АЛГОРИТМИ ЗА СОРТИРАНЕ

Димитър Драгостинов
F54320

Има 2 типа алгоритми за сортиране :

- Базирани на сравнения
- Базирани на броене

АЛГОРИТМИ БАЗИРАНИ НА СРАВНЕНИЯ

- Бавни алгоритми
- Insertion sort
- Selection sort
- Bubble sort

Running time : $O(n^2)$ –

НЕ е достатъчно за състезания

- Добри алгоритми
- Merge sort
- Quick sort
- Heap sort

Сложност : O(n lgn)

Достатъчно за състезания, но може и по-добре...

АЛГОРИТМИ БАЗИРАНИ НА БРОЕНЕ

- Counting sort – Сложност: O(n+m)
- Radix sort – Сложност: O(d(n+m))

УСТОЙЧИВО СОРТИРАНЕ

Когато сортираме индексите е от значение как действаме с еднаквите индекси, ако има такива.

За всеки два елемента k_i и k_j от множеството S, такива че $k_i = k_j$, ако k_i предшества k_j преди сортирането, то тогава k_i трябва да предшества k_j и след сортирането.

СОРТИРАНЕ ЧРЕЗ БРОЕНЕ (COUNTING SORT)

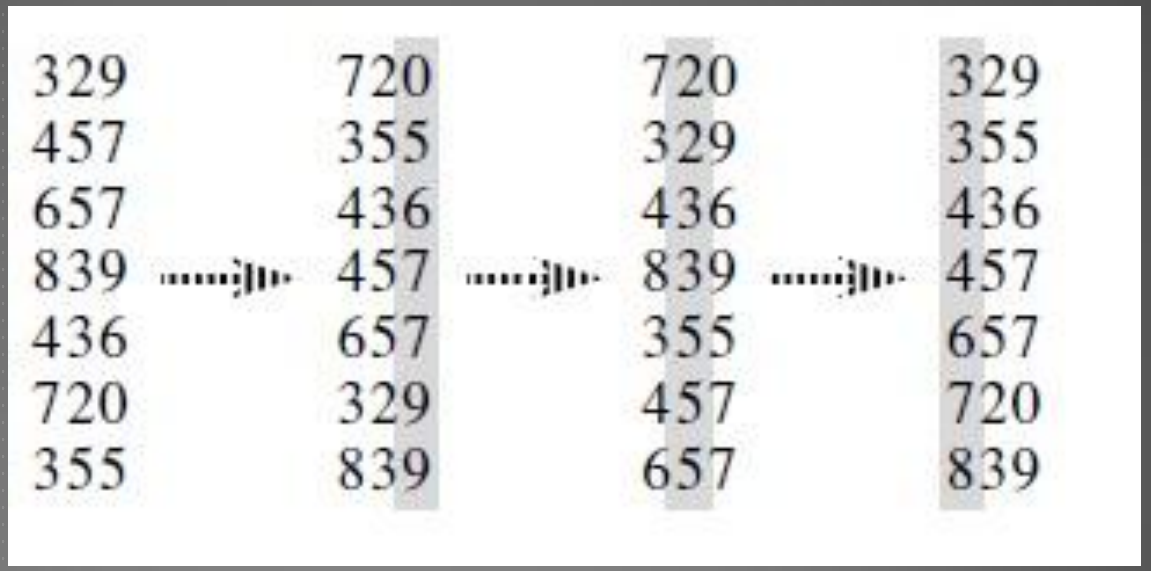
- Основната идея тук е да определеним, за всеки входящ елемент x, броя на елементите, които са по-малки от него
- Тази иформация се използва за да сложим елемент x директно на мястото му в изходния масив

- Input: array A [0… n-1], keys [0…n-1] (parallel)
- Output: array A [0…n-1] sorted according to the keys array

```
let B be a temporary array [0…n-1]
let C be an array with counters for each key [0 – 255]
for i ← 0 to k-1
        do C[ i ] ← 0
for i ← 0 to n-1
        do C[ keys[ i ] ] ← C[ keys[ i ] ] + 1
for i ← 1 to k-1
        do C[ i ] ← C[ i ] + C[ i – 1 ]
for i ← n -1 down to 0
        do B[ C[ keys[ i ] ] - 1 ] ← A[ i ]
        C[ keys[ i ] ] ← C[ keys[ i ] ] – 1
for i ← 0 to n-1 A[ i ] ← B[ i ]
```

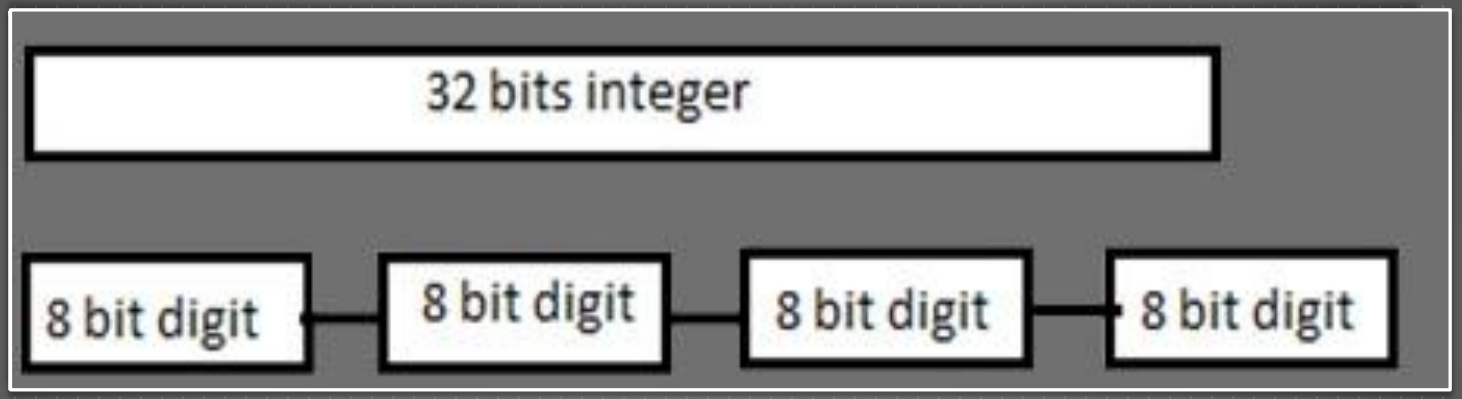
RADIX SORT

- Този метод използва алгоритъм за устойчиво сортиране, като сортира по всяка „цифра“ последователно.



Компютрите съществуват в двоичен свят, така че не може да използваме десетична бройна система, за да дефинираме цифра.

Може да вземем поредица от битове(8 е най-разумният избор) и да си представим, че формират 1 цифра от бройна система с база 256.



- Input unsorted array **A**[0…n-1], number of digits **d**
- Output sorted array **A**[0…n-1]

Let **digit**[0…n-1] be a parallel array of A consisting with the current digit of each element

curr_digit ← 0xFF // the first digit == 11111111(bin)

for **j** ← 0 to d – 1

do for **i** ← 0 to n - 1

do digit[i] ← A[i] & curr_digit

digit[i] ← digit[i] >> d*8

counting_sort(A, digit, n)

curr_digit ← curr_digit << 8

СРАВНИТЕЛЕН АНАЛИЗ

Code Blocks

Input size	Radix sort	Sort()	Stable_sort()	Heap_sort()
100000	0.01	0.01	0.02	0.02
1000000	0.08	0.13	0.19	0.33
10000000	0.72	1.28	1.62	4.72

Visual Studio

Input size	Radix sort	Sort()	Stable_sort()	Heap_sort()
100000	0.01	0.21	0.26	0.20
1000000	0.13	2.69	2.95	2.39
10000000	1.34	31.77	35.69	32.02

ПРИЛОЖЕНИЯ В СЪСТЕЗАЛНОТО ПРОГРАМИРАНЕ

- **Проверка за уникалност на елементите на масив** – може да се сортира и след това да се обходи като се сравняват всеки 2 съседни елемента.
- **Изтриване на повтарящи се елементи** – може да се реализира чрез сортиране на масива и прилагане на функцията unique от STL библиотеката.
- **Отговор на запитване за пореден елемент** – ако искаме да намерим к-тия елемент по големина от масива можем да го сортираме и да прочетем a[k]. Като специални случаи можем да намерим медиана, най-голям и най-малък елемент.

ПРИЛОЖЕНИЯ В СЪСТЕЗАЛНОТО ПРОГРАМИРАНЕ (ПРОДЪЛЖЕНИЕ)

- **Честота на срещане в масив –** след като е сортиран масива можем ефективно да отоговрим на въпросите:
  - Кой е най-често срещания елемент?
  - Колко са, ако е повече от един и т.н.
- **Възвръщане на начална подредба –** можем да направим някаква структура, която да пази индекса, с който сме прочели елемента и след обработка на масива да възобновим началната наредба на елементите.
- **Сечение и обединение на множества –** Ако сортираме 2те множества можем да ги слеем като последователно взимаме по-малкия елемент от всяка двойка.

ПРИЛОЖЕНИЯ В СЪСТЕЗАЛНОТО ПРОГРАМИРАНЕ (ПРОДЪЛЖЕНИЕ)

- **Ефективно търсене – след като масива е сортиран може да използваме двоично търсене, за да го отговорим на въпроса дали даден елемент е във масива и кой е най-големия елемен по-малък от търсения или кой е най-малкия елемент по-голям от търсения.**

Благодаря за вниманието!