Rekurze a Quick Sort

Bc. Katarína Olejková

KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Rekurze

- Programovací proces, kde funkcia volá samú seba za účelom rozdeliť problém na menšie prípady toho istého problému.
- Vyriešením menších prípadov dostaneme potom celkové riešenie problému

- Funkcia, ktorá volá samú seba sa nazýva – REKURZÍVNA funkcia
- Rekurzívna funkcia rozdeľuje problém na menšie prípady tak, že vo svojom tele volá samú seba s menším vstupom

- Volanie, kedy funkcia zavolá samú seba – REKURZÍVNE volanie

Vlastnosti rekurzívnej funkcie

BASE CASE

- Rekurzívna funkcia vo svojom tele volá samú seba – PROBLÉM volaná funkcia bude zase volať samú seba atď. tento proces môže pokračovať donekonečna
- Preto musíme definovať BASE CASE, ktorý bude slúžiť ako ukončovacia podmienka rekurzie (už neprebehne ďalšie rekurzívne volanie)
- BASE CASE – základný, najjednoduchší prípad, pre ktorý je riešenie známe alebo triviálne

Vlastnosti rekurzívnej funkcie

RECURSIVE CASE

- Časť rekurzívnej funkcie, kde funkcia volá samú seba s menším vstupom
- Rozdeľuje problém na menšie prípady toho istého problému
- Každým rekurzívnym volaním by sme sa mali priblížiť bližšie k BASE CASE

Príklad - Faktoriál

Výpočet: $n! = 1 * 2 \ldots * n$ pre $n > 0$

$5! = 1 * 2 * 3 * 4 * 5$

$5! = 120$

Faktoriál iteratívne

```
Factorial(n)
1.      fact ← 1
2.      for i ← 1 to n
3.          fact ← fact * i
4.      return fact
```

Príklad – Faktoriál rekurzívne

Výpočet: $n! = 1 * 2 \ldots * n$ pre $n > 0$

$5! = 1 * 2 * 3 * 4 * 5$

$5! = 120$

Môžeme uvažovať:

$5! = 5 * 4!$

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1!$

$1! = 1$

$n! = n * (n - 1)!$

$(n - 1)! = (n - 1) * (n - 2)!$

...

$1! = 1$

Príklad – Faktoriál rekurzívne

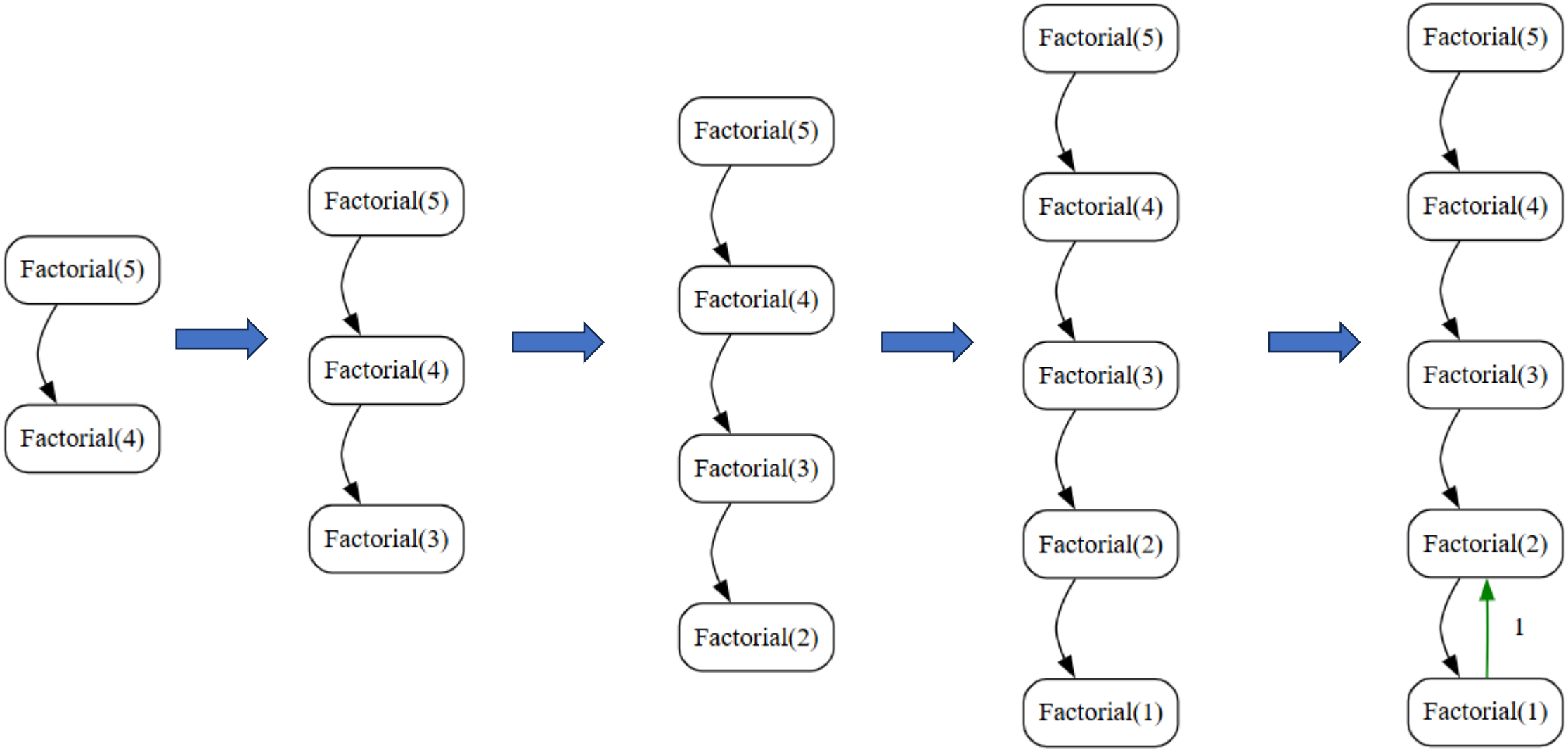
- 1! = 1 — triviálny výpočet – BASE CASE
- n! = n * (n – 1)! — Definovali sme problém výpočtu faktoriálu ako menší prípad toho istého problému – RECURSIVE CASE

Faktoriál rekurzívne

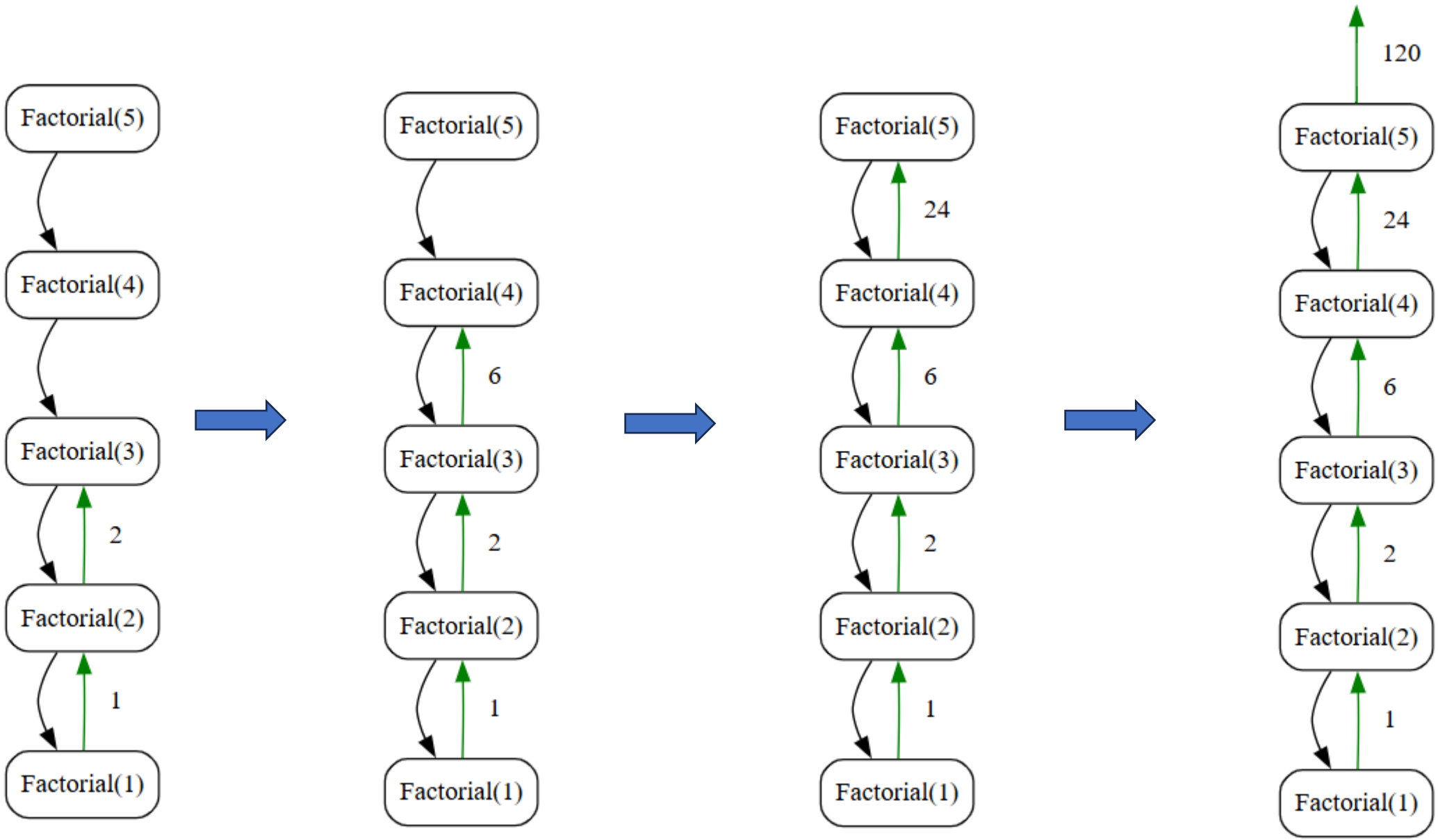
```
    Factorial(n)
1.      if n = 1                BASE CASE
2.          return 1
3.      else
4.          return n * Factorial(n – 1)        RECURSIVE CASE
```

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

Výpočet 5! – zavoláme funkciu Factorial(5)



Výpočet 5! – zavoláme funkciu Factorial(5)



Príklad – Fibonacciho číslo

Fibonacciho postupnosť: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

- každé číslo je súčtom dvoch predchádzajúcich čísel

Výpočet:

$$F_n = \begin{cases} 0, & \text{pro } n = 0; \\ 1, & \text{pro } n = 1; \\ F_{n-1} + F_{n-2} & \text{jinak.} \end{cases}$$

BASE CASE

BASE CASE

RECURSIVE CASE

Fibonacciho číslo rekurzívne

```
    Fibonacci(n)
1.      if n = 0            BASE CASE
2.          return 0
3.      if n = 1            BASE CASE
4.          return 1
5.      else                RECURSIVE CASE
6.          return Fibonacci(n – 1) + Fibonacci(n – 2)
```

Funkcia bude volať samú seba 2x, dokým nedosiahne BASE CASE, potom začne vracať výsledky jednotlivých rekurzívnych volaní

Vizualizácia

- https://www.recursionvisualizer.com/
- https://recursion.vercel.app/

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

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

```
Partition(A, p, r)
1.    x ← A[r]
2.    i ← p - 1
3.    for j ← p to r - 1
4.        if A[j] <= x
5.            i ← i + 1
6.            swap(A[i], A[j])
7.    swap(A[i + 1], A[r])
8.    return i + 1
```

- x - pivot
- p, r, i, j – indexy

Quick Sort - časová zložitosť

- Veľkosť vstupu – veľkosť vstupného poľa
- V najhoršom prípade
  - $\Theta(n^2)$ kvadratická
  - Vstupné pole je zoradené vzestupně alebo sestupně a ako pivot sa vždy pri volaní Partition vyberie najmenší alebo najväčší prvok
  - Vznikne maximálne nevyvážené pole (ľavá alebo pravá polovica sú prázdne)
- V priemernom prípade
  - $\Theta(n \log n)$ linearne logaritmická
  - Pole sa podľa pivota rozdelí na dve polovice, ktoré nie sú rovnako veľké, ale ani jedna nie je prázdna
- V najlepšom prípade
  - $\Theta(n \log n)$ linearne logaritmická
  - Pole sa podľa pivota rozdelí na dve rovnako veľké polovice (max. môže byť jedno o 1 prvok menšie)
  - Vznikne maximálne vyvážené pole

Quick Sort

- Animácia:
  - Quick Sort Algorithm

Úkol

- Simulácia algoritmu
  - Simulujte kroky algoritmu QuickSort na postupnosti

    A = [9, 1, 7, 6, 0, 8, 4, 5]

- Implementácia
  - Pomocou šablóny QuickSort-Sablona.c/.py na GitHube naprogramujte v C alebo Pythone Quick Sort
  - (pomôžte si pseudokódom)

- Spôsob odovzdávania – info na GitHube na konci README