Семинар 3 – Долна граница за сложност на сортиращи алгоритми. Counting Sort

Сортиращи алгоритми – обобщение

	Best case	avg. case	worst case	сложност по памет	стабилен	адаптивен
Bubble sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	✓	зависи
Selection sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	×	×
Insertion sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	✓	✓
Merge sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n\log n)$	$O(n)$	✓	×
Quick sort	$\Theta(n\log n)$	$\Theta(n\log n)$	$\Theta(n^2)$	$O(\log n)$	×	×

* std::merge и std::partition

Въпрос: Можем ли да намалим сортиращ алгоритъм по-бърз от $\Theta(n\log n)$?

Отговор: Ако алгоритъмът е базиран на директни сравнения, НЕ. Тези алгоритми имат долна граница $\Omega(n\log n)$.

Идея: Нека имаме масива $[a_1, a_2, a_3]$, който ще сортираме $\Rightarrow$търсим такава пермутация на елементите му, че $a_{i_1} < a_{i_2} < a_{i_3}$. Варианти за решение:

1) Brute force алгоритъм – минаваме през всички пермутации (пр. Bogosort)

2) Чрез директни сравнения – на една стъпка (1 сравнение) ще отпадат няколко пермутации (грубо половината)
В този случай имаме 3! пермутации, в общия – $n!$

$$n! \xrightarrow{\text{ст.1}} \frac{n!}{2} \xrightarrow{\text{ст.2}} \frac{n!}{3} \longrightarrow \dots \longrightarrow 1$$

общо $\log(n!)$ стъпки $\approx n\log(n)$

→ Алгоритъм за сортиране на масив с ограничен размер на домейна (напр. масив от цифри) – Counting Sort

Идея: Броим колко пъти се среща всяка ст-ст от домейна, след което ги отпечатваме в сортиран вид.

Пример 1: arr: 3 5 2 2 1 6 4 3

countArr:

1	2	3	4	5	6
1	2	2	1	1	1

⇒ 1 2 2 3 3 4 5 6

⚠ Проблем – стабилност и работа с комплексни обекти

Пример 2: масив от {име-на-студент, оценка}

Стъпка 1. Броим срещанията → countArr. [дълж. размера на домейна]

Стъпка 2. Уплътняваме countArr – във i-тия елемент добавяме (i-1)-вия последователно, т.е. накрая имаме масив, съдържащ броя на елементите ≤ от i-тия

Стъпка 3. Попълваме resultArr [дълж. размера на arr] ⚠ отзад-напред, за да е стабилен.

Стъпка 4. Прехвърляме данните от resultArr в arr.

arr: [{A, 3}, {B, 4}, {C, 3}, {D, 6}, {E, 2}, {F, 4}]

1) countArr: [1, 2, 2, 0, 1]

2) countArr: [1, 3, 5, 5, 6] → ≤6; ↙ брой ≤2; ↓ ≤3; ↘ ≤4; ↘ ≤5

3) resultArr: [{E, 2}, {A, 3}, {C, 3}, {B, 4}, {F, 4}, {D, 6}]

4) arr: —11—

Анализ: n-размер на arr, k-размер на домейна

* по памет – $\Theta(n+k)$
* по време $\Theta(n) + \Theta(k) + \Theta(n) + \Theta(n) = \Theta(n+k)$ (стъпки ① ② ③ ④)
* стабилен – да
* адаптивен – не

=2=