FS12

Week 06
Sorting Algorithms II

Mikhail Masyagin
Daniil Devyatkin

K-th order statistic I

- Let's assume that we have to find **K-th** element in array if it was sorted. This task is named **"K-th order statistic"**.
- Naive solution: sort an array and get its K-th element:
  - Pros: easy to implement, possible to use simplest and well-known STL methods.
  - Cons: slow on large arrays.

K-th order statistic II

- Better solution: in previous presentation we discussed **qsort** sorting algorithm. Internally it uses partition logic. After each partition 3 variants are possible (**k** – **”K-th order statistic”**, m - **“support element“**):
  - **k = m** (found);
  - **k < m** (check left part of array);
  - **k > m** (check right part of array).

K-th order statistic III

```
def k_order_stat(a: List[int],k: int):
    l, r = 0, len(a)
    while True:
        mid = partition(a, l, r)
        if mid == k:
            return a[mid]
        elif k < mid:
            r = mid
        else:
            l = mid + 1
```

K-th order statistic IV

- Looks like default **qsort**, however, there are no recursive **qsort** calls.
- You can prove that in most cases algorithm's time complexity will be **O(n)**, however, in come cases you can get **O(n^2)** due to bad selection of **"support element"**.

BFRPT

- History: **BFRPT** – Manuel **B**lum, Robert **F**loyd, Vaughan Ronald **P**ratt, Ron **R**ivest, Robert **T**arjan. It's a modification of **"K-th order statistic"** with guaranteed linear time complexity.
- Algorithm:
  - Split array into subarrays: each subarray consists of 5 elements (last subarray – **n mod 5**);
  - Sort each subarray and select median element from all of them.
- Select median from all medians. This element will be a **"support element"**. Partition array over it.

Linear Sorting Algorithms

- Previously we discussed that best asymptotic complexity for sorting algorithm with comparator is **Ω(Nlog(N))**.
- Sometimes we can beat this asymptotic complexity if we have additional knowledge about our array.

Counting Sort

- Let's assume that array, that should be sorted, consists of **N** unique values and **N** is small enough. Then we can simply create mapping from unique elements of array and calculate them in linear time. After that we can fill original array as an empty one using counts from map.
- Example: for today biggest long-liver age is much less than 200. You can create map (array) of 200 elements with keys from 0 to 199 and fill its counts from non-sorted array of ages. Than you can fill it.

Counting Sort

- Let's assume that array, that should be sorted, consists of **N** unique values and **N** is small enough. Then we can simply create mapping from unique elements of array and calculate them in linear time. After that we can fill original array as an empty one using counts from map.
- Example: for today biggest long-liver age is much less than 200. You can create map (array) of 200 elements with keys from 0 to 199 and fill its counts from non-sorted array of ages. Than you can fill it.

Distribution Sort

- If number of unique elements is too big to use Counting Sort, you can use Radix Sort. It splits the key on multiple subkeys.
- For example, for strings with equal length each subkey is substring or string letter.
- For integers each subkey is I’th rank of selected number.
- You can try to implement Distribution Sort & Counting Sort by yourself.


Прощай
Пока - пока!
До свидания