Improving Quicksort

- 三路快速排序 (Split-end partition)　推荐指数★★★★

  - What will happen to quicksort if the array under sorting consists of a large number of identical elements?
  - And the extreme case is that the array is filled with identical elements only!

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

ELSEVIER

Improving multikey Quicksort for sorting strings with many equal elements ☆,☆☆

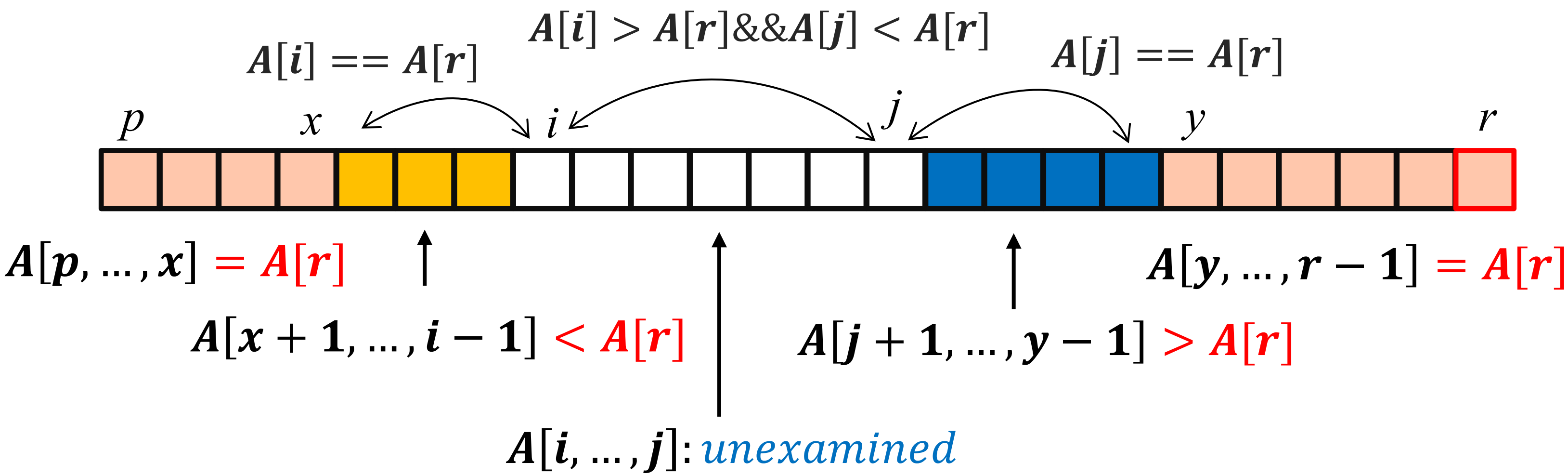
Eunsang Kim, Kunsoo Park *

School of Computer Science and Engineering, Seoul National University, 599 Gwanak-ro, Gwanak-gu, Seoul, 151-742, South Korea

Improving Quicksort

- 聚集相同元素（三路快速排序）Split-end partition

（1）在划分过程中将与基准值相等的元素放入数组两端



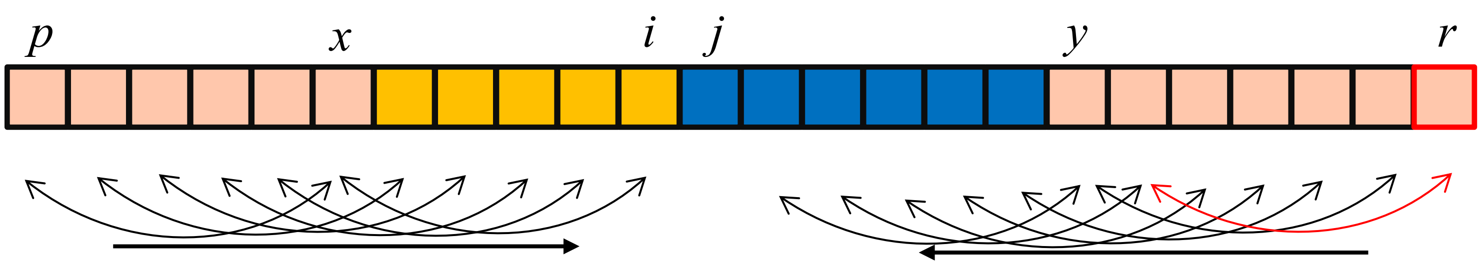
Improving Quicksort

Q: 三路快排是稳定排序吗？

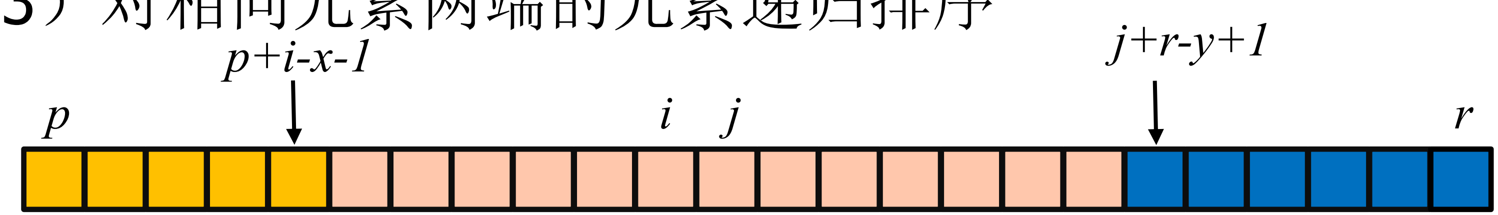
三路快速排序: Split-end partition

（1）在划分过程中将与基准值相等的元素放入数组两端

（2）划分结束后，再将两端的相同元素移到基准值周围



（3）对相同元素两端的元素递归排序



sort(A, p, p+i-x-1)

sort(A, j+r-y+1, r)

Simpler partition

```
partition(A, p, r)        //A[r]为基准值
i ← p − 1
h ← p − 1                 //新增指针
for j ← p to r − 1
    do if A[j] ≤ A[r]
        then h ← h + 1
            swap(A[j], A[h])
            if A[h] < A[r]
                then i ← i + 1
                    swap(A[i], A[h]))
swap(A[r], A[h + 1]))
return h + 1
```

$A[p, \ldots, i] < A[r]$

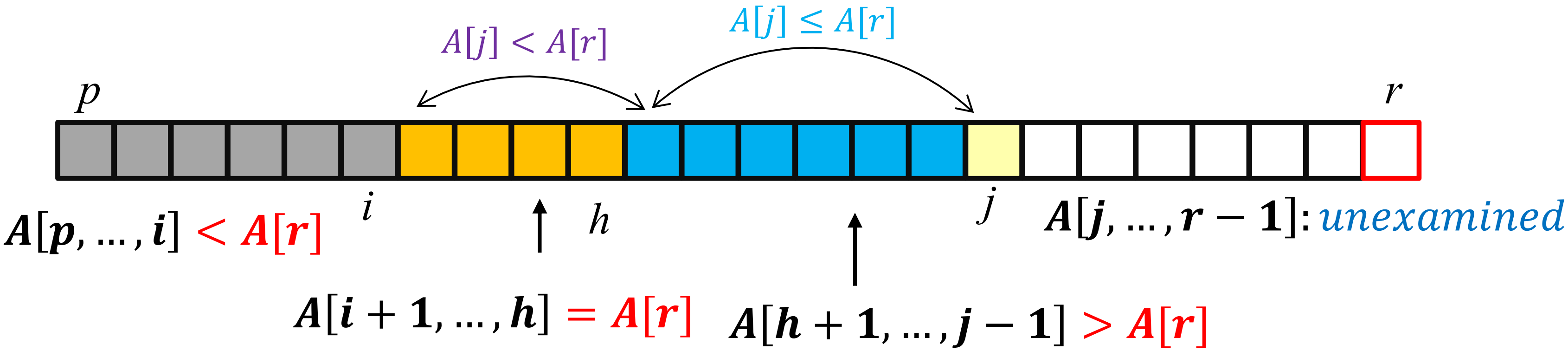
$A[j, \ldots, r-1]$: *unexamined*

$A[i+1, \ldots, h] = A[r]$

$A[h+1, \ldots, j-1] > A[r]$

Simpler partition

```
partition(A, p, r)      //A[r]为基准值
    i ← p − 1
    h ← p − 1           //新增指针
    for j ← p to r − 1
        do if A[j] ≤ A[r]
            then h ← h + 1
                 swap(A[j], A[h])
                 if A[h] < A[r]
                     then i ← i + 1
                          swap(A[i], A[h]))
    swap(A[r], A[h + 1]))
    return h + 1
```



Simpler partition

```
qsort(A, i, j)        //主程序
if j ≤ i
      then return
pivotindex ← findpivot(A, i, j)
swap(A[pivotindex], A[j])
k ← partition(A, i, j)  //聚集相同元素划分
qsort(A, k+1, j)        //大于基准值的元素区间
while k>0 && A[k-1]=A[k]  //与基准相同元素不需要再排
      do k ← k – 1
qsort(A, i, k-1)        //小于基准值的元素区间
```

$A[p, \ldots, i] < A[r]$

$A[j, \ldots, r-1]$: *unexamined*

$A[i+1, \ldots, h] = A[r]$

$A[h+1, \ldots, j-1] > A[r]$

Improving Quicksort

- Multikey Quicksort（三路基数快速排序）

1. Sort an array S of (pointers to) strings in lexicographically nondecreasing order.
2. By quicksort, it recursively sorts the array S of strings that are known to be identical in the first d-1 characters. (d>0 and d=1 at initial time)
3. Like radix sort, it partitions S to sub-arrays smaller than, equal to, and greater than the d-th character of the pivot string.

Improving Quicksort

- Multikey Quicksort（三路基数快速排序）

<u>mkqsort(</u><u>S, i, j, d</u>**<u>)</u>** $//\forall k \in (i,j]\ \forall t \in [0, d-1)\colon\ S[k-1][t] = S[k][t]$

if j $\leq i$

then return

k ← **mkpartition(**S, i, j, d**)** //choose $S[j]$ as pivot

mkqsort(S, k+1, j, d**)** //第d位字符大于基准字符的子集排序

//任然选第d位字符作为基准(?)

$m \leftarrow k$

while $m > 0$ && $S[m-1][d-1] = S[m][d-1]$

//第d位字符相同

do $m \leftarrow m - 1$

mkqsort(S, m, k, d+1**)** //第d位字符等于基准字符的子集排序

mkqsort(S, i, m-1, d**)** //第d位字符小于基准字符的子集排序

Improving Quicksort

mkpartition(S, p, r, d)　　//S[r]为基准

$i \leftarrow p - 1$

$h \leftarrow p - 1$　　//聚集前d位字符与基准相同的字符串

for $j \leftarrow p$ **to** $r - 1$

　　do if $S[j][d-1] \leq S[r][d-1]$　//比较第d位字符

　　　　then $h \leftarrow h + 1$

　　　　　　swap($S[j]$, $S[h]$)　　//交换字符串（指针）

　　　　　　if $S[h][d-1] < S[r][d-1]$

　　　　　　　　then $i \leftarrow i + 1$

　　　　　　　　　　swap($S[i]$, $S[h]$)

swap($S[r]$, $S[h+1]$))

return $h + 1$

Quicksort applications

(1) 百度面试题：假设一整型数组存在若干正数和负数，现在通过某种算法使得该数组的所有负数在正数的左边，且保证负数和正数间元素相对位置不变。时间复杂度O(n)

算法：选0作基准值进行划分！

(2) 长度为n的正整数数组，分成两个不想交的子数组并分别计算其中的元素和。求两个子数组长度差最小，且和相差最大的分法。

分法1：从小到大快速排序，前$\left\lfloor \frac{n}{2} \right\rfloor$个元素为一组，其它一组

*时间复杂度$O(n\log(n))$

Quicksort applications

(1) 百度面试题：假设一整型数组存在若干正数和负数，现在通过某种算法使得该数组的所有负数在正数的左边，且保证负数和正数间元素相对位置不变。时间复杂度O(n)

(2) 长度为n的正整数数组，分成两个不想交的子数组并分别计算其中的元素和。求两个子数组长度差最小，且和相差最大的分法。

分法2：只要找到排位$\left\lfloor\frac{n}{2}\right\rfloor$的基准进行一次划分，不需要继续划分和排序！？

$\boxed{\text{快速查找算法：QuickSearch}}$

*时间复杂度$O(n)$

QuickSearch(A, l, r, k**)**

//数组A[l..r]存放排位在 l 到 r 的元素，初始时A[0..n-1]

// 找出排位k的元素 ($l \le k \le r$)

---

if l = r **then return** l   // l=r=k

t ← **Partition(**A, l, r**)**

//以A[r]为基准值进行划分，返回其位置（排位）

if t = k **then return** t   //A[r]就是排位k的元素，返回位置t

if k < t **then**

QuickSearch(A, l, t-1, k**)**

// find k-th smallest element in A[l..t-1]

else //k > t

QuickSearch(A, t+1, r, k**)**

// find k-th smallest element in A[t+1..r]

Quick Search

```
QuickSearch(A, l, r, k)
    if l = r then return l
    t ← Partition(A, l, r)
    if t = k then return t
    if k < t then
        QuickSearch(A, l, t-1, k)
    else
        QuickSearch(A, t+1, r, k)
```

Time complexity

Worst case: $O(n^2)$

Best case: $O(n)$

Average case: $O(n)$?

- If we choose a random pivot, this will, on average, divide a set of n items into two sets of size $\frac{n}{4}$ and $\frac{3n}{4}$.
  - 90 % of the time the width will have a ratio of 1:19 or better.

$$T(n) = T\left(\frac{19}{20}n\right) + \Theta(n)$$