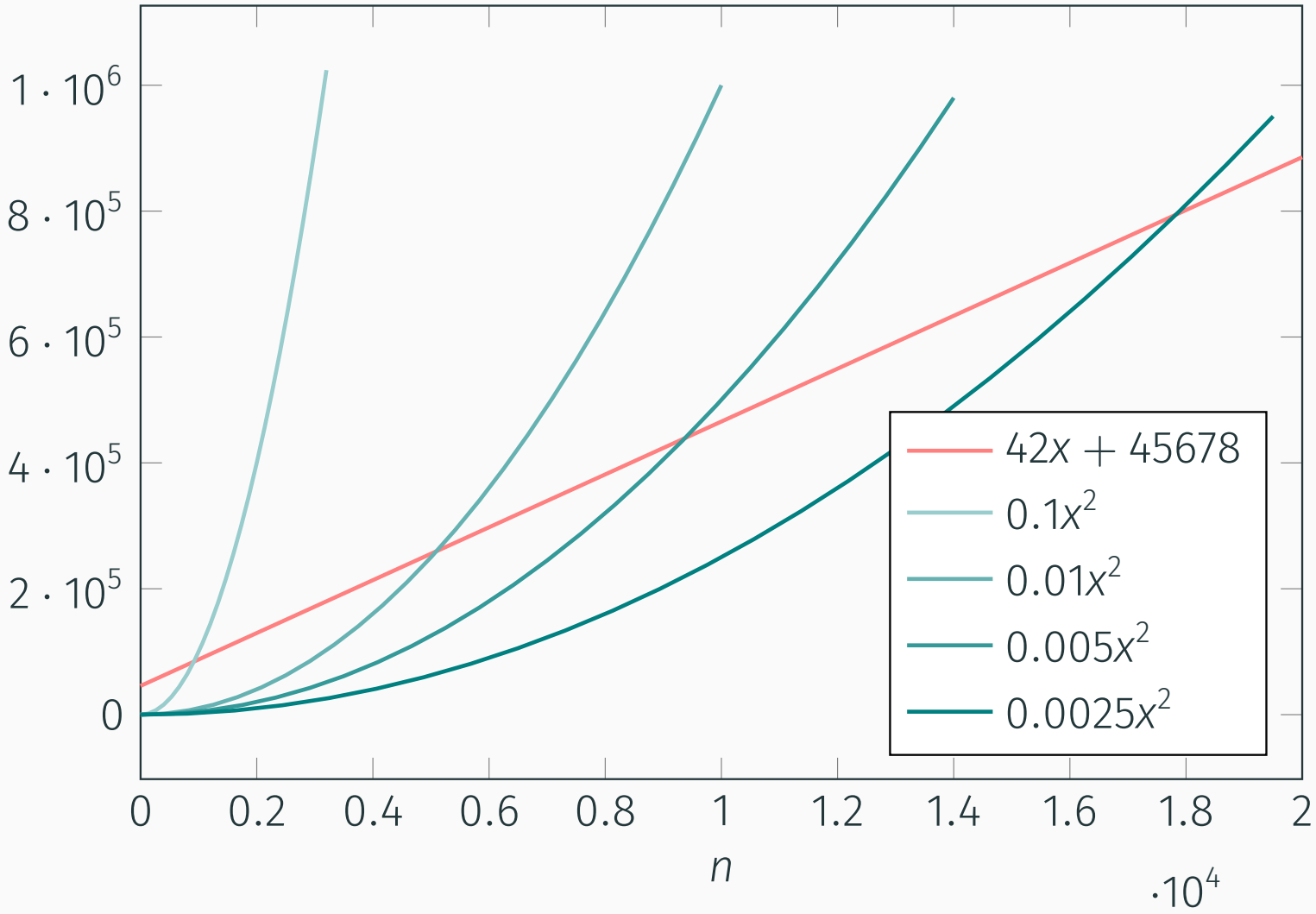


kvadratický vs lineárny : **worst-case časová zložitosť $T(n)$**

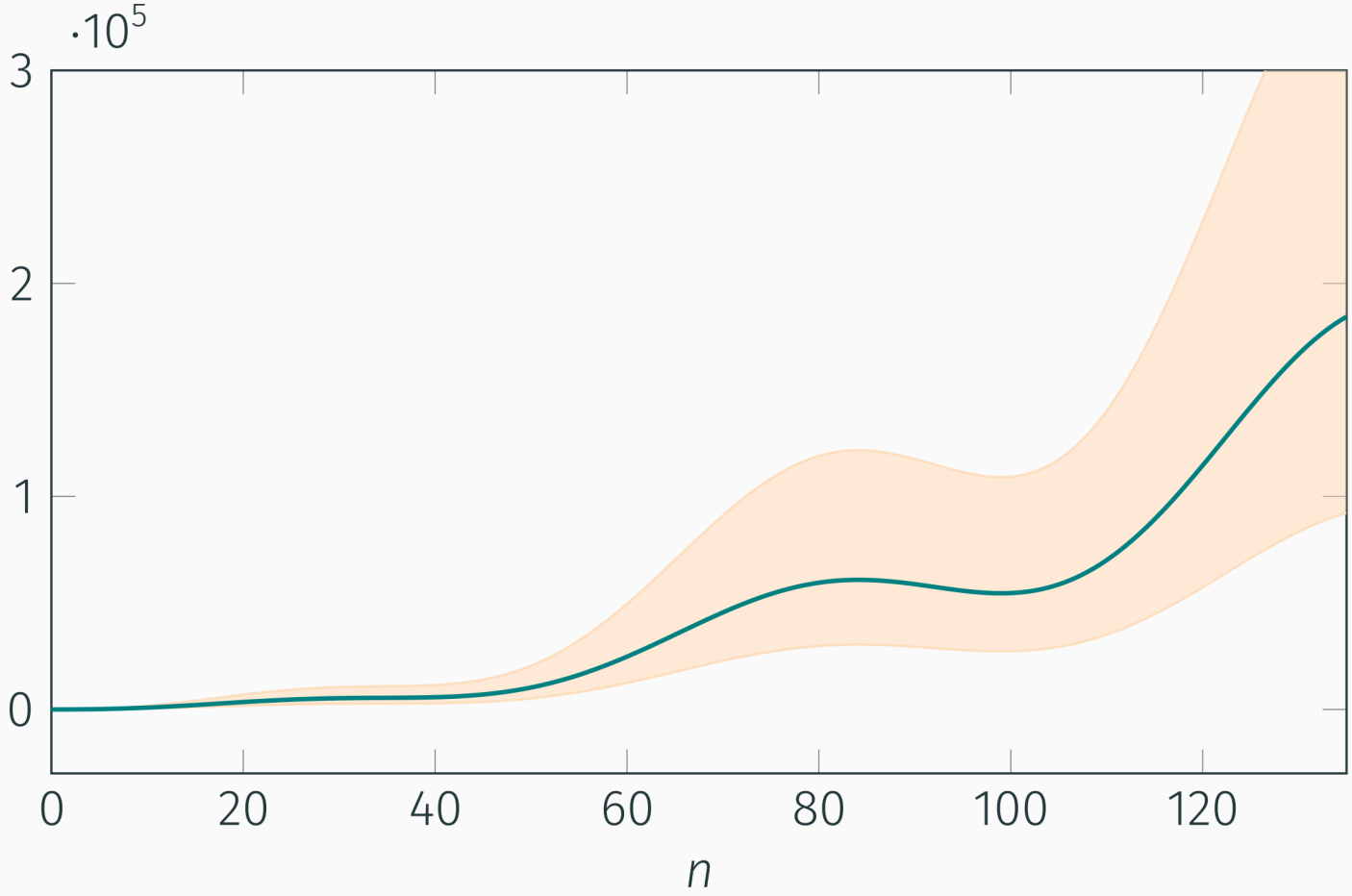




je lineárny vždy lepší?



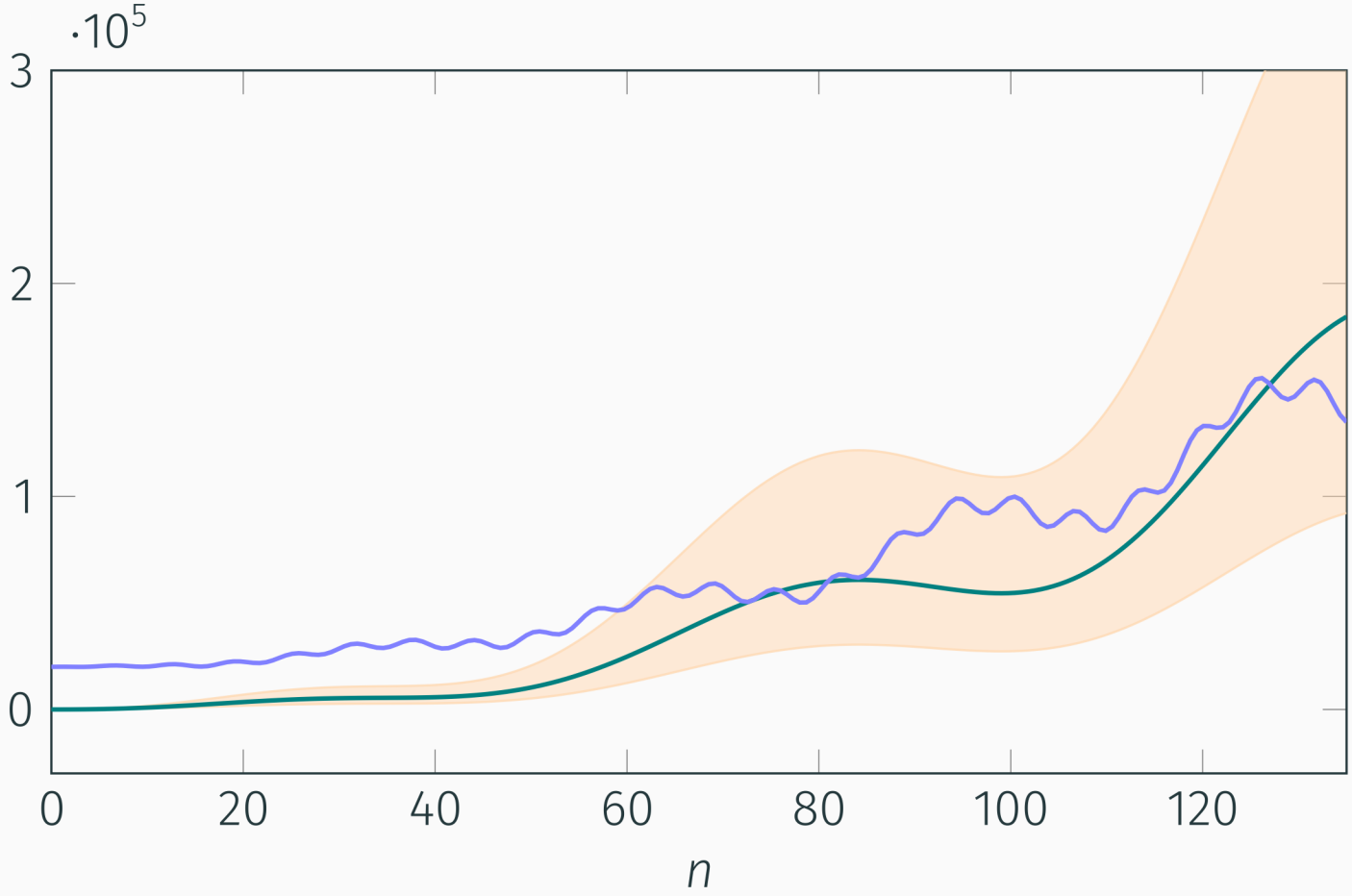
časová zložitosť – porovnávanie funkcií



horný odhad: $f(n) \in O(g(n))$

$$\exists c > 0 \;\; \exists n_0 \in \mathbb{N} \;\; \forall n \geq n_0 \;\; f(n) \leq c \cdot g(n)$$

časová zložitosť – porovnávanie funkcií



horný odhad: $f(n) \in O(g(n))$

$$\exists c > 0 \;\; \exists n_0 \in \mathbb{N} \;\; \forall n \geq n_0 \;\; f(n) \leq c \cdot g(n)$$

príklady

```
int mx = a[0];
for (i = 1; i < n; i++)
    if (a[i] > mx)
        mx = a[i];
```

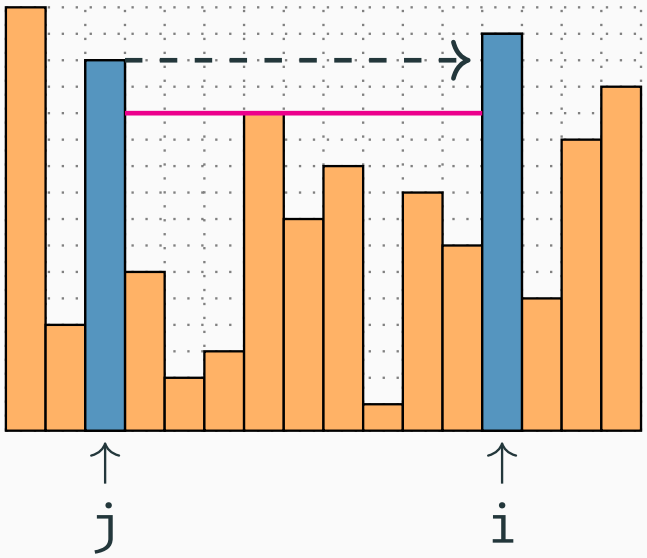
príklady

```
int mx = a[0];
for (i = 1; i < n; i++)
    if (a[i] > mx)
        mx = a[i];
```

```
int s = 0;
for (int i = 1; i < n; i = i * 2)
    for (int j = 1; j * j < i; j++)
        s++;
```

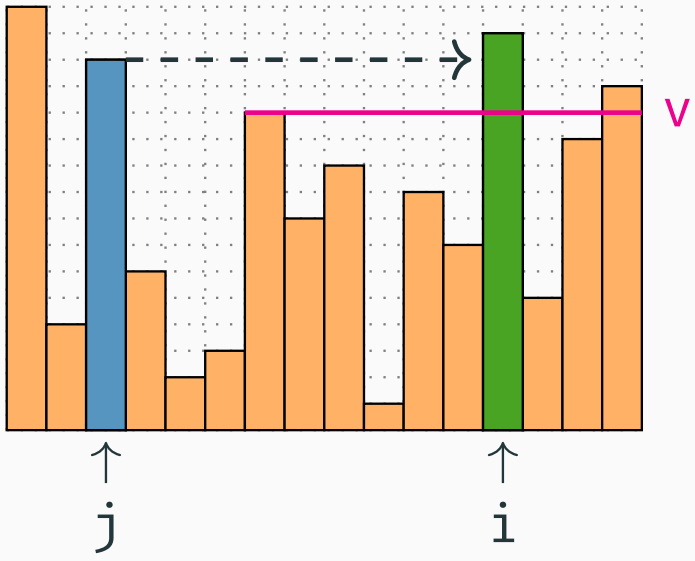
príklad o klaunoch a tortách, verzia 1

```
bool trafi(int j, int i) {
  int k;
  if (a[i] < a[j]) return false;
  for (k = j + 1; k < i; k++)
    if (a[k] > a[j]) return false;
  return true;
}
```



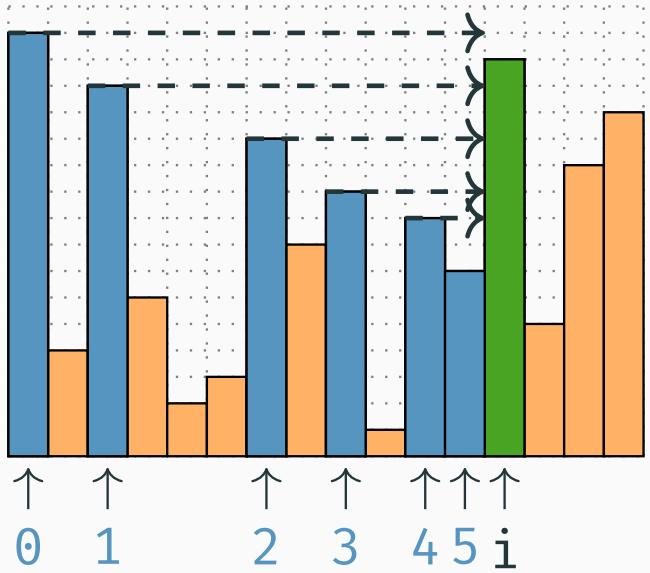

```
int klauni1() {
  int i, j, max = 0;
  for (i = 1; i < n; i++) {  // skusame kazdeho klauna
    int hits = 0;            // kolko zasahov dostal
    for (j = 0; j < i; j++)  // vsetci klauni nalavo od i
      if (trafi(j, i)) hits++;
    if (hits > max) max = hits;
  }
  return max;
}
```

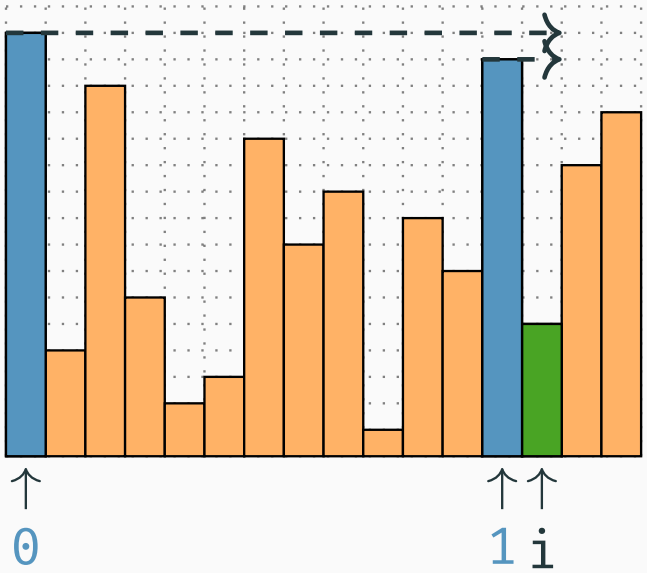
príklad o klaunoch a tortách, verzia 2




```
int klauni2() {
  int i, j, max = 0;
  for (i = 1; i < n; i++) {  // skusame kazdeho klauna
    int hits = 0,            // kolko zasahov dostal
        v = 0;               // najvyssi klaun medzi nimi
    for (j = i - 1; j >= 0 && a[j] < a[i]; j--)  // skusame vlavo
      if (a[j] > v) {                             // trafi ?
        hits++;                                   // zaratame zasah
        v = a[j];                                 // upravime vysku
      }
    if (hits > max) max = hits;  // pamatame si najzapatlanejsieho
  }
  return max;
}
```

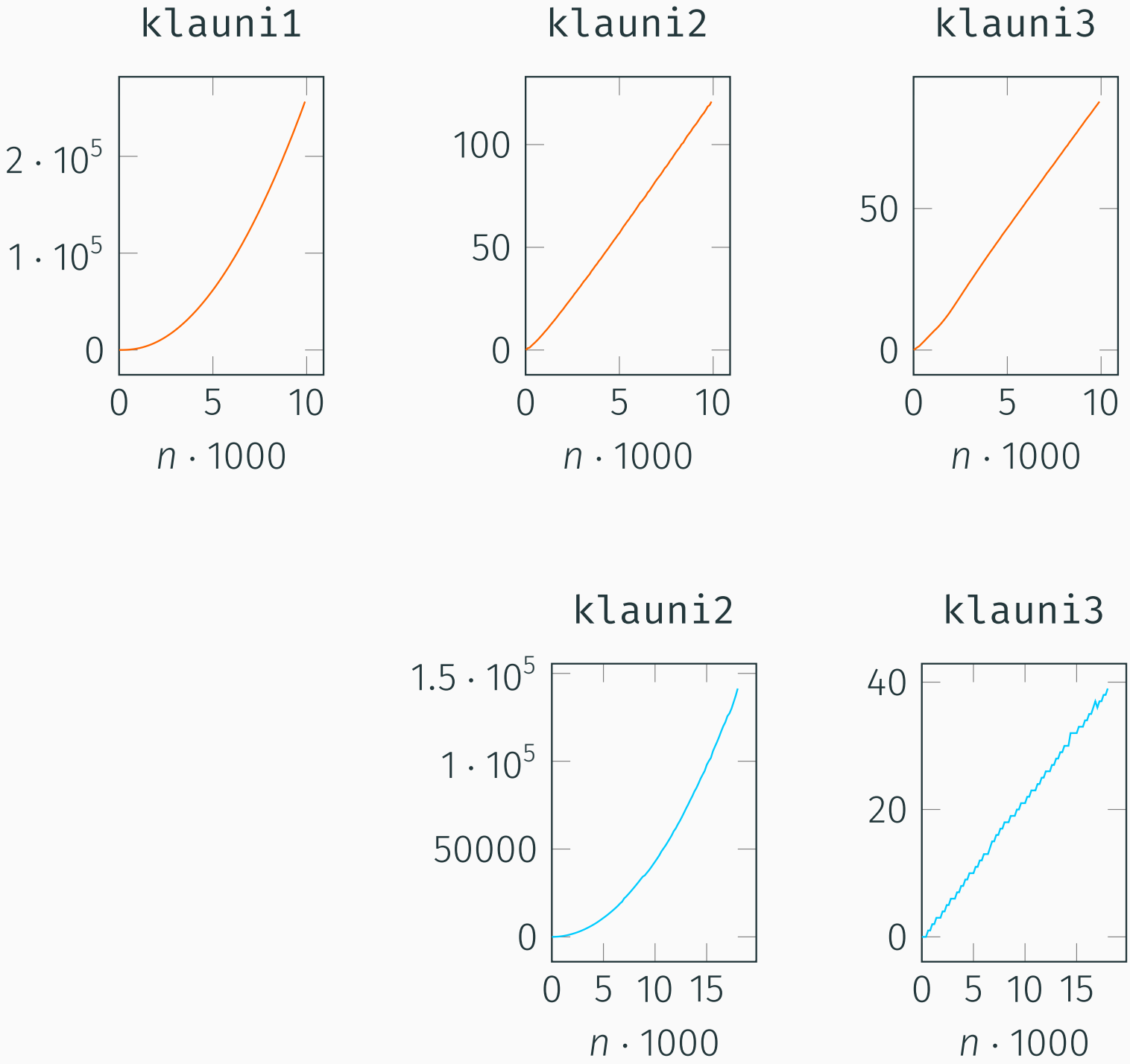
príklad o klaunoch a tortách, verzia 3






```
int klauni3() {
  int i, max = 0;
  int s[n], m = 0;              // zasobnik a jeho velkost
  for (i = 0; i < n; i++) {     // skusame kazdeho klauna
    int h = 0;                  // kolko zasahov dostal
    while (m > 0 && s[m - 1] < a[i]) {
      h++;  // zaratame zasah
      m--;  // vyhodime zo zasobnika
    }
    s[m] = a[i];  // pridame i-cka do zasobnika
    m++;
    if (h > max) max = h;  // aktualizujeme zasahy
  }
  return max;
}
```

porovnanie času



o rekurzívnych algoritmoch

Fibonacciho čísla: 0 1 1 2 3 5 8 13 21 34 . . .

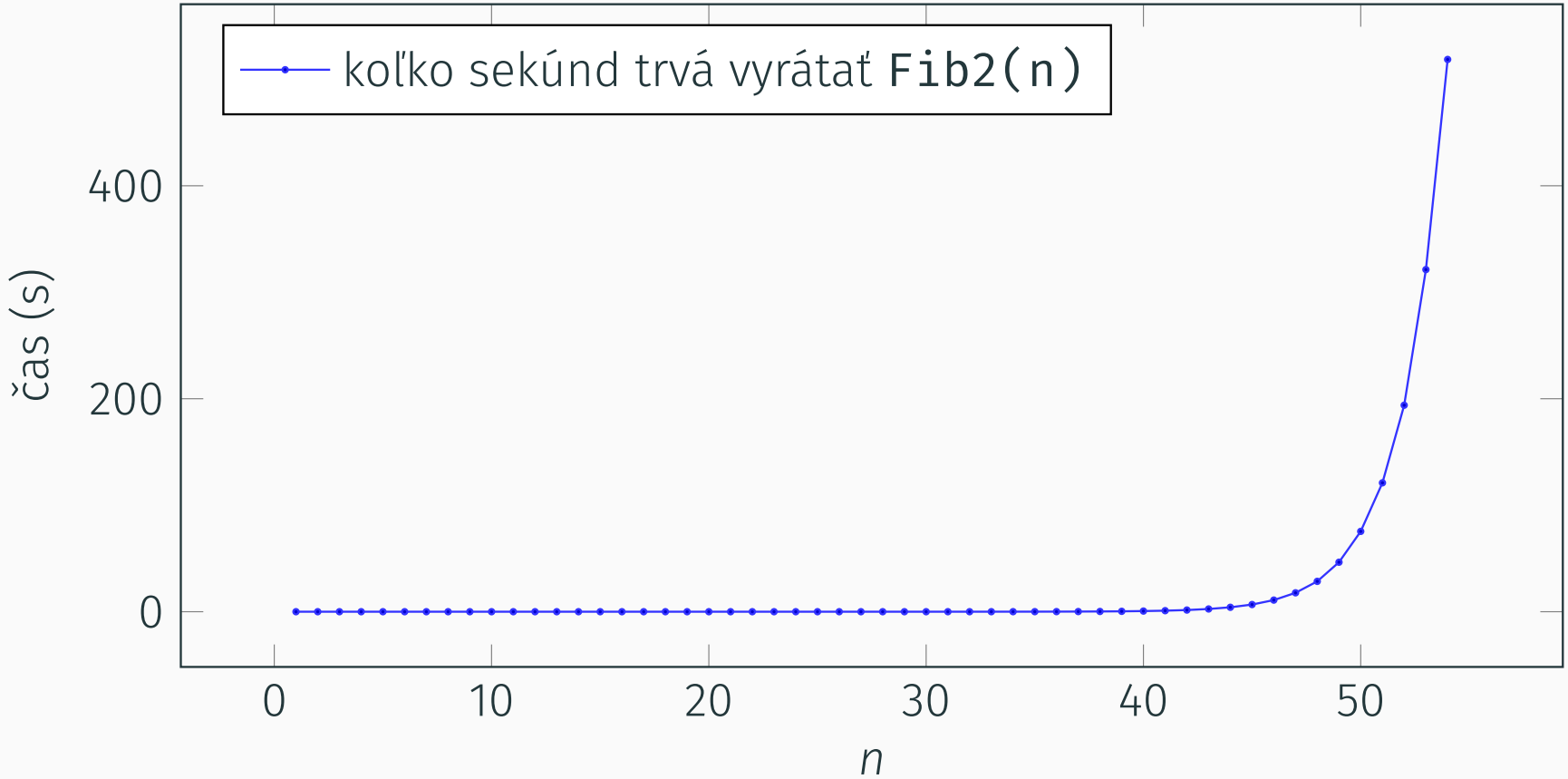
```
long int Fib1(int n) {
  if (n < 3) return n - 1;
  long int a = 0, b = 1, c;
  int i;
  for (i = 2; i < n; i++) {
    c = a + b; a = b; b = c;
  }
  return c;
}
```

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1)
              + Fib2(n - 2);
}
```

Fibonacciho čísla: 0 1 1 2 3 5 8 13 21 34 . . .

```
long int Fib1(int n) {
  if (n < 3) return n - 1;
  long int a = 0, b = 1, c;
  int i;
  for (i = 2; i < n; i++) {
    c = a + b; a = b; b = c;
  }
  return c;
}
```

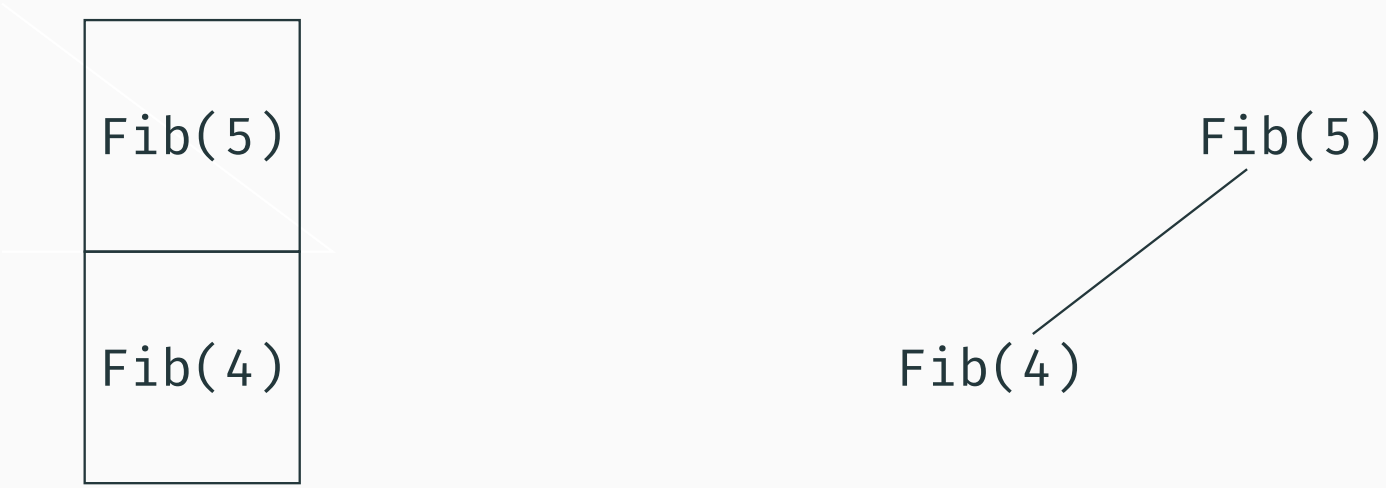
```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1)
              + Fib2(n - 2);
}
```



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

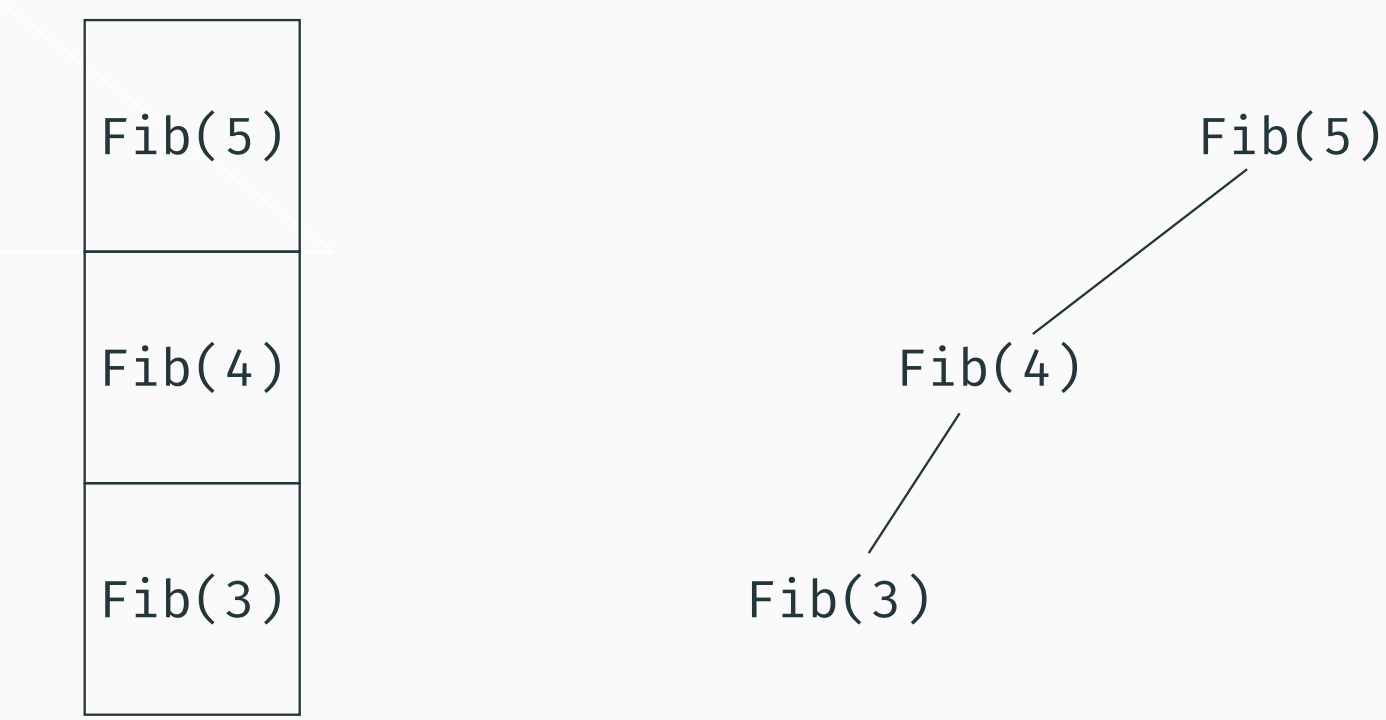
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

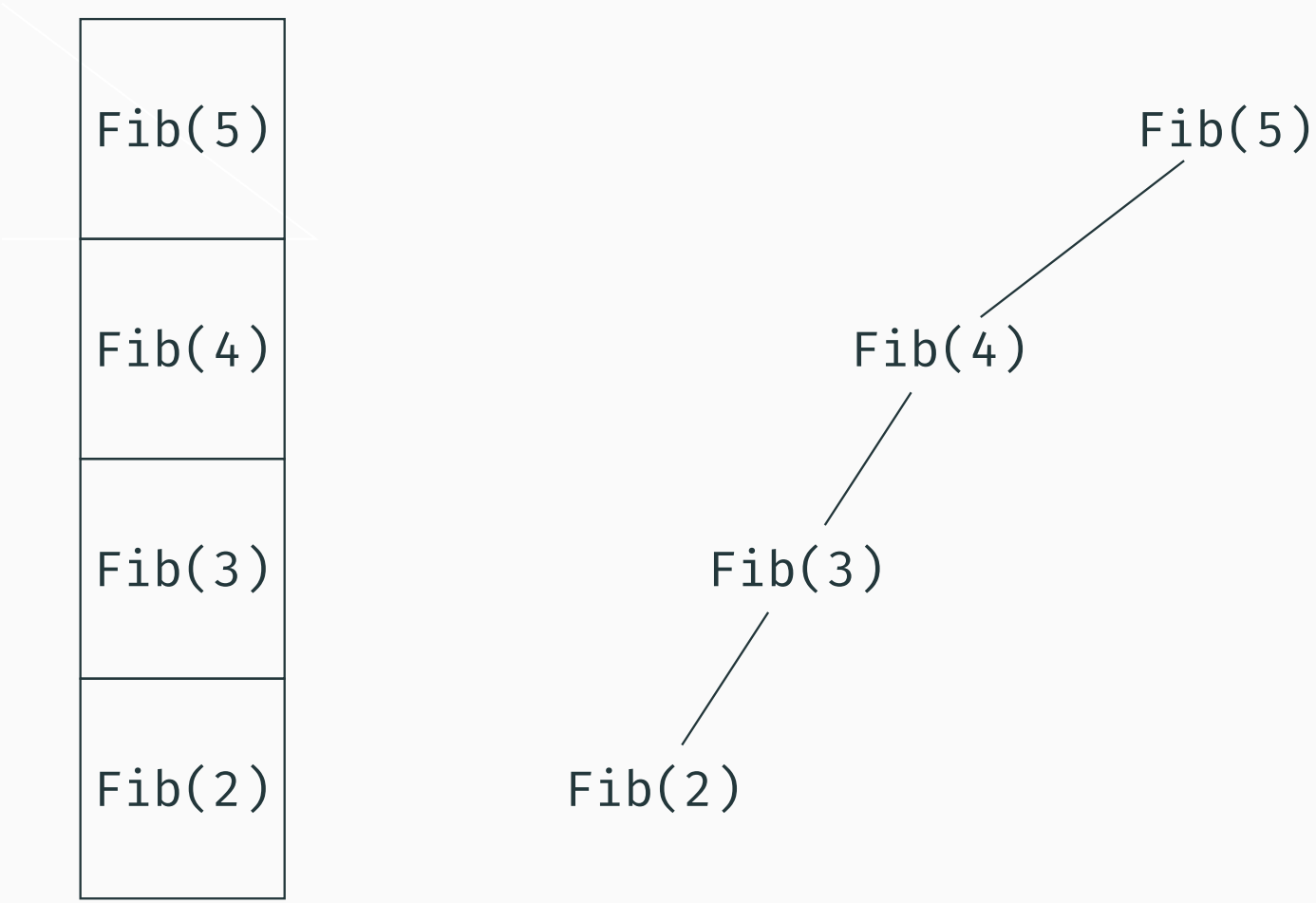
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

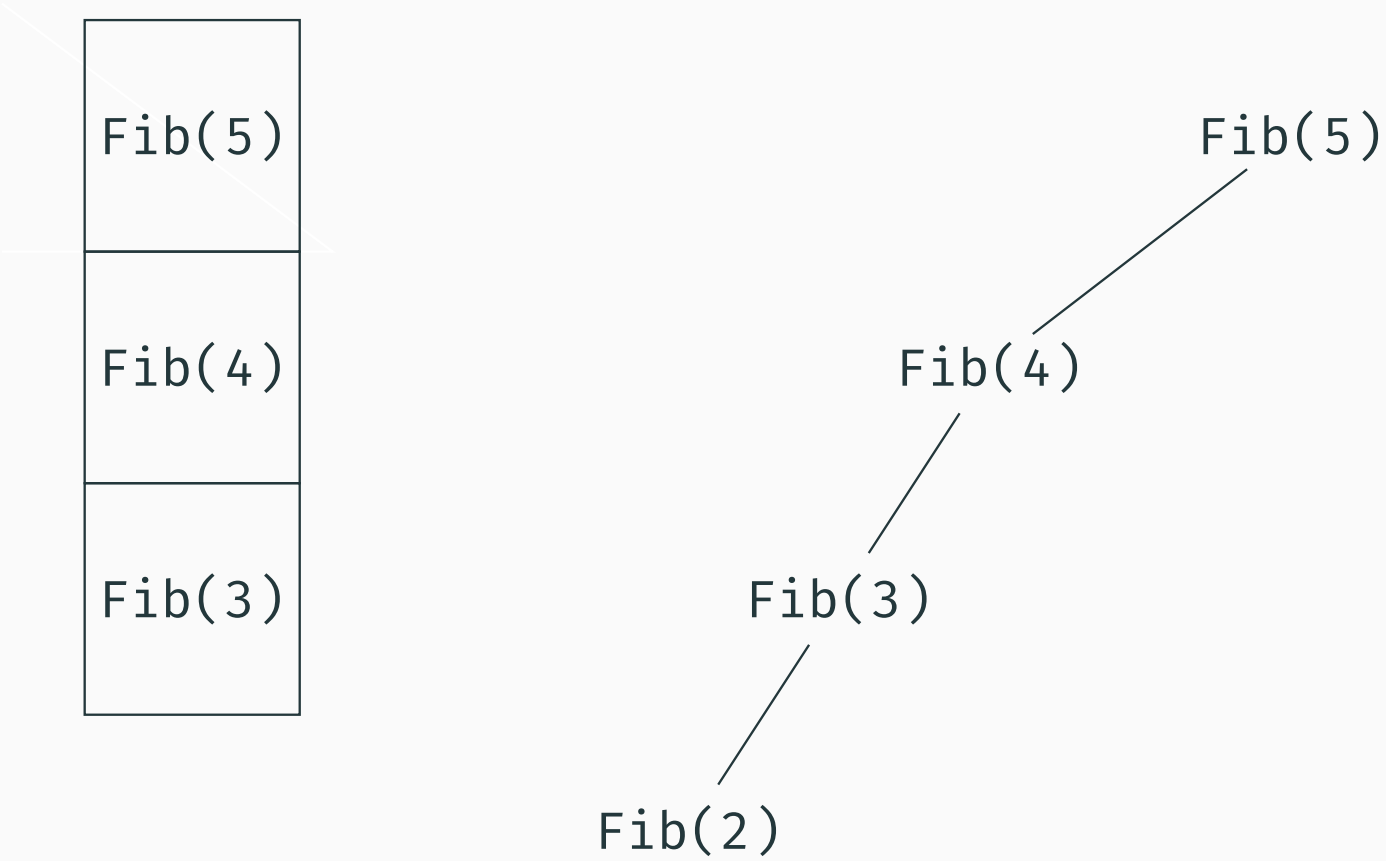
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

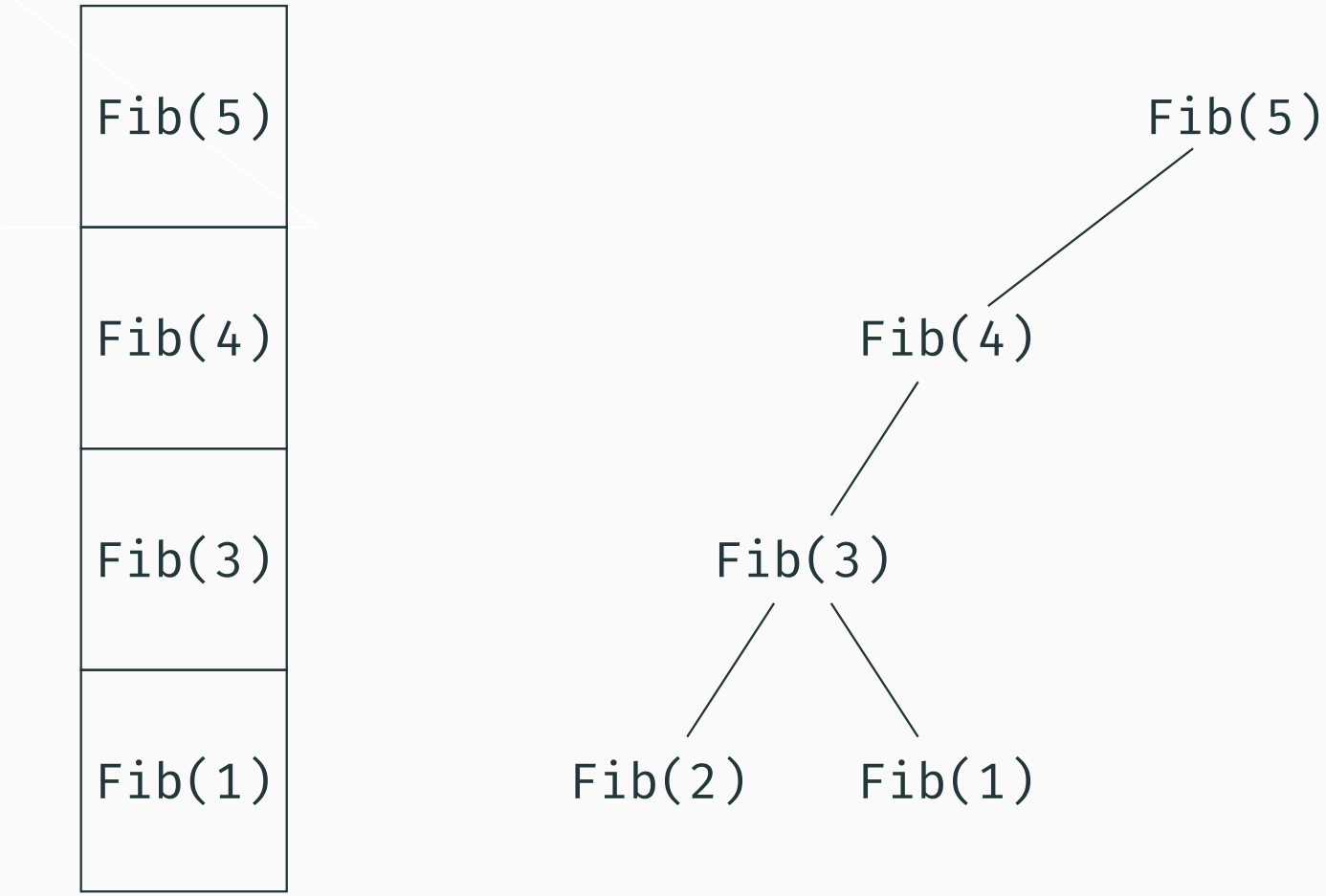
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

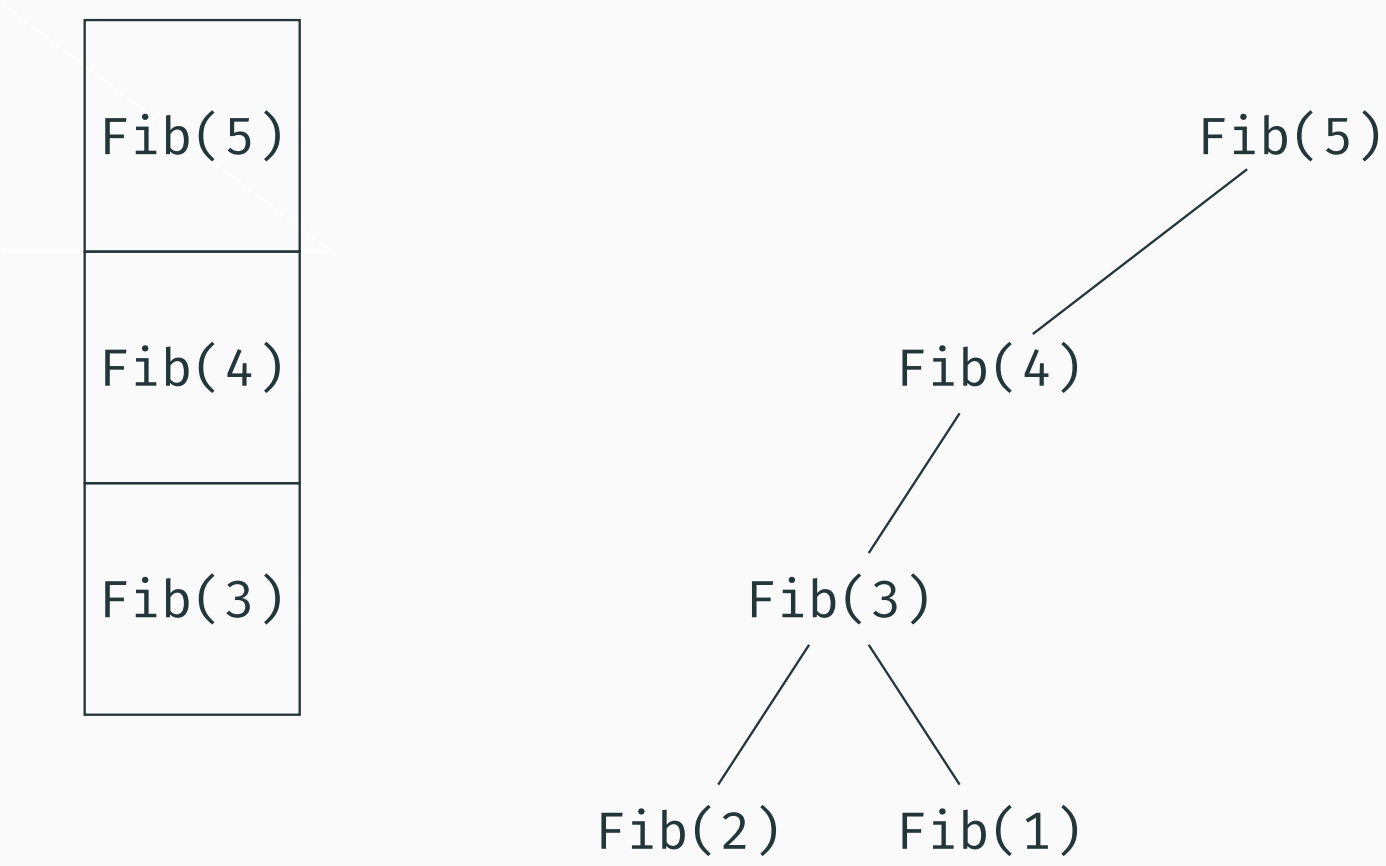
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

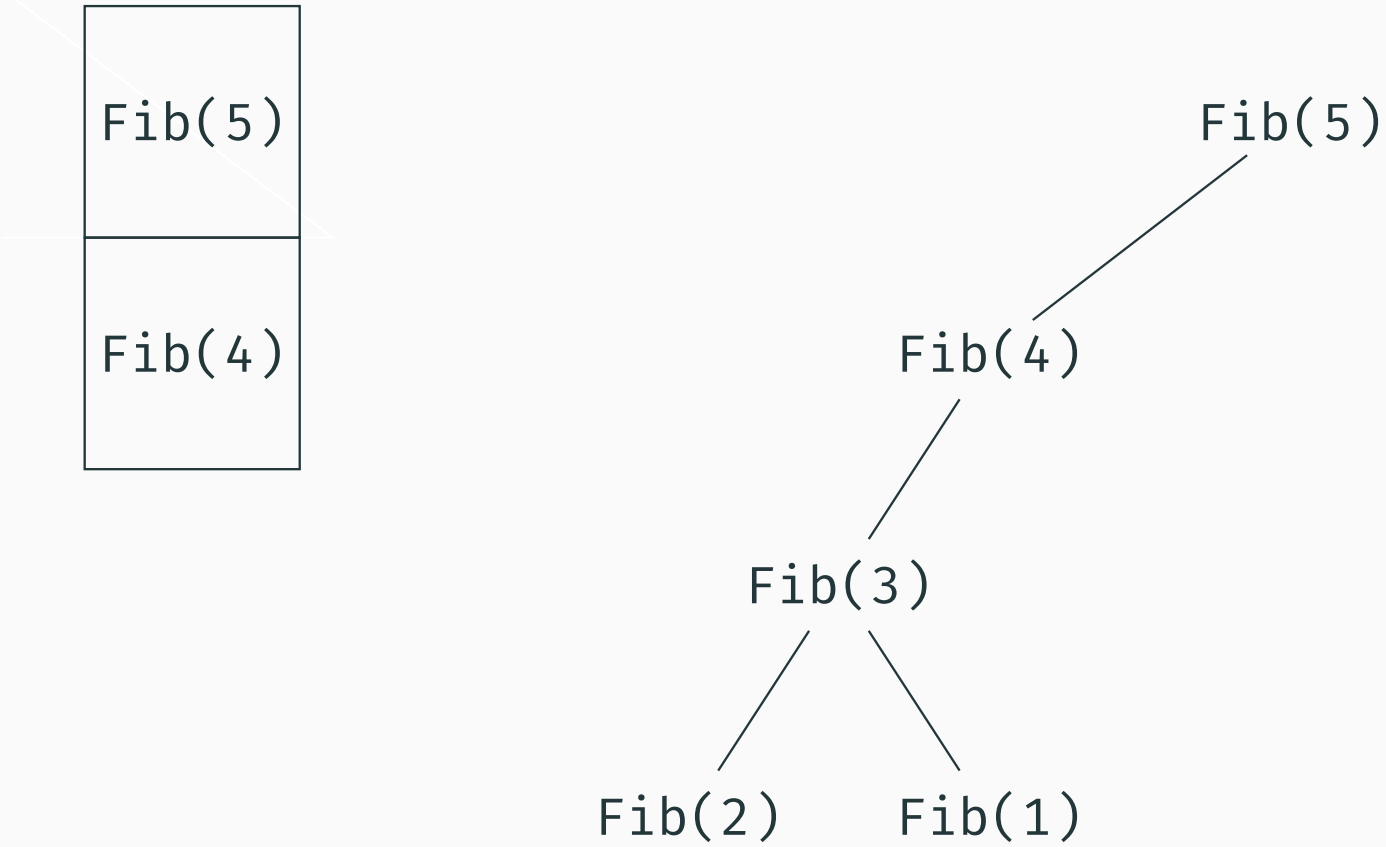
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

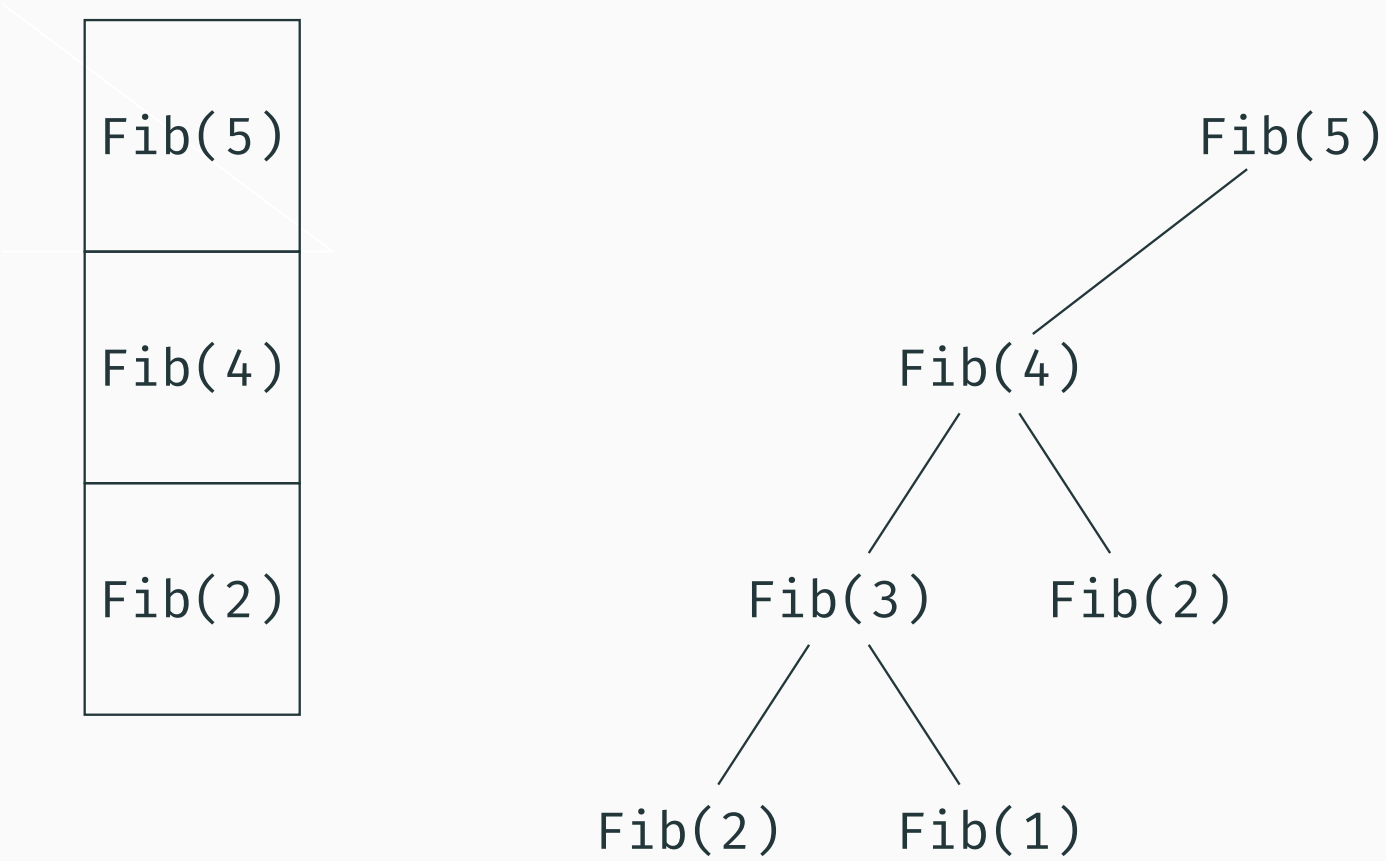
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

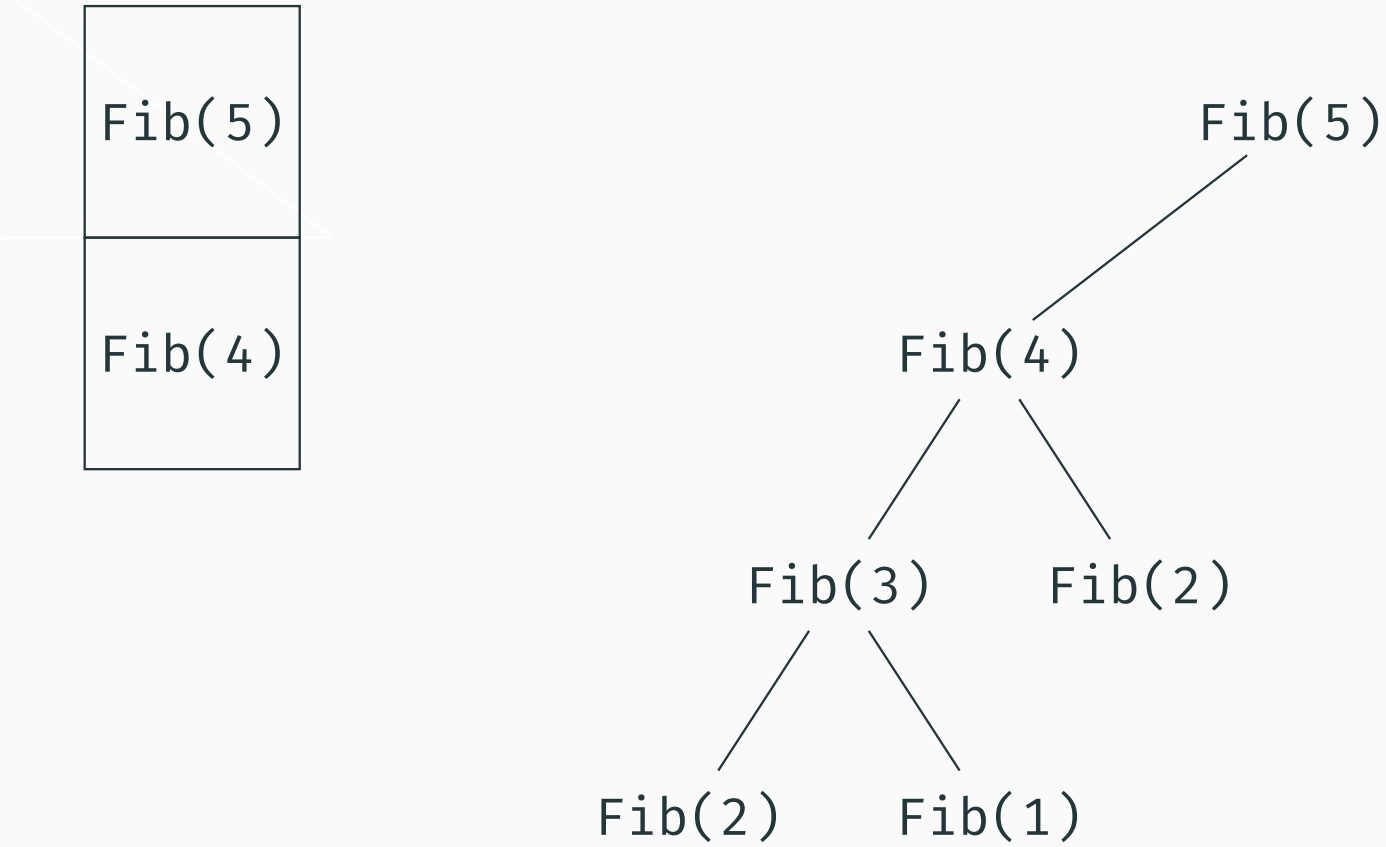
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

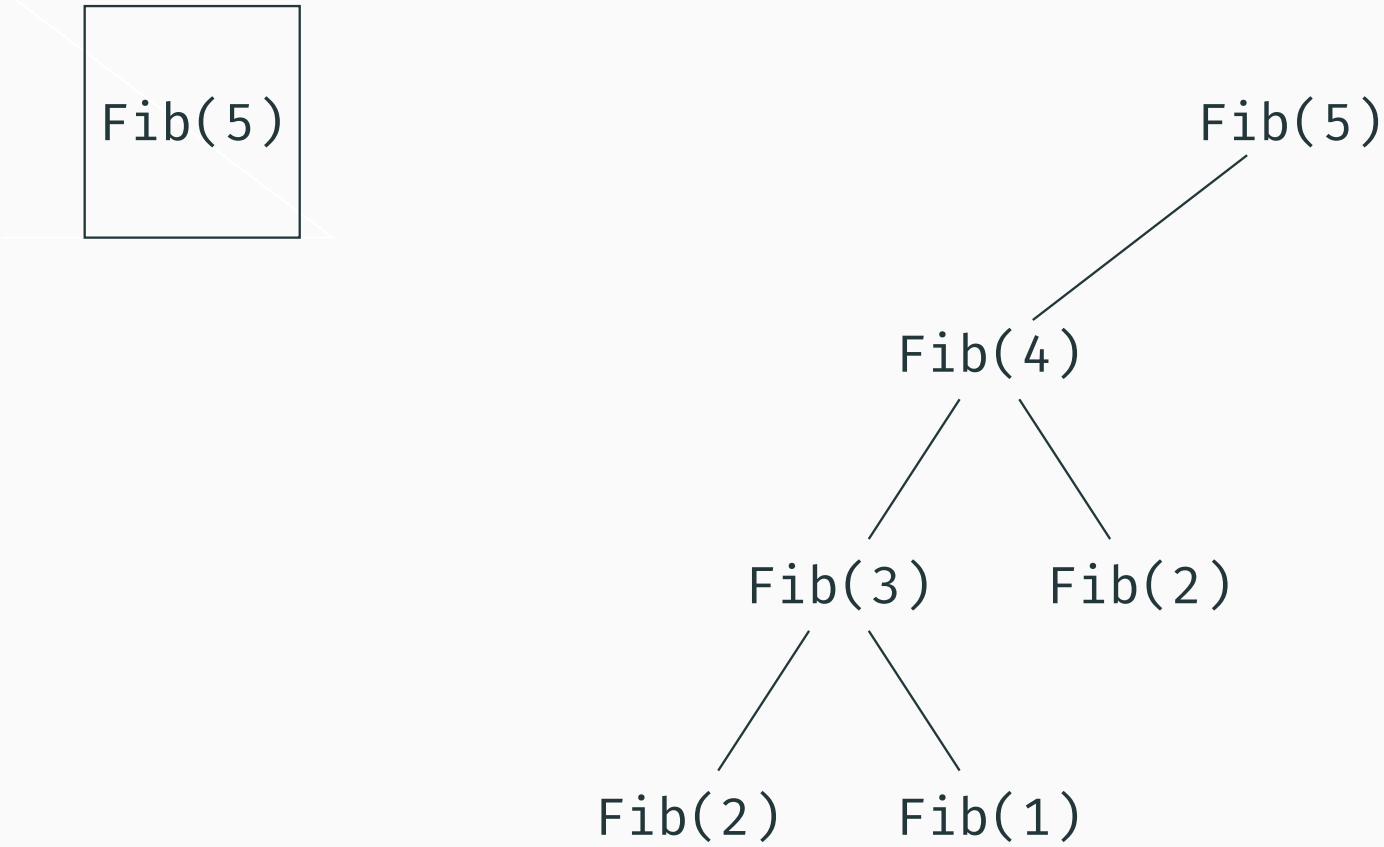
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

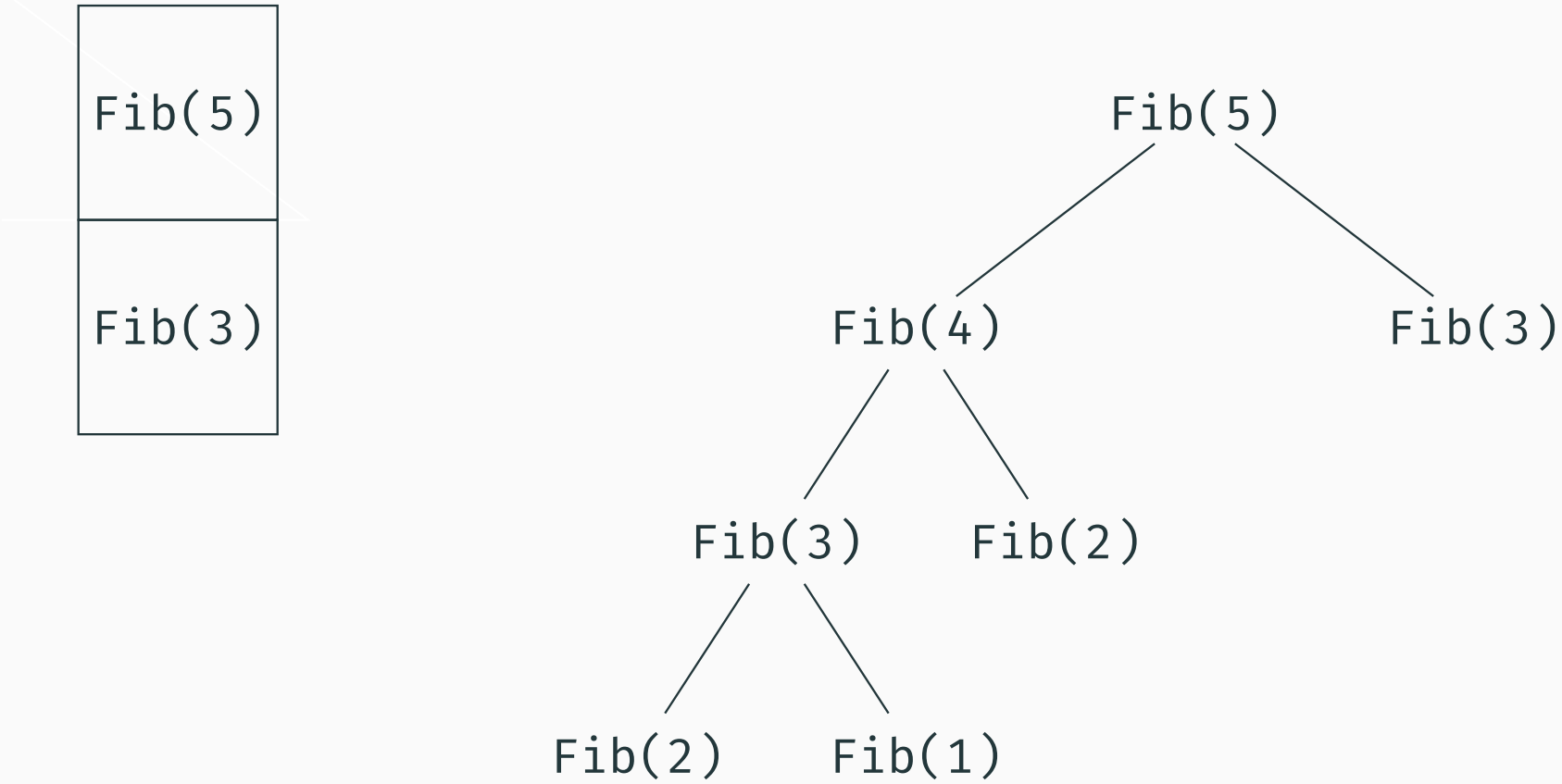
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

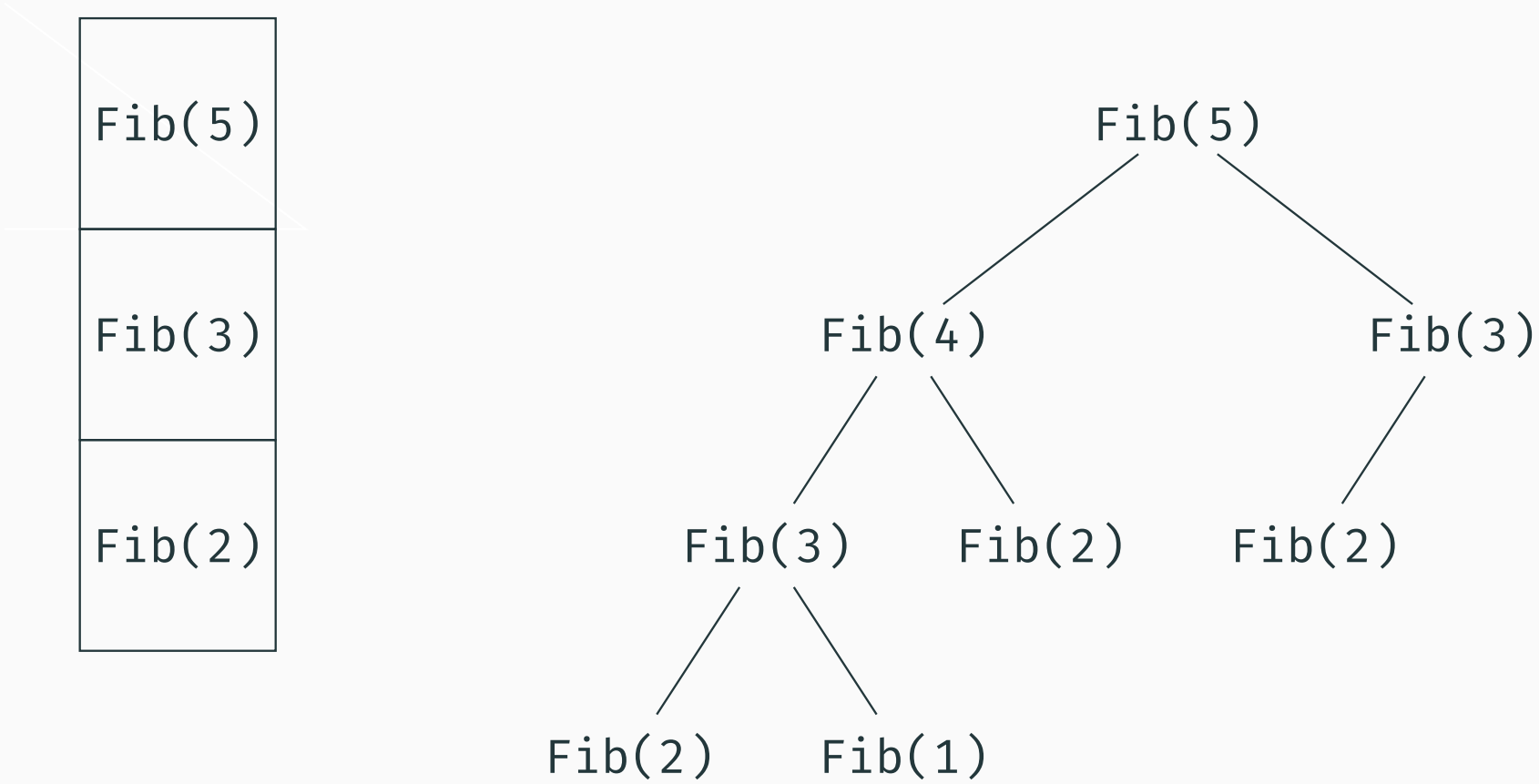
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

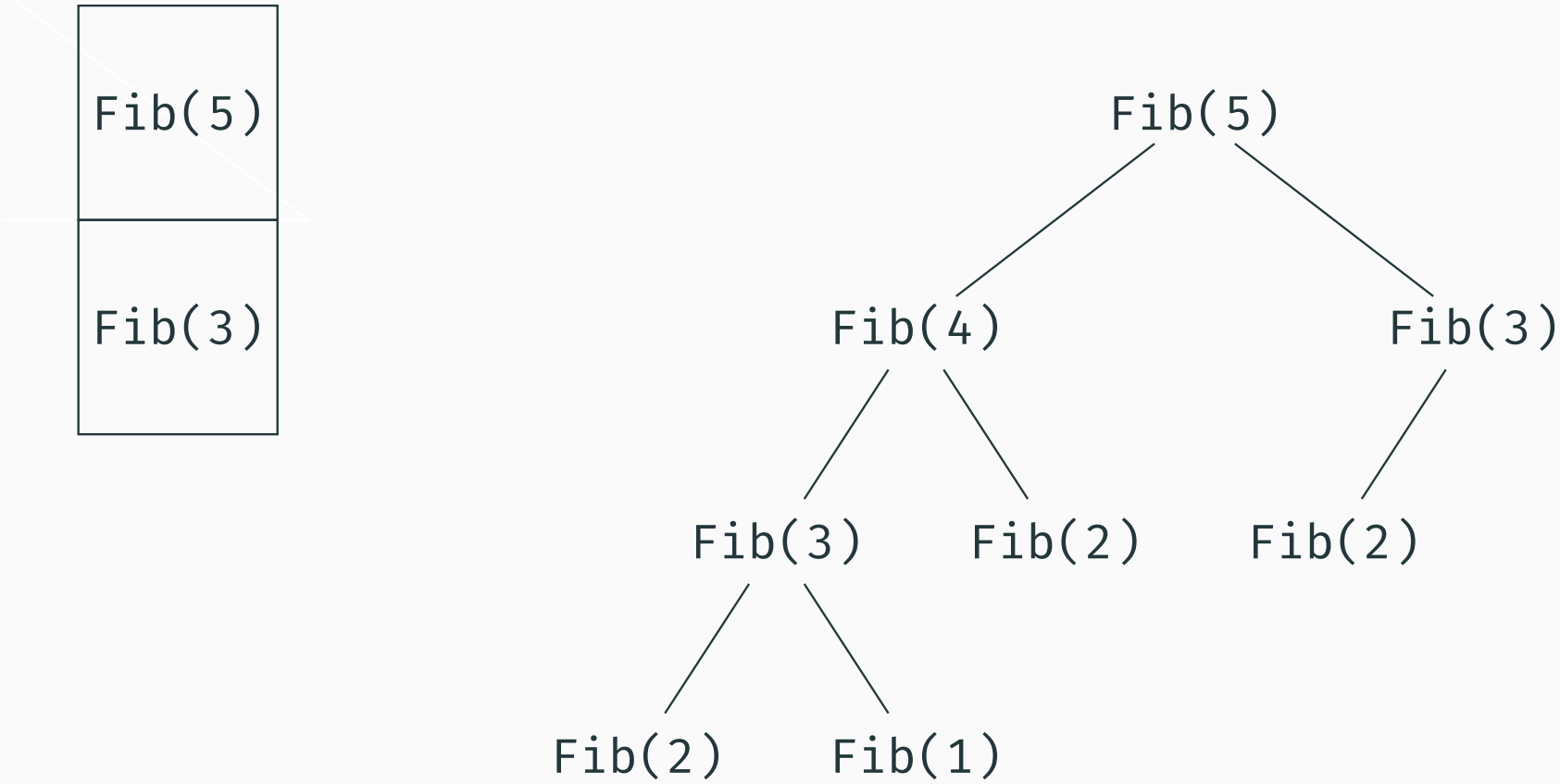
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

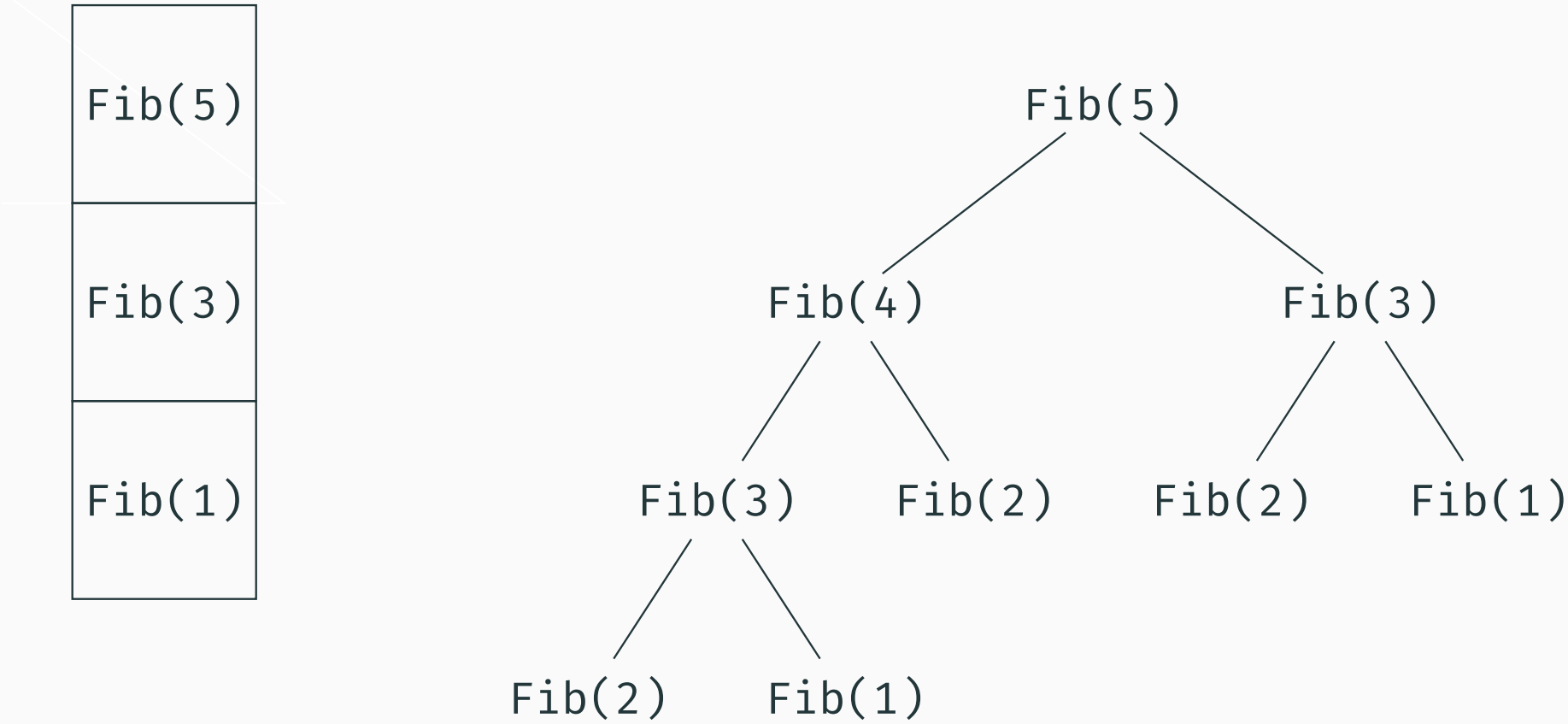
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

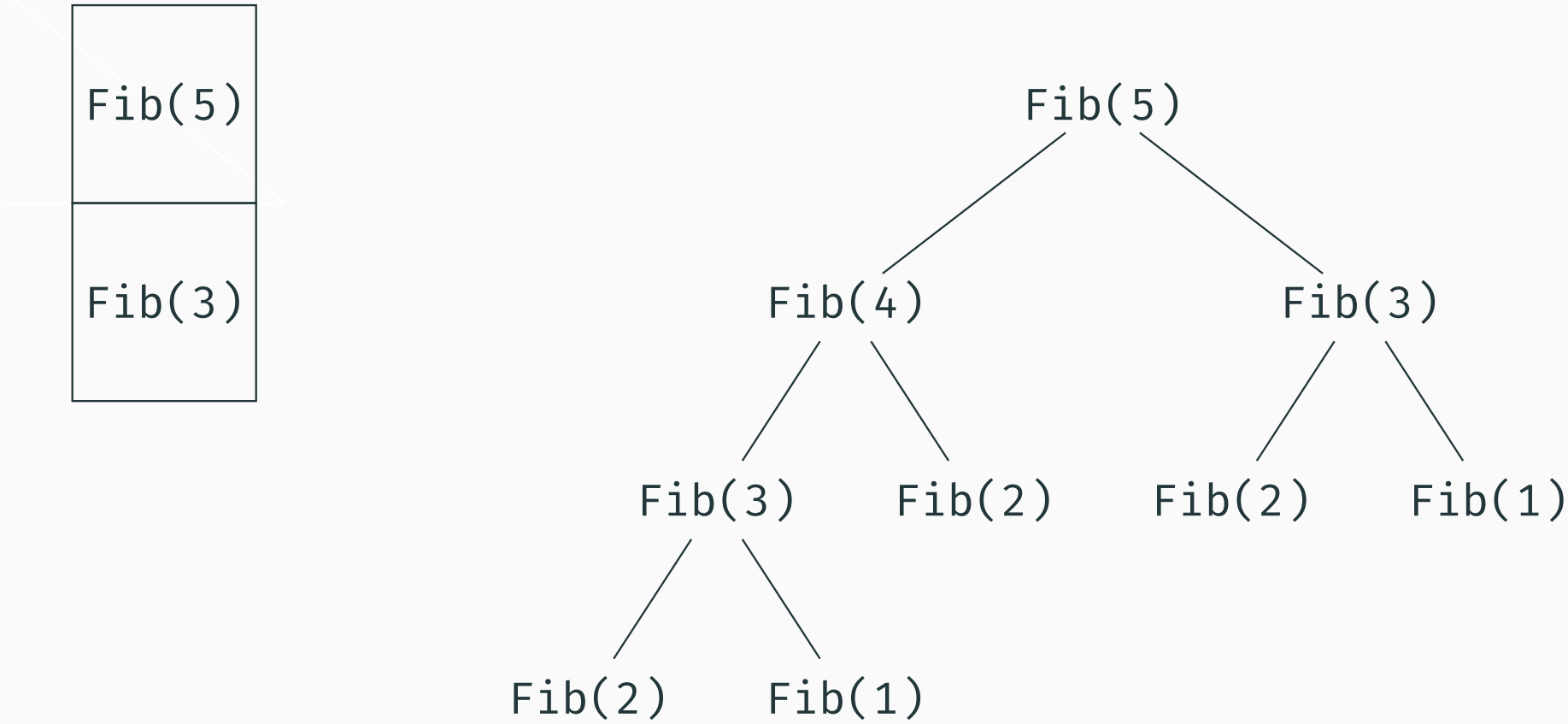
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

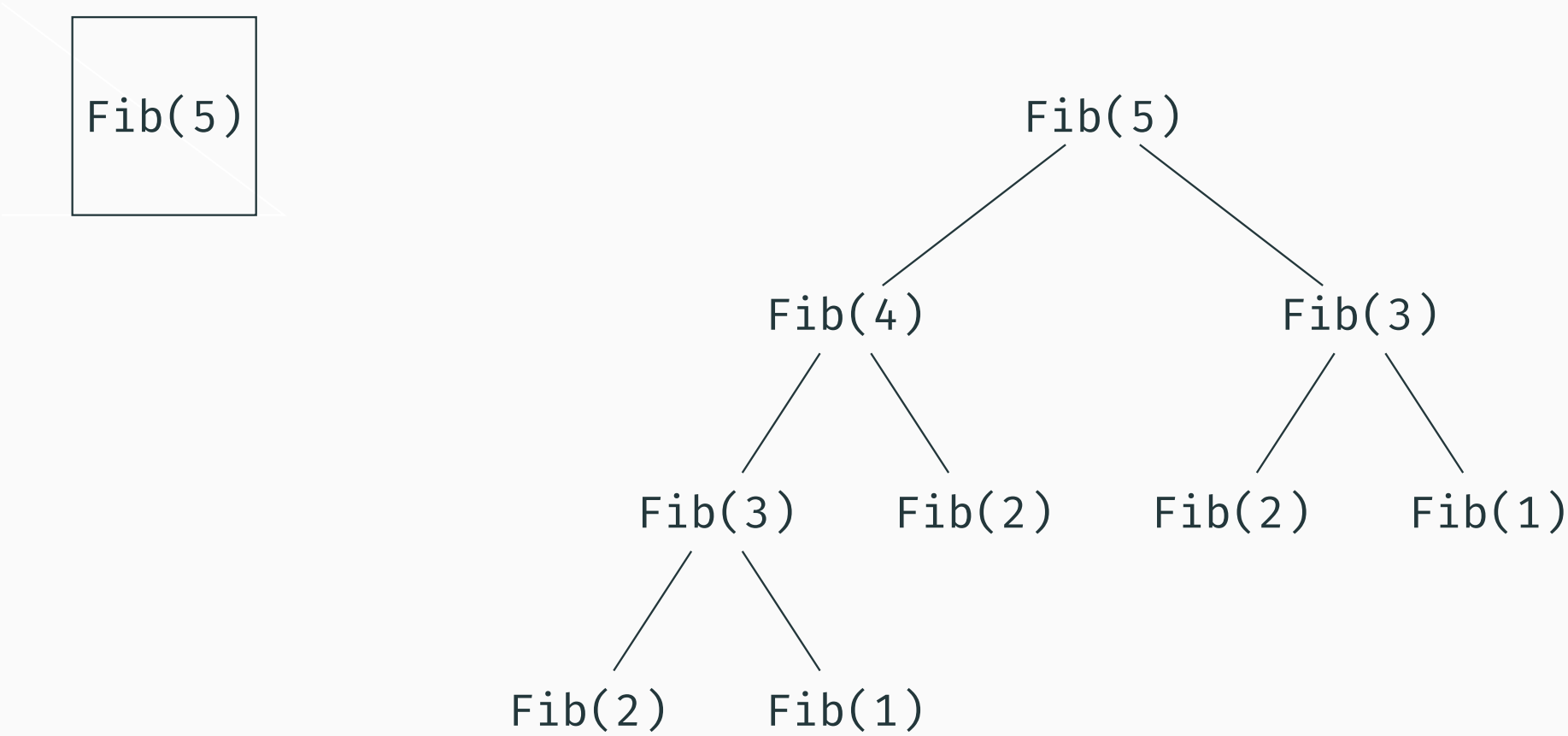
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

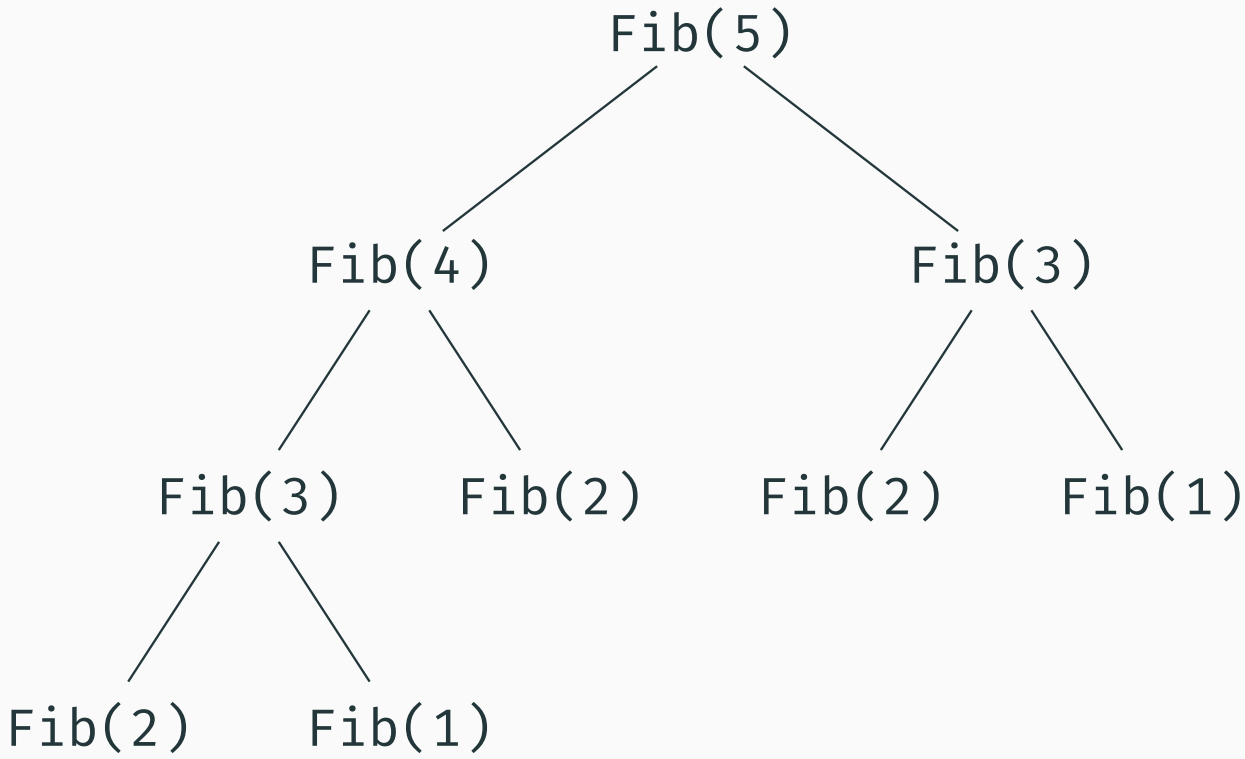
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

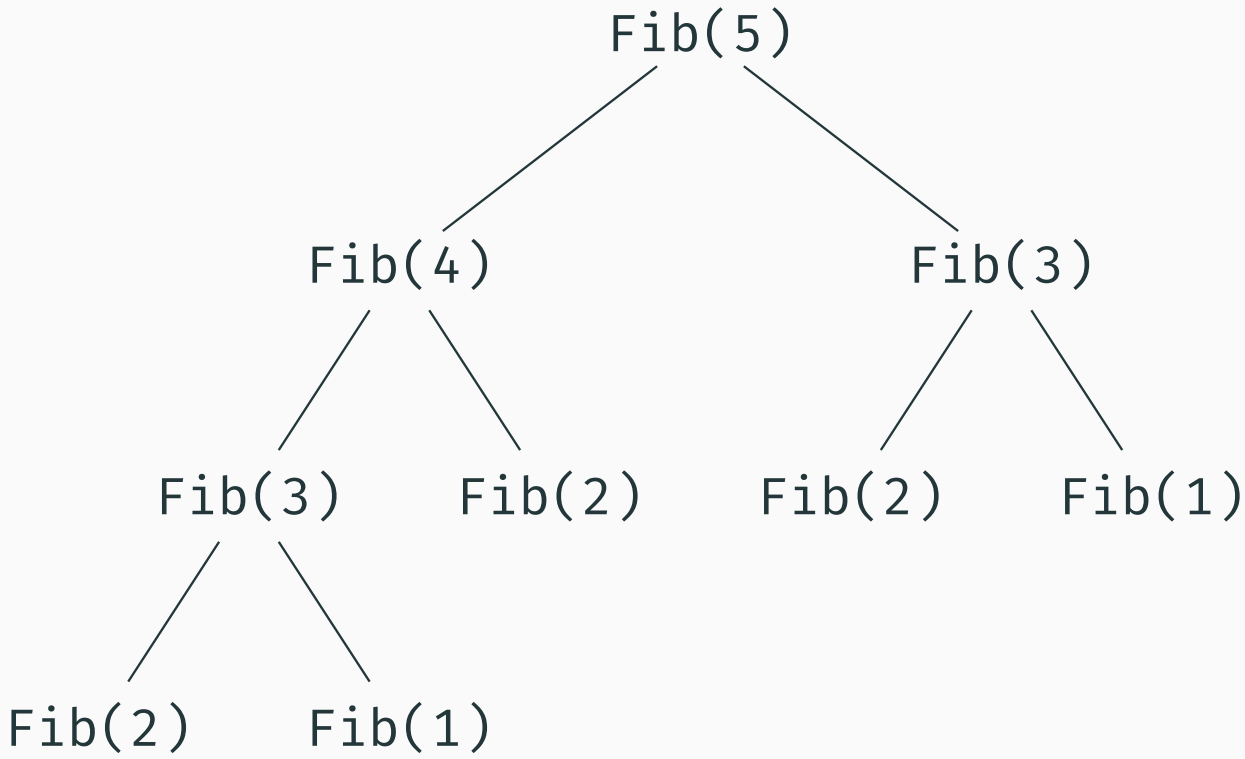
$$T(n) = T(n-1) + T(n-2) + c$$



zložitosť Fib2

```
long int Fib2(int n) {
  if (n < 3) return n - 1;
  return Fib2(n - 1) + Fib2(n - 2);
}
```

$$T(n) = T(n-1) + T(n-2) + c$$

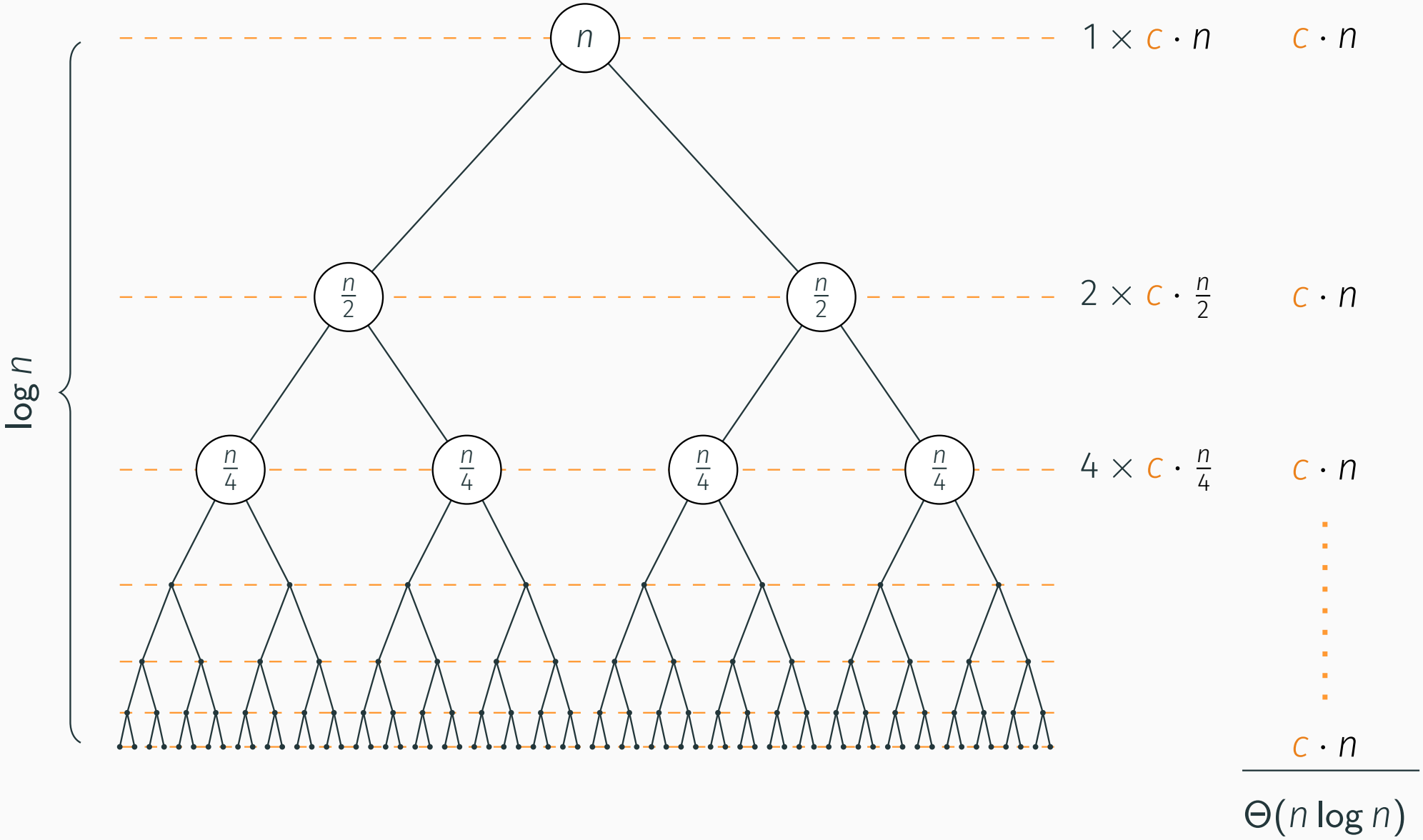


MergeSort

```
void mergesort(std::vector<int> &a) {

  if (a.size() < 2) return;
  int m = a.size() / 2;
  std::vector<int> b[2]{
    { a.begin()      ,  a.begin() + m },
    { a.begin() + m  ,  a.end()       }
  };

  for (int i : {0, 1}) mergesort(b[i]);

  // std::merge ( ... )
  int j[2] = {0, 0};
  for (int i = 0; i < a.size(); i++)
    if ( j[1] >= b[1].size() ||
         (j[0] <  b[0].size() && b[0][j[0]] < b[1][j[1]]) )
      a[i] = b[0][j[0]++];
    else
      a[i] = b[1][j[1]++];

}
```

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n$$

MergeSort – zložitosť

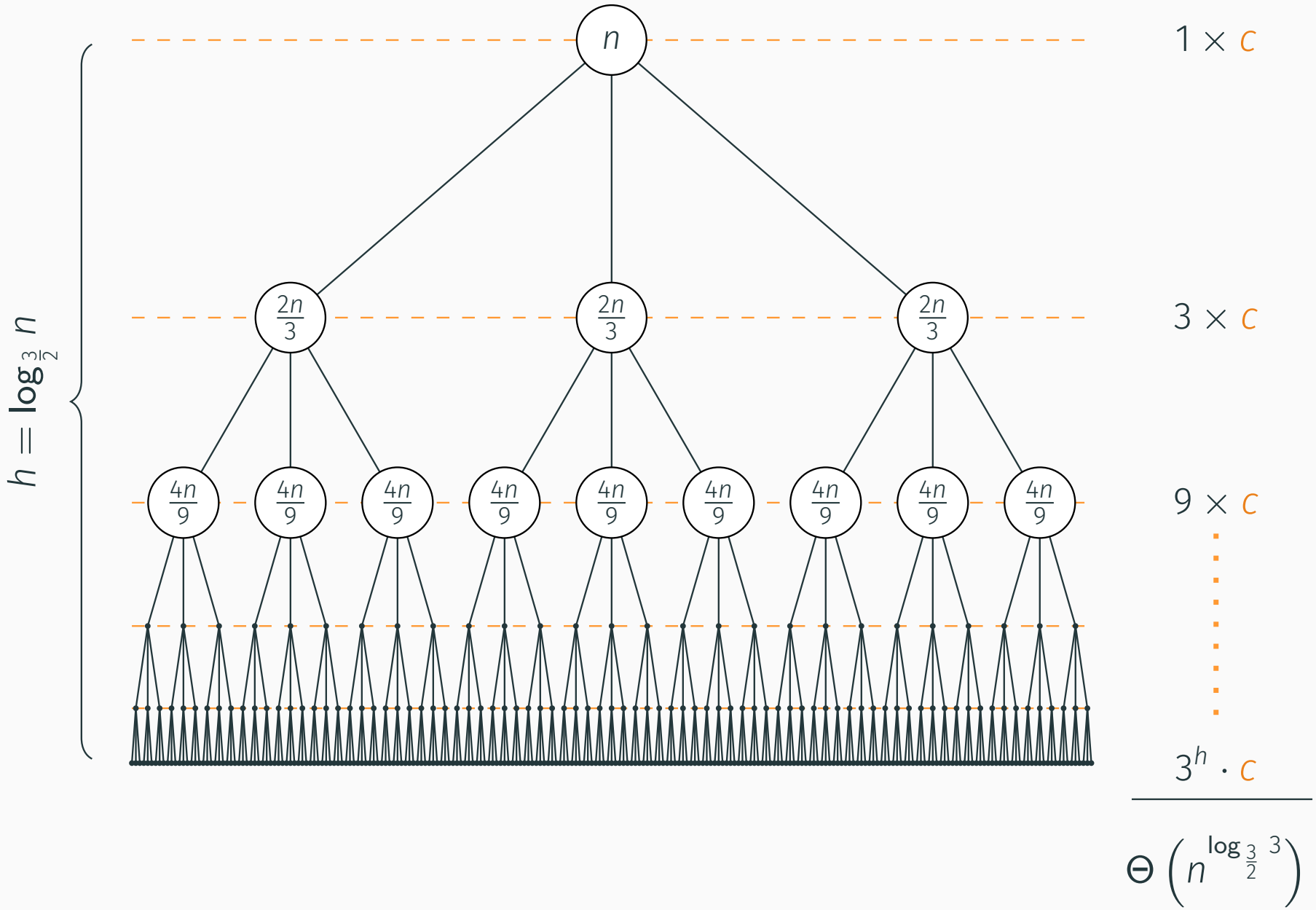


StoogeSort

```
using It = std::vector<int>::iterator;

void stoogesort(It a, It b) {
  int n = b - a, m = n / 3;
  b--;
  if (*a > *b) std::swap(*a, *b);
  if (m == 0) return;
  stoogesort(a, a + (n - m));
  stoogesort(a + m, a + n);
  stoogesort(a, a + (n - m));
}
```

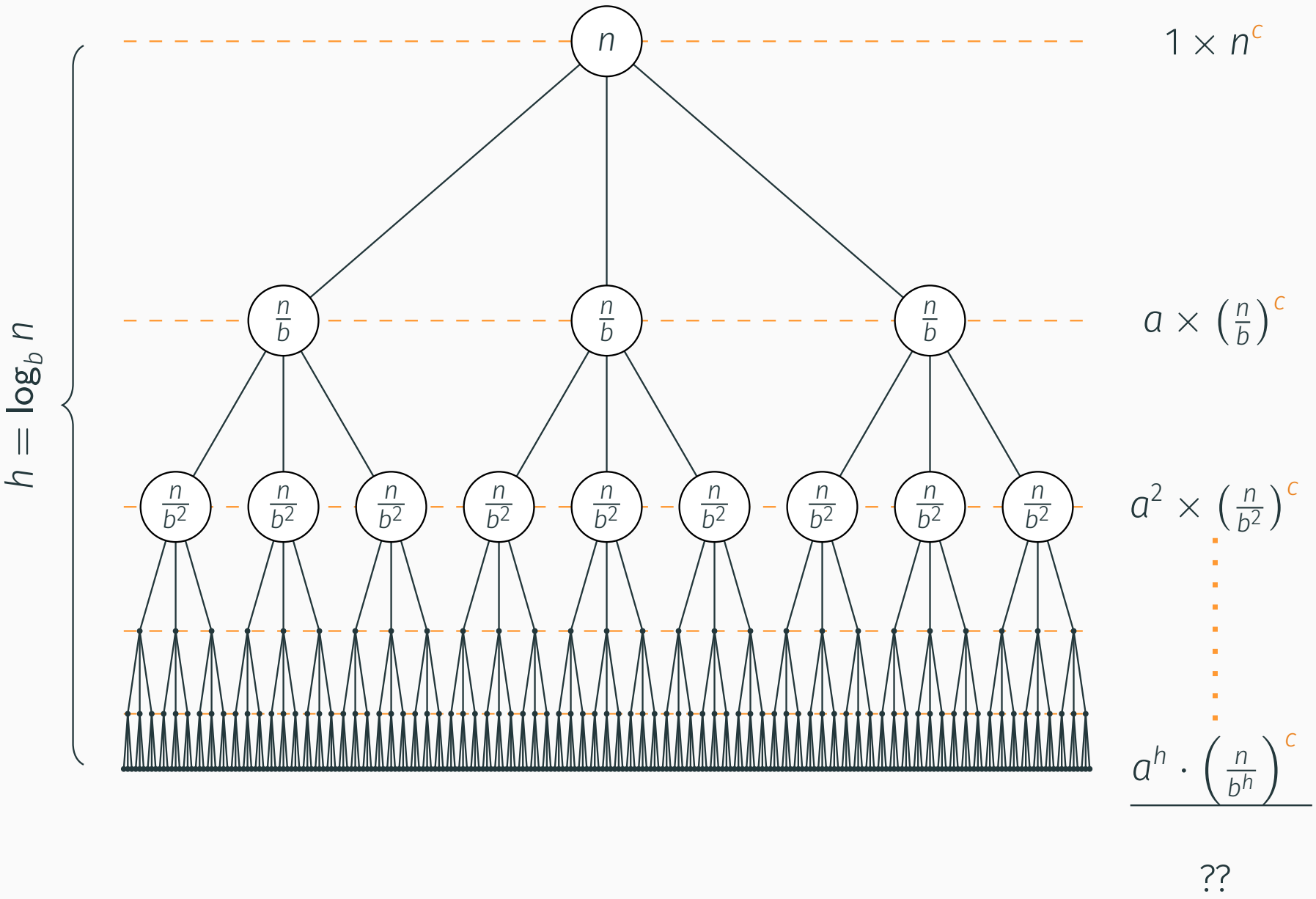
$$T(n) = 3 \cdot T\left(\frac{2n}{3}\right) + c$$

StoogeSort – zložitosť



$$\Theta\left(n^{\log_{\frac{3}{2}} 3}\right)$$

Master Theorem: $T(n) = a \cdot T\left(\frac{n}{b}\right) + n^c$



Master Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + n^c$$

Označme $q := \frac{a}{b^c}$.

1. ak $q < 1$, potom $T(n) = \Theta\left(n^c\right)$
2. ak $q = 1$, potom $T(n) = \Theta\left(n^c \cdot \log n\right)$
3. ak $q > 1$, potom $T(n) = \Theta\left(n^{\log_b a}\right)$

... a čo tak $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{7n}{10}\right) + n$?

... a prípadne

```
int f(int a, int b) {
  int i, c = 0;

  if (a == b) return 1;
  for (i = a; i < b; i++)
      c = c + f(a, i) + f(i + 1, b);
  return c / (b - a);
}
```