Vyčíslitelnost a složitost výpočtů

Asymptotiky

- V informatice se asymptotika používá k odhadu časové a prostorové složitosti algoritmů.
- Porovnání efektivity algoritmů a výběr toho nejlepšího pro konkrétní problém.
- Zajištění škálovatelnosti systémů při růstu dat.

Asymptotiky

- **f(x)=O(g(x))**: Funkce f(x) roste nejvýše tak rychle jako g(x). (Horní odhad růstu.)
- **f(x)=Ω(g(x))**: Funkce f(x) roste nejméně tak rychle jako g(x). (Dolní odhad růstu.)
- **f(x)=Θ(g(x))**: Funkce f(x) a g(x) rostou stejně rychle. (Funkce mají stejnou asymptotickou rychlost růstu.)
- **f(x)=o(g(x))**: Funkce f(x)roste pomaleji než g(x). (Rychlost růstu g(x) je větší než u f(x).

Asymptotiky

Pořadí rychlosti růstu funkcí:

- Různé typy funkcí mají různé asymptotické rychlosti růstu. Nejčastěji používané funkce v analýze algoritmů seřazujeme podle jejich rychlosti růstu:
- **Logaritmické funkce**: $\log(x)$, $\log_2(x)$, $\log_3(x)$, ...

  Tyto funkce rostou velmi pomalu.
- **Polynomy**: x, x^2, x^3, ...

  Polynomy rostou rychleji než logaritmické funkce.
- **Exponenciální funkce**: 2^x, 3^x, 4^x, ...

  Exponenciální funkce rostou rychleji než polynomy.
- **Funkce vyššího řádu**: $2^{(x^2)}$, $2^{(x^3)}$, ...

  Tyto funkce rostou extrémně rychle.

Časová složitost algoritmu

Časová složitost algoritmu

- Časová složitost algoritmu popisuje, jak se doba běhu algoritmu mění v závislosti na velikosti vstupních dat.
- Především u velkých dat.
- Časová složitost se obvykle vyjadřuje pomocí tzv. **asymptotické notace** (Horní odhad růstu).

Časová složitost algoritmu

O(1) – Konstanta:

- Algoritmus běží vždy ve stejném čase bez ohledu na velikost vstupu.
- Příklad: Přístup k prvku v poli pomocí indexu.

O(n) – Lineární:

- Doba běhu algoritmu roste přímo úměrně s velikostí vstupu.
- Příklad: Prohledání nesetříděného seznamu s n prvky.

O(log n) – Logaritmická:

- Časová složitost roste pomaleji než velikost vstupu, což je typické pro algoritmy, které opakovaně dělí problém na menší části.
- Příklad: Binární vyhledávání.

O(n log n) – Lineárně logaritmická:

- Typická pro efektivní třídicí algoritmy jako je *quicksort* nebo *mergesort*.
- Časová složitost roste rychleji než lineární, ale pomaleji než kvadratická.

Časová složitost algoritmu

$O(n^2)$ – Kvadratická:

- Doba běhu roste úměrně druhé mocnině velikosti vstupu.
- Příklad: Třídění bublinkou (bubble sort).

$O(2^n)$ – Exponenciální:

- Časová složitost exponenciálně roste s velikostí vstupu.
- Příklad: Rekurzivní řešení problému cestujícího obchodníka.

$O(n!)$ – Faktoriální:

- Extrémně náročné algoritmy, kde doba běhu dramaticky roste s velikostí vstupu.
- Příklad: Brute-force.

Časová složitost algoritmu

1. Identifikace základních operací algoritmu

Každý algoritmus je sestaven z jednotlivých operací, jako jsou:

- Přiřazení hodnot proměnným
- Podmínky (if-else)
- Smyčky (for, while)
- Rekurze

Časová složitost algoritmu

2. Analýza smyčky

- Smyčky (cykly) mají velký vliv na časovou složitost algoritmu, protože určují, jak často se provádí základní operace.
- **Jednoduchá smyčka, která běží n krát** má časovou složitost **O(n)**.
- **Smyčka uvnitř smyčky** (vnořená smyčka) má složitost násobenou, tj. pro dvě vnořené smyčky, kde každá běží n krát, bude celková složitost **$O(n^2)$**.

Časová složitost algoritmu

3. Podmínky

- Podmínky (if-else) mohou mít různé větve, ale z hlediska asymptotické notace se uvažuje vždy ta nejhorší možná větev (*worst-case* analýza).
- Pokud rozhodnutí v podmínce nijak neovlivňuje složitost (provádí se jen jednou), přidává jen **O(1)** (konstantní složitost).

Časová složitost algoritmu

4. Rekurze

- U rekurzivních algoritmů musíte vytvořit tzv. rekurentní vztah a analyzovat ho.

Příklad:

- **Binární vyhledávání**: Rozdělí vstup na polovinu v každém kroku, což vede k **O(log n)**.
- **Třídění pomocí merge sortu**: Rozděluje vstupní seznam a kombinuje je zpět, což vede k **O(n log n)**.

Časová složitost algoritmu

```
public static int BinarySearch(int[] arr, int x)
{
    int left = 0;
    int right = arr.Length - 1;

    while (left <= right)
    {
        int mid = left + (right - left) / 2;

        if (arr[mid] == x)
            return mid;

        if (arr[mid] > x)
            right = mid - 1;
        else
            left = mid + 1;
    }

    return -1;
}
```

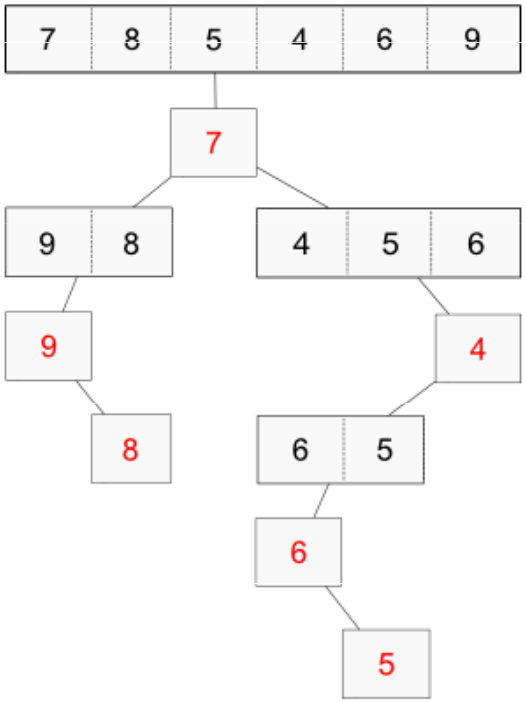
Časová složitost algoritmu

```
static void BubbleSort(int[] arr)
{
    for (int i = 0; i < arr.Length - 1; i++)
    {
        for (int j = 0; j < arr.Length - i - 1; j++)
        {
            if (arr[j + 1] < arr[j])
            {
                int tmp = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = tmp;
            }
        }
    }
}
```

Časová složitost algoritmu

```java
public static void shakerSort(int[] array) {
    for (int i = 0; i < array.length/2; i++) {
        boolean swapped = false;
        for (int j = i; j < array.length - i - 1; j++) {
            if (array[j] < array[j+1]) {
                int tmp = array[j];
                array[j] = array[j+1];
                array[j+1] = tmp;
                swapped = true;
            }
        }
        for (int j = array.length - 2 - i; j > i; j--) {
            if (array[j] > array[j-1]) {
                int tmp = array[j];
                array[j] = array[j-1];
                array[j-1] = tmp;
                swapped = true;
            }
        }
        if(!swapped) break;
    }
}
```

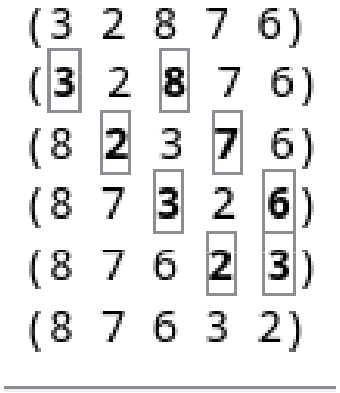
Časová složitost algoritmu




```
public static void Quicksort(int[] array, int left, int right)
{
    if (left < right)
    {
        int boundary = left;
        for (int i = left + 1; i < right; i++)
        {
            if (array[i] > array[left])
            {
                Swap(array, i, ++boundary);
            }
        }
        Swap(array, left, boundary);
        Quicksort(array, left, boundary);
        Quicksort(array, boundary + 1, right);
    }
}

/**
 * Prohodi prvky v zadanem poli
 * @param array pole
 * @param left prvek 1
 * @param right prvek 2
 */
private static void Swap(int[] array, int left, int right)
{
    int tmp = array[right];
    array[right] = array[left];
    array[left] = tmp;
}
```

Časová složitost algoritmu




```
public static void SelectionSort(int[] array)
{
    for (int i = 0; i < array.Length - 1; i++)
    {
        int maxIndex = i;
        for (int j = i + 1; j < array.Length; j++)
        {
            if (array[j] > array[maxIndex]) maxIndex = j;
        }
        int tmp = array[i];
        array[i] = array[maxIndex];
        array[maxIndex] = tmp;
    }
}
```