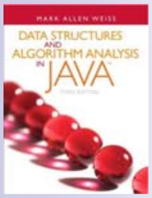
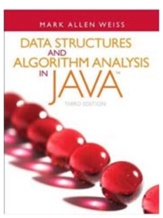


Sorting (IV)

Dr. Antonio L. Bajuelos
FIU School of Computing & Information Sciences



COP-3530 - Data Structures

Module #6: Sorting (part IV)

Outline:

- **D&C sorting algorithms:**
  - **Quick-Sort**
  - **Examples**
  - **Complexity analysis and Java code**

2

Divide and Conquer Methods

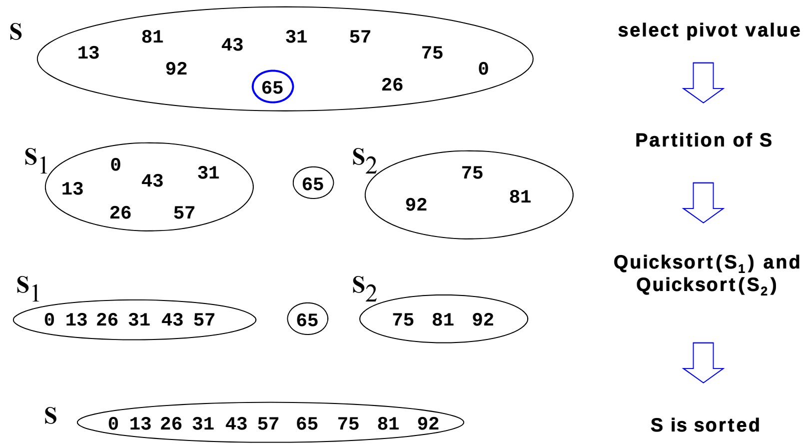
- **Divide and Conquer** is a very important **technique** in **algorithm design**.
- **Main Idea:**
  - **Divide** problem into sub-problems.
  - **Conquer** by solving sub-problems recursively. If the sub-problems are small enough, solve them in brute force approach.
  - **Combine** the solutions of sub-problems into a solution of the original problem.

Quick-Sort

- A **divide-and-conquer** algorithm.
- The classic **quicksort** algorithm to sort an array S consists of the following four steps:
  - If the number of elements in S is 0 or 1, then return.
  - Pick any element **v** in S. This is called the **pivot.**
  - **Partition** S \ {**v**} (the remaining elements in S) into two disjoint groups:
    - $S1 = \{x \in S \setminus \{\mathbf{v}\} \mid x \leq v\}$, and
    - $S2 = \{x \in S \setminus \{\mathbf{v}\} \mid x > v\}$.
  - Return {**quicksort(S1)** ∪ **v** ∪ **quicksort(S2)**}.

Quick-Sort

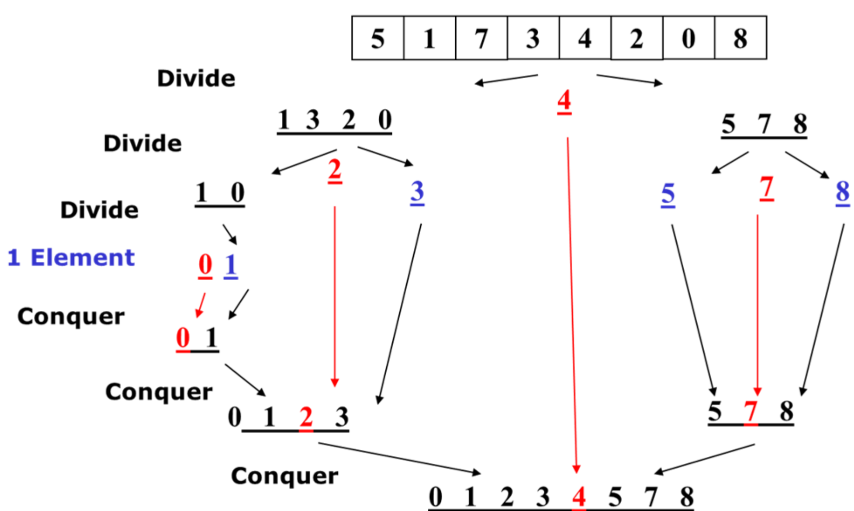
- **Example:**



5

Quick-Sort

- **Example with Recursion:**



6

Quick-Sort. Key steps

- **How to pick a pivot?**
  - **Any choice is correct** i.e. data will end up sorted.
  - But as analysis will show, want the two partitions to be about equal in size.
- **How to partition?**
  - In **linear time.**
  - **In place.**

7

Quick-Sort. Pick a pivot

- Use the **first element** as pivot
  - if the input is random, ok
  - if the input is presorted (or in reverse order)
    - all the elements go into S_2 (or S_1)
    - this happens consistently throughout the recursive calls
    - Results in **$O(n^2)$** behavior!!!
- Example: (4,6,7,8,9,12,15)

8

Quick-Sort. Pick a pivot

How to pick a pivot?

- Choose the **pivot randomly**
  - generally safe
  - random number generation can be expensive
- **Heuristic** that tends to work well
  - While sorting **arr** from **left** to **right-1** we can choose the pivot as the **median of 3**** elements.
  - For example:

    arr[left], arr[right-1], arr[(right+left)/2]

** the **median of 3** numbers is the **middle number** (in a sorted list of 3 numbers).

Quick-Sort. Partitioning Strategy

- After picking **pivot**, need to **partition** in **linear time in place**
- Step for a good partitioning:

```
swap pivot with arr[left]
Use two pointers i and j,
(starting at left+1 and right-1)
while (i < j)
   if (arr[j] > pivot) j--
   else if (arr[i] < pivot) i++
   else swap arr[i] with arr[j]
swap pivot with arr[i]**
```

  **skip this step if pivot ends up being the min element

- Note that this procedure can be adapted to swap the pivot with the arr[right]

Quick-Sort. Partitioning Strategy

- **Example #1**:
  - **Step one**: pick **pivot** as median of 3

    left = 0, right = 10 (arr.length)

0	1	2	3	4	5	6	7	8	9
8	1	4	9	0	3	5	2	7	6

pivot = 6

  - **Step two**: move *pivot* to the left position.

    pointers i = left +1; j = right -1

0	1	2	3	4	5	6	7	8	9
6	1	4	9	0	3	5	2	7	8

i j

```
swap pivot with arr[left]
Use two pointers i and j,
(starting at left+1 and right-1)
while (i < j)
   if (arr[j] > pivot) j--
   else if (arr[i] < pivot) i++
   else swap arr[i] with arr[j]
 swap pivot with arr[i]**
```

11

Quick-Sort. Partitioning Strategy

- **Example #1 (cont…)**:

pivot = 6

```
swap pivot with arr[left]
Use two pointers i and j,
(starting at left+1 and right-1)
while (i < j)
   if (arr[j] > pivot) j--
   else if (arr[i] < pivot) i++
   else swap arr[i] with arr[j]
 swap pivot with arr[i]**
```

partition in place	6	1	4	9	0	3	5	2	7	8
move pointers	6	1	4	9	0	3	5	2	7	8
swap	6	1	4	2	0	3	5	9	7	8
move fingers	6	1	4	2	0	3	5	9	7	8
move pivot	5	1	4	2	0	3	6	9	7	8

12

Quick-Sort. Partitioning Strategy (cont…)

- **Example #2**:

http://www.w3resource.com/c-programming-exercises/searching-and-sorting/c-search-and-sorting-exercise-7.php

13

Quick-Sort. Analysis

- **Best-case:** Pivot is always the median

  T(0)=T(1)=1

  T(n)=2T(n/2) + n **O(n) for the partition step**

  The same recurrence as **Merge-Sort**: **O(n log n)**

- **Worst-case:** Pivot is always smallest or largest element

  T(0)=T(1)=1

  T(n) = T(n-1) + n

  The same recurrence as **Selection sort**: **O(n^2)**

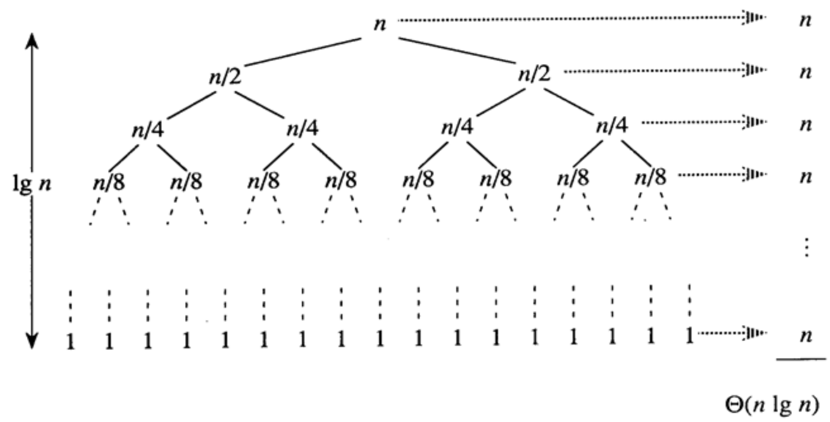
- **Average-case** (e.g., with random pivot)

  O(n log n) (consult the text book for the proof)

14

Quick-Sort. Analysis

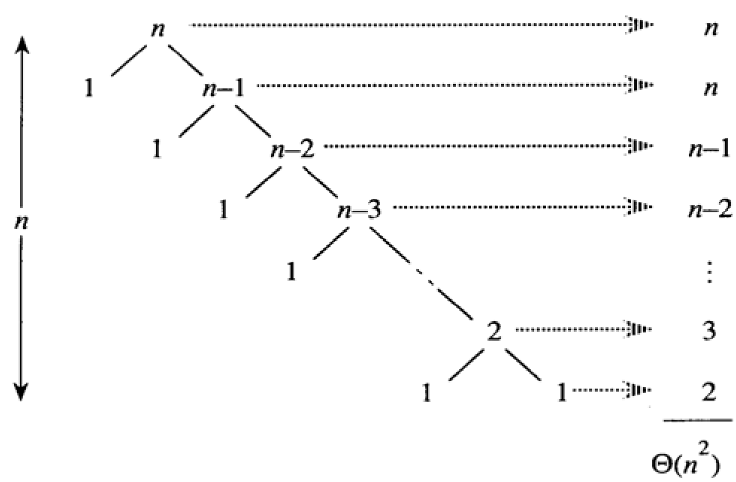
- **Best case:**



15

Quick-Sort. Analysis

- **Worst case:**



16

Quick-Sort. Small Arrays

- For small arrays ($n \leq 20$, for example) **Quicksort** does not perform as well as **insertion sort** but these cases will occur frequently (**QuickSort** is recursive!).
- A common solution: use a sorting algorithm that is efficient for small arrays, such as **Insertion sort**.
- Engineering rule: switch algorithm below a **cutoff** for example use insertion sort for $n < 10$.
- **Cutoff** sample code:

```
private static void quicksort( Comparable [ ] a, int low, int high )
{
        if( low + CUTOFF > high )
           insertionSort( a, low, high );
        else
        {
          ...
        }
}
```

- **Note:** Could also use a cutoff for **Mergesort**

17

Quick-Sort. Java Code

- See **Java Code** of **QuickSort** in:

http://users.cis.fiu.edu/~weiss/dsj2/code/Sort.java

(Prof. Mark Weiss)

Visualization:

http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

https://www.youtube.com/watch?v=ywWBy6J5gz8

Some Facts:

http://algs4.cs.princeton.edu/lectures/23Quicksort.pdf

18