Rekurence a Quick Sort

Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Rekurzia pripomenutie

- Rekurzívna funkcia je funkcia, ktorá vo svojom tele obsahuje volanie samej seba – toto volanie sa nazýva rekurzívne volanie
- Cieľom rekurzívnej funkcie je v každom rekurzívnom volaní zjednodušiť problém, ktorý rieši.
- Problém zjednodušuje tak dlho, dokým sa nedostane na tak jednoduchú instanciu daného problému, ktorú sme už schopný vyriešiť triviálne.

Faktoriál rekurzívne - pripomenutie

```
Factorial(n)
    if n = 1            BASE CASE
        return 1
    else
        return n * Factorial(n – 1)        RECURSIVE CASE
```

Funkcia bude volať samú seba, dokým nedosiahne BASE CASE, potom začne vracať výsledky jednotlivých rekurzívnych volaní

Časová zložitosť rekurzívnych algoritmov

- Budeme riešiť pomocou **rekurencí** (rekurentných vzťahov)

Rekurence Faktoriálu

```
Factorial(n)
    if n = 1                          BASE CASE
        return 1
    else
        return n * Factorial(n – 1)         RECURSIVE CASE
```

REKURENCE:

$T(1) = 1$

$T(n) = 1 + T(n-1)$ (cost of non-recursive work + cost of recursive work)

Rekurence Faktoriálu

- REKURENCE:

  $T(1) = 1$ $T(1) = \Theta(1)$

  $T(n) = 1 + T(n-1)$ $T(n) = \Theta(1) + T(n-1)$

- RIEŠENIE REKURENCE

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

  $T(n-1) = 1 + T(n-2)$ .. z toho $T(n) = 2*1 + T(n-2)$

  $T(n-2) = 1 + T(n-3)$ .. z toho $T(n) = 3*1 + T(n-3)$

  .

  .

  $T(2) = 1 + T(1)$

  $T(1) = 1$ Keď rozpíšeme T(n) dostaneme $T(n) = 1 + 1 + 1 + .. 1 + T(1)$

  Koľko musí prebehnúť rekurzivných zavolaní aby sme dosiahli T(1) (base case), keď budeme vstup zmenšovať vždy o 1?

Rekurence Faktoriálu

Koľko musí prebehnúť rekurzivných zavolaní aby sme dosiahli T(1) (base case), keď budeme vstup zmenšovať vždy o 1?

T(2) 2 až n = n - 1

$$T(n) = \underbrace{1 + 1 + 1 + .. 1}_{\mathbf{n - 1}} + T(1)$$

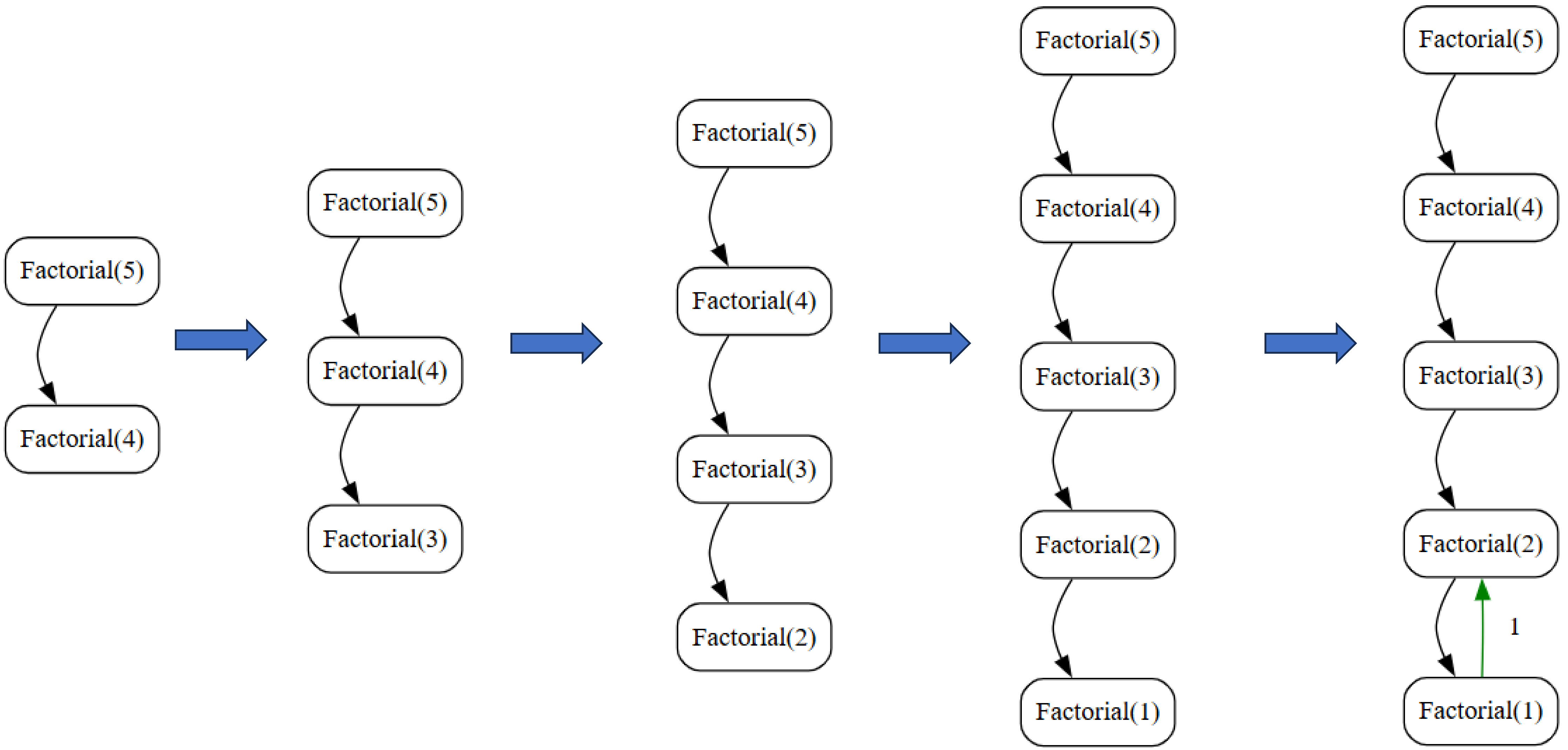
$$T(n) = (n - 1)*\Theta(1) + \Theta(1)$$

$$T(n) = \Theta(n - 1) + \Theta(1)$$

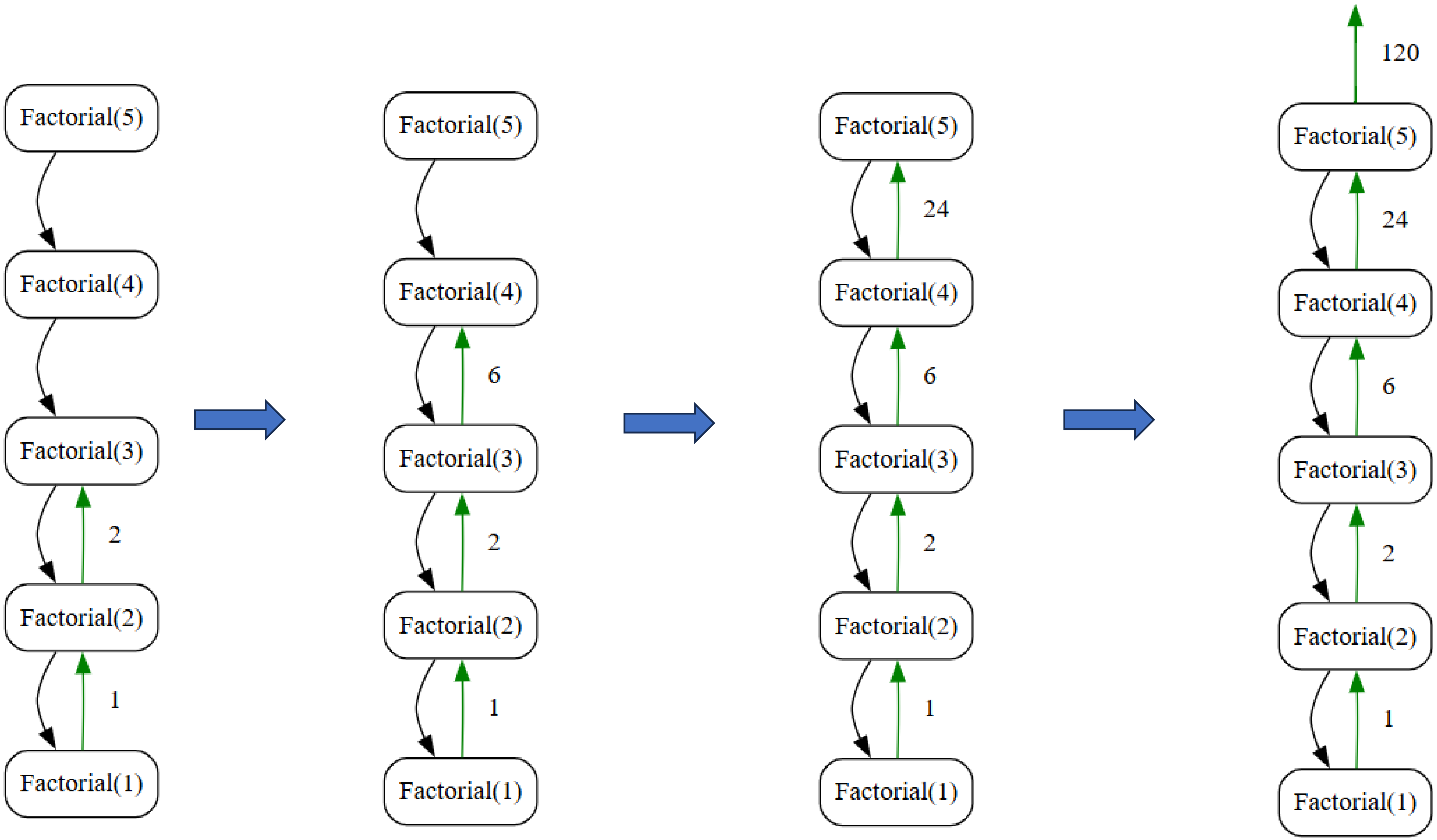
V celkovej zložitosti bude „dominovať“ výraz, ktorý má väčší asymptotický rast, v tomto prípade to bude $\Theta(n - 1)$ pretože lineárna funkcia rastie rýchlejšie ako konštantná ($\Theta(1)$)

Stačí potom dokázať že $n - 1 = \Theta(n)$ a dostaneme výsledok $\mathbf{T(n) = \Theta(n)}$

Výpočet 5! – zavoláme funkciu Factorial(5)



Výpočet 5! – zavoláme funkciu Factorial(5)



1. Príklad

```
Algo1(n)
    if n <= 0
        return 0
    else
        return 3 + Algo1(n – 1)
```

REKURENCE:

T(0) = 1 (konkrétnejšie T(n) = 1 pre n <= 0)

T(n) = 1 + T(n – 1) (cost of non-recursive work + cost of recursive work)

2. Príklad

```
Algo2(n)
    if n <= 0
        return 0
    else
        return 3 + Algo2(n – 2)
```

REKURENCE:

T(0) = 1

T(n) = 1 + T(n – 2) (cost of non-recursive work + cost of recursive work)

3. Príklad

```
Algo3(n)
    if n > 0
        return 3 * Algo3(n – 1)
    else
        return 1
```

REKURENCE:

$T(0) = 1$

$T(n) = 1 + T(n-1)$ (cost of non-recursive work + cost of recursive work)

4. Príklad

```
Algo4(n)
    if n > 0
        return 3 * Algo4(n – 2)
    else
        return 1
```

REKURENCE:

$T(0) = 1$

$T(n) = 1 + T(n - 2)$ (cost of non-recursive work + cost of recursive work)

5. Príklad

```
Algo5(n)
    if n > 0
        return 3 + Algo5(n / 3)
    else
        return 0
```

REKURENCE:

$T(0) = 1$

$T(n) = 1 + T(n / 3)$ (cost of non-recursive work + cost of recursive work)

Quick Sort

- Rekurzívny triediaci algoritmus

- Zvolíme prvok poľa ako PIVOT (v našom prípade to bude posledný prvok vstupného poľa)
- Rozdelíme pole na dve časti podľa pivota – naľavo budú prvky menšie ako pivot a napravo prvky väčšie ako pivot
- RECURSIVE CASE - funkcia zavolá samú seba na ľavú a pravú polovicu (2 rekurzívne volania)
- BASE CASE – keď ľavá alebo pravá polovica bude obsahovať iba 1 prvok, pretože jeden prvok je už zotriedený

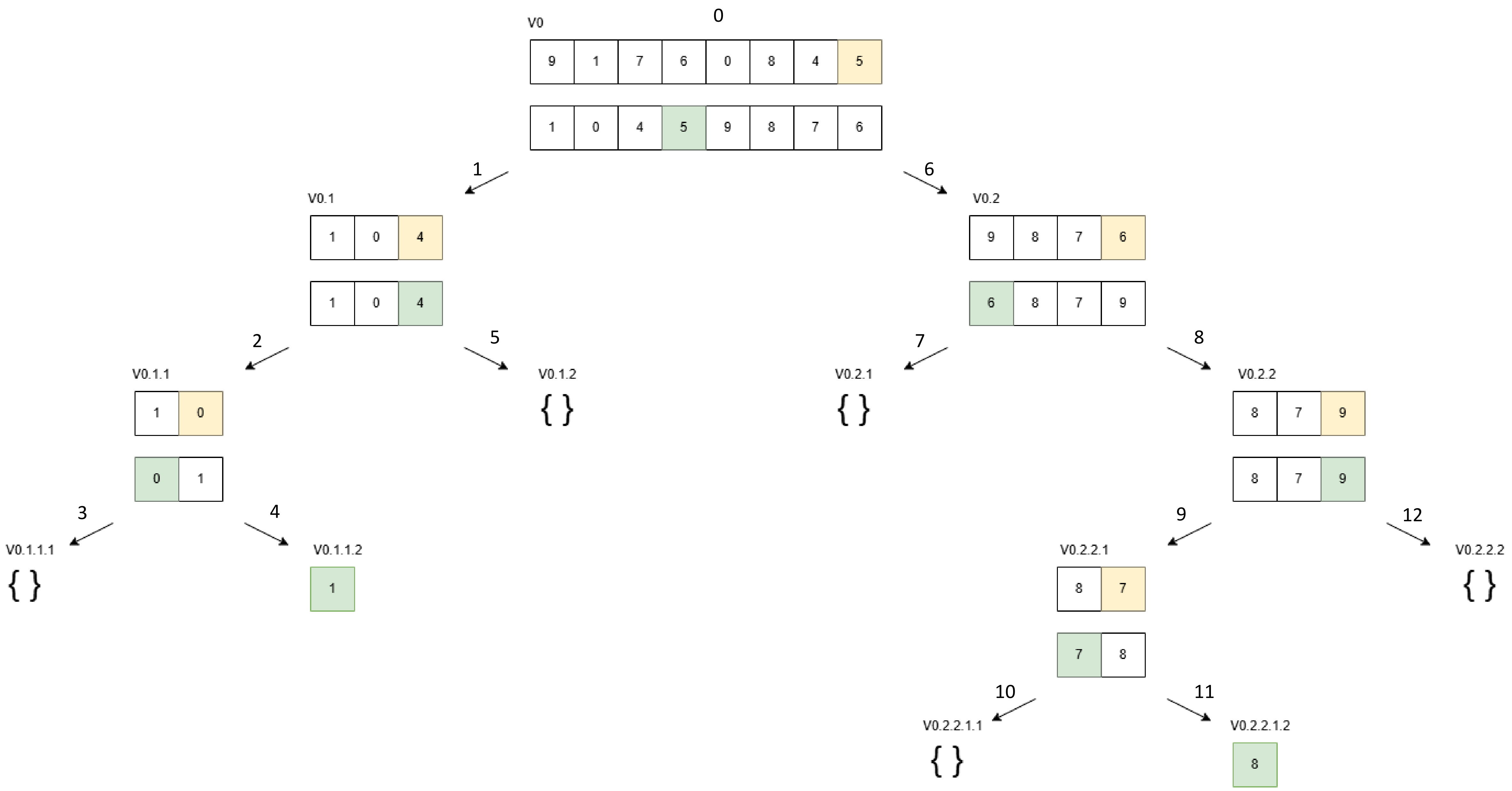
Quick Sort

```
Quick-Sort(A, p, r)
    if p < r
        q ← Partition(A, p, r)
        Quick-Sort(A, p, q – 1)
        Quick-Sort(A, q + 1, r)
```

- A – vstupné pole
- p, r – indexy
- q – index pivota

```
Partition(A, p, r)
    x ← A[r]
    i ← p - 1
    for j ← p to r - 1
        if A[j] <= x
            i ← i + 1
            swap(A[i], A[j])
    swap(A[i + 1], A[r])
    return i + 1
```

- x - pivot
- p, r, i, j – indexy


V0
0
9 1 7 6 0 8 4 5
1 0 4 5 9 8 7 6
1
6
V0.1
1 0 4
1 0 4
V0.2
9 8 7 6
6 8 7 9
2
5
7
8
V0.1.1
1 0
0 1
V0.1.2
{ }
V0.2.1
{ }
V0.2.2
8 7 9
8 7 9
3
4
9
12
V0.1.1.1
{ }
V0.1.1.2
1
V0.2.2.1
8 7
7 8
V0.2.2.2
{ }
10
11
V0.2.2.1.1
{ }
V0.2.2.1.2
8

```
VOLANIE 0
vstup: A = [9, 1, 7, 6, 0, 8, 4, 5], p = 0, r = 7
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 3, A = [1, 0, 4, 5, 9, 8, 7, 6], cele pole = [1, 0, 4, 5, 9, 8, 7, 6]
vykoname VOLANIE 0.1 so vstupom: A = [1, 0, 4], p = 0, r = 2

VOLANIE 0.1
vstup: A = [1, 0, 4], p = 0, r = 2
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 2, A = [1, 0, 4], cele pole = [1, 0, 4, 5, 9, 8, 7, 6]
vykoname VOLANIE 0.1.1 so vstupom: A = [1, 0], p = 0, r = 1

VOLANIE 0.1.1
vstup: A = [1, 0], p = 0, r = 1
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 0, A = [0, 1], cele pole = [0, 1, 4, 5, 9, 8, 7, 6]
vykoname VOLANIE 0.1.1.1 so vstupom: A = [], p = 0, r = -1

VOLANIE 0.1.1.1
vstup: A = [], p = 0, r = -1
podmienka p < r neplati, volanie skoncilo

vykoname VOLANIE 0.1.1.2 so vstupom: A = [1], p = 1, r = 1
VOLANIE 0.1.1.2
vstup: A = [1], p = 1, r = 1
podmienka p < r neplati, volanie skoncilo

vykoname VOLANIE 0.1.2 so vstupom: A = [], p = 3, r = 2
VOLANIE 0.1.2
vstup: A = [], p = 3, r = 2
podmienka p < r neplati, volanie skoncilo
```

```
vykoname VOLANIE 0.2 so vstupom: A = [9, 8, 7, 6], p = 4, r = 7
VOLANIE 0.2
vstup: A = [9, 8, 7, 6], p = 4, r = 7
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 4, A = [6, 8, 7, 9], cele pole = [0, 1, 4, 5, 6, 8, 7, 9]
vykoname VOLANIE 0.2.1 so vstupom: A = [], p = 4, r = 3

VOLANIE 0.2.1
vstup: A = [], p = 4, r = 3
podmienka p < r neplati, volanie skoncilo

vykoname VOLANIE 0.2.2 so vstupom: A = [8, 7, 9], p = 5, r = 7
VOLANIE 0.2.2
vstup: A = [8, 7, 9], p = 5, r = 7
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 7, A = [8, 7, 9], cele pole = [0, 1, 4, 5, 6, 8, 7, 9]
vykoname VOLANIE 0.2.2.1 so vstupom: A = [8, 7], p = 5, r = 6

VOLANIE 0.2.2.1
vstup: A = [8, 7], p = 5, r = 6
podmienka p < r plati
vykoname Partition..
vysledok po vykonani Partition: q = 5, A = [7, 8], cele pole = [0, 1, 4, 5, 6, 7, 8, 9]
vykoname VOLANIE 0.2.2.1.1 so vstupom: A = [], p = 5, r = 4

VOLANIE 0.2.2.1.1
vstup: A = [], p = 5, r = 4
podmienka p < r neplati, volanie skoncilo

vykoname VOLANIE 0.2.2.1.2 so vstupom: A = [8], p = 6, r = 6
VOLANIE 0.2.2.1.2
vstup: A = [8], p = 6, r = 6
podmienka p < r neplati, volanie skoncilo

vykoname VOLANIE 0.2.2.2 so vstupom: A = [], p = 8, r = 7
VOLANIE 0.2.2.2
vstup: A = [], p = 8, r = 7
podmienka p < r neplati, volanie skoncilo
```