

# Sorting atomic items

## Chapter 5

Distribution based sorting paradigms

# The distribution-based sorting

QuickSort is an **in place** algorithm, but...

Consider the stack for the recursive calls.

For balanced partitions:  $O(\log n)$  space

Worst case of unbalanced partitions:  $\Omega(n)$  calls,  $\Theta(n)$  space!!

QuickSort is modified.

We can bound the recursive depth. **Algorithm Bounded**

Based on the fact that: **Quicksort does not depend on the order in which recursive calls are executed!**

Small arrays can be better sorted with **InsertionSort** (when  $n$  is typically of the order of tens).

The modified version mixes **one recursive call** with an iterative **while loop**.

# Algorithm Bounded( $S, I, j$ )

---

**Algorithm 5.6** The binary quick-sort with bounded recursive-depth:

---

```
1: while ( $j - i > n_0$ ) do
2:      $r =$  pick the position of a “good pivot”;
3:     swap  $S[r]$  with  $S[i]$ ;
4:      $p =$  PARTITION( $S, i, j$ );
5:     if ( $p \leq \frac{i+j}{2}$ ) then n > no
6:         BOUNDEDQS( $S, i, p - 1$ );
7:          $i = p + 1$ ;
8:     else
9:         BOUNDEDQS( $S, p + 1, j$ );
10:     $j = p - 1$ ;
11:    end if
12: end while
13: INSERTIONSORT( $S, i, j$ ); n ≤ no
```

---

# Algorithm Bounded( $S, I, j$ )

- The recursive call is executed on the smaller part of the partition
- It drops the recursive call on the larger part of the partition in favor of another execution of the **while-loop**.
- **Ex:**

10 3 7 15 21 18 2 11

pivot =  $S[2]=3$

Partition

2 | 3 | 10 7 15 21 18 11

Only one recursive call on  $S(i, p-1)$ ; For the larger part  $S(i+1, j)$  we iterate to the while loop

- Technique: **Elimination of Tail Recursion**
- Bounded takes  $O(n \log n)$  time in average and  $O(\log n)$  space.

# Multi-way QuickSort

- **2-level model**:  $M$  = internal memory size ;  $B$  = block size ;
- Split the sequence  $S$  into  $k = \Theta(M/B)$  sub-sequences using  $k-1$  pivots.
- We would like balanced partitions, that is of  $\Theta(n/k)$  items each.
- **Select  $k-1$**   $s_1, s_2, \dots, s_{k-1}$  "good pivots" is not a trivial task!

Let **bucket**  $B_i$  the portion of  $S$  between pivot  $s_{i-1}$  and  $s_i$ .

We want guarantee that  $|B_i| = \Theta(n/k)$  for all buckets.

So, at the first step of Multi-way QuickSort size of portions  $n/k$

at the second step the size of portions  $n/k^2$

at the third step the size of portions  $n/k^3$

.....

Stop when  $n/k^i \leq M$ :  $n/M \leq k^i$  ,  $i$  is the number of steps

$$i \geq \log_k n/M = \log_{M/B} n/M$$

This number of steps is enough to have portions shorter than  $M$ , and sorted in internal memory!

**Partition** takes  $O(n/B)$  I/O's (dual to MS) : 1 input block,  $k$  output blocks.

# Multi-way QuickSort

- **2-level model:**  $M$  = internal memory size ;  $B$  = block size ;
- Split the sequence  $S$  into  $k = \Theta(M/B)$  sub-sequences using  $k-1$  pivots.
- We would like balanced partitions, that is of  $\Theta(n/k)$  items each.
- **Select  $k-1$**   $s_1, s_2, \dots, s_{k-1}$  "good pivots" is not a trivial task!

Let **bucket**  $B_i$  the portion of  $S$  between pivot  $s_{i-1}$  and  $s_i$ .

We want guarantee that  $|B_i| = \Theta(n/k)$  for all buckets.

So, at the first step of Multi-way QuickSort size of portions  $n/k$

at the second step the size of portions  $n/k^2$

at the third step the size of portions  $n/k^3$

.....

Stop when  $n/k^i \leq M$ :  $n/M \leq k^i$  ,  $i$  is the number of steps

$$i \geq \log_k n/M = \log_{M/B} n/M$$

This number of steps is enough to have portions shorter than  $M$ , and sorted in internal memory!

**Partition** takes  $O(n/B)$  I/O's (dual to MS) : 1 input block,  $k$  output blocks.

# Multi-way QuickSort

Find  $k$  good pivots efficiently.

Randomized strategy called **oversampling**.

$\Theta(ak)$  items are sampled,  $a \geq 0$  parameter of the oversampling.

---

**Algorithm 5.7** Selection of  $k - 1$  good pivots via oversampling

---

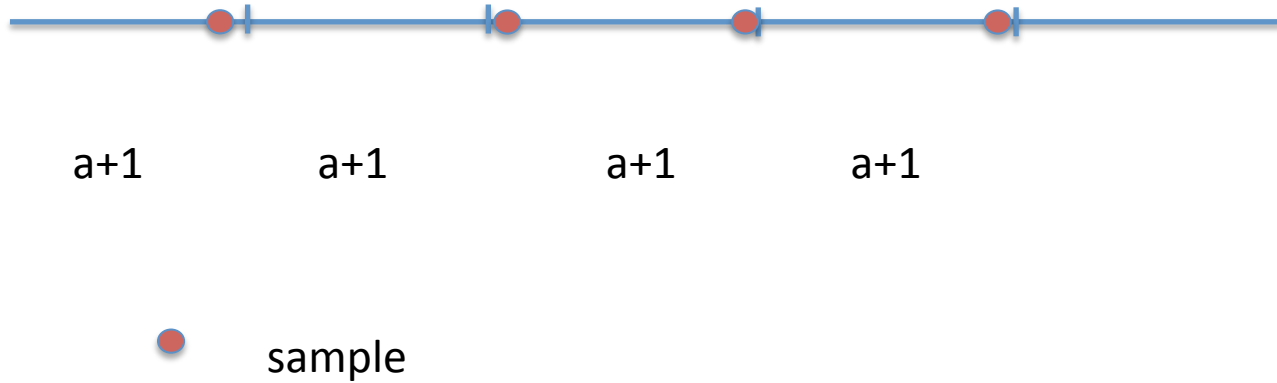
- 1: Take  $(a + 1)k - 1$  samples at random from the input sequence;
  - 2: Sort them into an ordered sequence  $A$ ;
  - 3: For  $i = 1, \dots, k - 1$ , pick the pivot  $s_i = A[(a + 1)i]$ ;
  - 4: **return** the pivots  $s_i$ ;
- 

$\Theta(ak)$  candidates  
 $\Theta(ak)\log(ak)$  time  
Select  $k-1$  pivots  
evenly distributed

Balanced selection of  $s_i = A[(a+1)i]$  should provide good pivots!!

# Multi-way QuickSort

- $K=5$





# Multi-way QuickSort

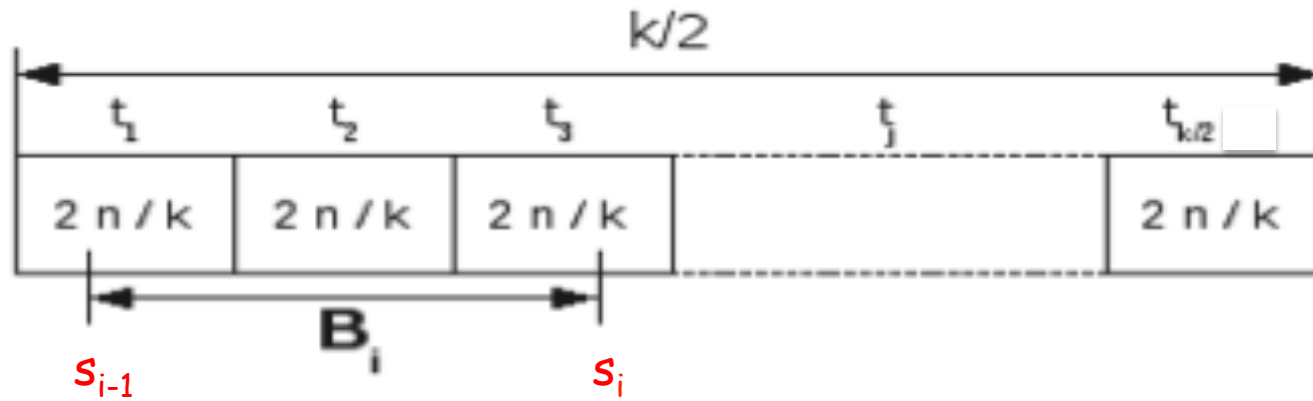
- The larger is  $a$  the closer to  $\Theta(n/k)$
- If  $a=n/k$  the elements of set  $A$  cannot be sorted in  $M$  !
- If  $a = 0$  the selection is fast, but unbalanced partitions are more probable.
- Good choice for  $a$ :  $\Theta(\log k)$ . Pivot-selection costs  $\Theta(k \log^2 k)$

**Lemma.** Let  $k \geq 2$ ,  $a + 1 = 12 \ln k$ . A sample of  $(a+1)k-1$  suffices to ensure that all buckets receive less than  $4n/k$  elements, with probability at least  $\frac{1}{2}$ .

**Proof.** We find an upper bound to complement event: there exists 1 bucket containing more than  $4n/k$  elements with probability at most  $\frac{1}{2}$ . **Failure sampling.**

Consider the sorted version of  $S$ ,  $S'$ . Logically split  $S'$  in  $k/2$  segments  $(t_1, t_2, \dots, t_{k/2})$  of  $2n/k$  elements each.

# Multi-way QuickSort



- The event is that there exists a bucket  $B_i$  with more than  $4n/k$  items. It spans more than one segment: pivots  $s_{i-1}$  and  $s_i$  fall outside  $t_2$ .
- In  $t_2$  fall less than  $(a+1)$  samples (see selection algorithm: between 2 pivots there are  $a+1$  samples, hence in  $t_2$  there are less).
- $\Pr(\text{exists } B_i : |B_i| \geq 4n/k) \leq \Pr(\text{exists } t_j : \text{contains } < (a+1) \text{ samples})$   
 $\leq k/2 \Pr(\text{a specific segment contains } < (a+1) \text{ samples})$

Since  $k/2$  is the number of segments.

# Multi-way QuickSort

- $\Pr(1 \text{ sample goes in a given segment}) = (2n/k)/n = 2/k$   
If drawn uniformly at random from  $S$  (and  $S'$ ).
- Let  $X$  the number of samples going in a given segment, we want to compute:

$$\Pr(X < a+1)$$

- Observe :  $E(X) = ((a+1)k-1) \times 2/k \geq 2(a+1) - 2/k$ , per  $k \geq 2$   
 $E(X) \geq 2(a+1) - 1 \geq 3/2(a+1)$  for all  $a \geq 1$   
 $a+1 \leq 2/3 E(X) = (1-1/3) E(X)$

By Chernoff bound

$$\Pr(X < (1 - \delta) E(X) \leq e^{\{-\delta^2 / 2\} E(X)})$$

Setting  $\delta=1/3$  and assume  $a+1 = 12 \ln k$

$$\Pr(X < a+1) \leq P(X \leq (1-1/3)E(X)) \leq$$

$$e^{-E(X)/18} \leq e^{-(a+1)/12} = e^{-\ln k} = 1/k$$

# Multi-way QuickSort

- $\Pr (X < a+1) \leq 1/k$

We have already derived:

- $\Pr (\text{exists } B_i : |B_i| \geq 4n/k) \leq k/2 \times \Pr (\text{a segment contains } (a+1) \text{ samples})$
- $\Pr (\text{exists } B_i : |B_i| \geq 4n/k) \leq 1/2$  complement event of the lemma
- All buckets receives less than  $4n/k$  elements with probability  $> 1/2$


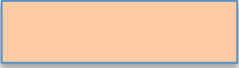

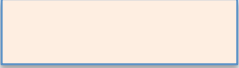


# Dual Pivot QuickSort

- Good strategy in practice no theoretical result.
- Empirical good results in average.



- $p, q$  pivots       $l, k, g$  indices → 4 pieces

1.  items smaller than  $p$
2.  items larger or equal to  $p$  and smaller or equal to  $q$ .
3.  items not yet considered
4.  items greater than  $q$

# Dual Pivot QuickSort

Similar to the 3-ways Partition: maintains the invariants.

- Items equal to the pivot are not treated separately.
- 2 indices move rightward ,  $l$  and  $k$ , while  $g$  moves leftward.
- Termination:  $k \geq g$ .
- For item  $k$ , compare  $S[k] : p$ ,  
if  $S[k] < p$  exchange  $S[k]$  and  $S[l]$  and increment pointers  
  
else if  $S[k] > q$  decrease  $g$  while  $S[g] > q$  and  $g \neq k$   
the last value of  $g : S[g] \leq q$   
exchange  $S[k]$  and  $S[g]$

.....

The comparison with  $S[k]$  drives the phases possibly including a long shift to the left. The nesting of comparison is the key for the efficiency of the algorithm.

# Dual Pivot Partition

$l$   $k$   $g$

5	12	9	12	13	15	17	19	12	26	18	22	20
---	----	---	----	----	----	----	----	----	----	----	----	----

$p=12$   $q=17$   
exchange 19 and 17,  $k++$ ,  $g--$

$l$   $k$   $g$

5	12	9	12	13	15	17	19	12	26	18	22	20
---	----	---	----	----	----	----	----	----	----	----	----	----

$l++$  exchange 12 and 13,  $k++$

$l$   $k$   $g$

5	12	9	12	12	15	17	19	12	13	26	18	22	20
---	----	---	----	----	----	----	----	----	----	----	----	----	----

no exchange

# Dual Pivot QuickSort

You can find the complete code description and the visualization of the algorithm on youtube by searching for Dual pivot QuickSort.

Conclusions:

Even a very old, classic algorithm such as QuickSort can be speed up and innovated!