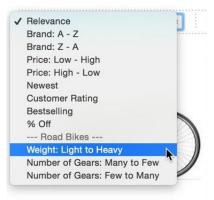
Sorting algorithms

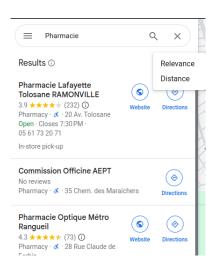
A Ammar, A Scemama, P Reinhardt, Y Damour

November 23, 2024

• Sorting algorithms are used in everyday applications

Sorting algorithms are used in everyday applications





```
:~/.../slides$ ls -lt
41231 Nov 13 23:29 main.log
206249 Nov 13 23:29 main.pdf
  1252 Nov 13 23:29 main.aux
  747 Nov 13 23:29 main.nav
    0 Nov 13 23:29 main.snm
    0 Nov 13 23:29 main.toc
    0 Nov 13 23:29 main.out
 4096 Nov 13 23:29 images
  1873 Nov 13 23:27 intro.tex
  1886 Nov 13 22:25 main.tex
 10290 Nov 13 19:55 quick sort.tex
10834 Nov 13 19:55 merge sort.tex
 5904 Nov 13 19:55 bubble sort.tex
 6337 Nov 13 19:47 radix sort.tex
   261 Nov 10 23:43 Makefīle
  /.../slides$ 🗌
```

Sorting for Efficiency

• Sorting algorithms are employed for more than just sorting



• Find 69 ?

• Find 69 ?

 71
 7
 59
 53
 50
 63
 69
 86
 93
 5
 22

• Find 69 ?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

• Find 69?

71 7 59 53 50 63 69 86 93 5 22

 \Rightarrow time scaling $\mathcal{O}(N)$

5 7 50 22 53 59 63 69 73 86 93

 \Rightarrow time scaling $\mathcal{O}(\log N)$

 $9 \log(116\,000\,000) \approx 18$!!

Example: CI wavefunction

Example: CI wavefunction

| 0 | 0 | 0 | 0 | 1 | 1 | $\rightarrow D_{HF} = 3$ |
|---|---|---|---|---|---|--------------------------|
| 0 | 1 | 0 | 0 | 0 | 1 | $\rightarrow D = 17$ |

Overlap between 2 CI wavefunctions

$$\begin{cases} \Psi = \sum_{I=1}^{N} c_{I} D_{I} \\ \tilde{V} = \sum_{I=1}^{\tilde{N}} \tilde{c}_{I} \tilde{D}_{I} \end{cases} \Rightarrow S = \left\langle \tilde{\Psi} \mid \Psi \right\rangle$$

Example: CI wavefunction

Overlap between 2 CI wavefunctions

$$\begin{cases} \Psi = \sum_{I=1}^{N} c_{I} D_{I} \\ \tilde{\Psi} = \sum_{I=1}^{\tilde{N}} \tilde{c}_{I} \tilde{D}_{I} \end{cases} \Rightarrow S = \left\langle \tilde{\Psi} \mid \Psi \right\rangle$$

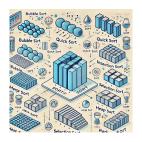
- Naive implementation: $\mathcal{O}(N\tilde{N})$
- Smart implementation: $\mathcal{O}(N \log(N) + \tilde{N} \log(\tilde{N}))$



• Key considerations for sorting algorithms:



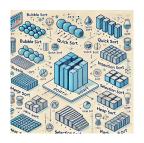
- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...



- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, . . .



- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, . . .
 - Stability: 2 = 2 = 2



- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, . . .
 - Stability: ② ② ≠ ② ②
 - Adaptiveness (best, worst, and average cases)



- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, . . .
 - Stability: ② ② ≠ ② ②
 - Adaptiveness (best, worst, and average cases)
 - Online vs. Offline sorting

1 Bubble Sort

2 Merge sort

3 Quick sort

4 Radix sort

Bubble Sort

Bubble Sort Algorithm

- Goal: Sort an array of *n* items
- Algorithm:
 - 1 Compare first pair of adjacent items
 - 2 Swap if they are in the wrong order

Bubble Sort Algorithm

- Goal: Sort an array of *n* items
- Algorithm:
 - 1 Compare first pair of adjacent items
 - 2 Swap if they are in the wrong order
 - 3 Iterate over next pairs
 - Largest item "bubbles" to the end

Bubble Sort Algorithm

- Goal: Sort an array of *n* items
- Algorithm:
 - 1 Compare first pair of adjacent items
 - 2 Swap if they are in the wrong order
 - 3 Iterate over next pairs
 - Largest item "bubbles" to the end
 - 4 Reduce n by 1 and go to step 1

First pass

25 | 13 | 4 | 7 | 16

First pass

25 13 4 7 16

| 25 | 13 | 4 | 7 | 16 |
|----|----|---|---|----|
| | | | | |
| | | | | |

| 25 | 13 | 4 | 7 | 16 |
|----|----|---|---|----|
| | | | | |
| 10 | 5 | | 7 | 10 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|---|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|---|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |
| 13 | 4 | 7 | 16 | 25 |

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |
| 13 | 4 | 7 | 16 | 25 |

First pass

Second pass

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|-----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |
| 12 | 1 | 7 | 16 | OF. |

| 13 | 4 | 7 | 16 | 25 |
|----|---|---|----|----|

First pass

Second pass

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |

16 25

| 13 | 4 | 7 | 16 | 25 |
|----|----|---|----|----|
| 4 | 13 | 7 | 16 | 25 |

First pass

Second pass

| 25 | 13 | 4 | 7 | 16 | | |
|----|----|----|----|----|--|--|
| 13 | 25 | 4 | 7 | 16 | | |
| 13 | 4 | 25 | 7 | 16 | | |
| 13 | 4 | 7 | 25 | 16 | | |

16 25

| 13 | 4 | 7 | 16 | 25 |
|----|----|----|----|----|
| 4 | 13 | 7 | 16 | 25 |
| 4 | 7 | 13 | 16 | 25 |

First pass

Second pass

| 25 | 13 | 4 | 7 | 16 | | |
|----|----|----|----|----|--|--|
| 13 | 25 | 4 | 7 | 16 | | |
| 13 | 4 | 25 | 7 | 16 | | |
| 13 | 4 | 7 | 25 | 16 | | |
| 13 | 4 | 7 | 16 | 25 | | |

| 13 | 4 | 7 | 16 | 25 |
|----|----|----|----|----|
| 4 | 13 | 7 | 16 | 25 |
| 4 | 7 | 13 | 16 | 25 |
| 4 | 7 | 13 | 16 | 25 |

Data: Array A of n elements

Result: Sorted array A

Data: Array A of n elements

Result: Sorted array A

loop over passes

Data: Array A of n elements

Result: Sorted array A

loop over passes

for i = 0 to n - 2 do

```
Data: Array A of n elements

Result: Sorted array A

# loop over passes

for i = 0 to n - 2 do

| # compares adjacent elements
```

```
Data: Array A of n elements

Result: Sorted array A

# loop over passes

for i = 0 to n - 2 do

# compares adjacent elements

for j = 0 to n - 2 - i do
```

```
Data: Array A of n elements

Result: Sorted array A

# loop over passes

for i = 0 to n - 2 do

# compares adjacent elements

for j = 0 to n - 2 - i do

# bubble largest element
```

```
Data: Array A of n elements

Result: Sorted array A

# loop over passes

for i = 0 to n - 2 do

# compares adjacent elements

for j = 0 to n - 2 - i do

# bubble largest element

if A[j] > A[j+1] then

| Swap A[j] and A[j+1];
```

First pass

Second pass

| 25 | 13 | 4 | 7 | 16 |
|----|----|----|----|----|
| 13 | 25 | 4 | 7 | 16 |
| 13 | 4 | 25 | 7 | 16 |
| 13 | 4 | 7 | 25 | 16 |
| 13 | 4 | 7 | 16 | 25 |

| 13 | 4 | 7 | 16 | 25 |
|----|----|----|----|----|
| 4 | 13 | 7 | 16 | 25 |
| 4 | 7 | 13 | 16 | 25 |
| 4 | 7 | 13 | 16 | 25 |

| | Fire | st p | ass | | S | eco | nd | pass | 5 | Third pass | | | | |
|----|------|------|-----|----|----|-----|----|------|----|------------|---|----|----|----|
| 25 | 13 | 4 | 7 | 16 | 13 | 4 | 7 | 16 | 25 | 4 | 7 | 13 | 16 | 25 |
| 13 | 25 | 4 | 7 | 16 | 4 | 13 | 7 | 16 | 25 | | | | | |
| 13 | 4 | 25 | 7 | 16 | 4 | 7 | 13 | 16 | 25 | | | | | |
| 13 | 4 | 7 | 25 | 16 | 4 | 7 | 13 | 16 | 25 | | | | | |
| 13 | 4 | 7 | 16 | 25 | | | | | | | | | | |

| | Fir | S | eco | nd | pas | 5 | Third pas | | | | | | | |
|---|------|----|-----|----|-----|----|-----------|----|----|----|---|---|----|----|
| 2 | 5 13 | 4 | 7 | 16 | | 13 | 4 | 7 | 16 | 25 | 4 | 7 | 13 | 16 |
| 1 | 3 25 | 4 | 7 | 16 | | 4 | 13 | 7 | 16 | 25 | 4 | 7 | 13 | 16 |
| 1 | 3 4 | 25 | 7 | 16 | | 4 | 7 | 13 | 16 | 25 | | | | |
| 1 | 3 4 | 7 | 25 | 16 | | 4 | 7 | 13 | 16 | 25 | | | | |
| 1 | 3 4 | 7 | 16 | 25 | | | | | | | | | | |

| First pass | | | | | S | eco | nd | pass | 6 | Third pass | | | | | |
|------------|----|----|----|----|---|-----|----|------|----|------------|---|---|----|----|----|
| 25 | 13 | 4 | 7 | 16 | | 13 | 4 | 7 | 16 | 25 | 4 | 7 | 13 | 16 | 25 |
| 13 | 25 | 4 | 7 | 16 | | 4 | 13 | 7 | 16 | 25 | 4 | 7 | 13 | 16 | 25 |
| 13 | 4 | 25 | 7 | 16 | | 4 | 7 | 13 | 16 | 25 | 4 | 7 | 13 | 16 | 25 |
| 13 | 4 | 7 | 25 | 16 | | 4 | 7 | 13 | 16 | 25 | | | | | |
| 13 | 4 | 7 | 16 | 25 | | | | | | | | | | | |

Improved implementation

```
Data: Array A of n elements
Result: Sorted array A
for i = 0 to n - 2 do
   is_sorted = true:
   for i = 0 to n - 2 - i do
       if A[j] > A[j + 1] then
          Swap A[j] and A[j+1];
          is\_sorted = false;
   if is sorted then
       return;
```

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (i):
 - for i = 0: n 1 iterations

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - •
 - for i = n 2: 1 iteration

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - - for i = n 2: 1 iteration
 - Total number of iterations: $1 + 2 + \cdots + n 1 = \frac{n(n-1)}{2}$

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - •
 - for i = n 2: 1 iteration
 - Total number of iterations: $1 + 2 + \cdots + n 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n-1))$

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - •
 - for i = n 2: 1 iteration
 - Total number of iterations: $1 + 2 + \cdots + n 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n-1)) = \mathcal{O}(n^2 n)$

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - •
 - for i = n 2: 1 iteration
 - Total number of iterations: $1 + 2 + \cdots + n 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n-1)) = \mathcal{O}(n^2 n) = \mathcal{O}(n^2)$

- Time complexity:
 - Outer loop (i): 0, 1, ..., n-2
 - Inner loop (j):
 - for i = 0: n 1 iterations
 - for i = 1: n 2 iterations
 - . . .
 - for i = n 2: 1 iteration
 - Total number of iterations: $1 + 2 + \cdots + n 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n-1)) = \mathcal{O}(n^2 n) = \mathcal{O}(n^2)$
- We need a constant amount of memory: $\mathcal{O}(1)$

Merge sort

- Merge Sort is a divide-and-conquer algorithm
 - 1 Divide: Recursively divide the array into two halves
 - 2 Conquer: Merge the two halves

- Merge Sort is a divide-and-conquer algorithm
 - 1 Divide: Recursively divide the array into two halves
 - 2 Conquer: Merge the two halves
- Suppose time scaling is $\mathcal{O}(n^2)$
 - for $n: \rightarrow 1$ day
 - for n/2: $\rightarrow 1/4$ day

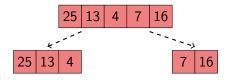
- Merge Sort is a divide-and-conquer algorithm
 - 1 Divide: Recursively divide the array into two halves
 - 2 Conquer: Merge the two halves
- Suppose time scaling is $\mathcal{O}(n^2)$
 - for $n: \rightarrow 1$ day
 - for n/2: $\rightarrow 1/4$ day
- Analysis (homework 6):
 - Time Complexity: $\mathcal{O}(n \log n)$
 - Space Complexity: $\mathcal{O}(n)$

Illustration

25 | 13 | 4 | 7 | 16

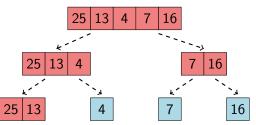
Divide

Illustration

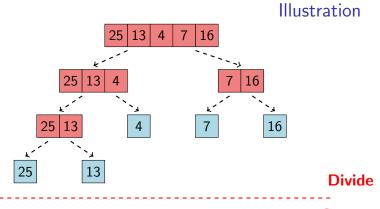


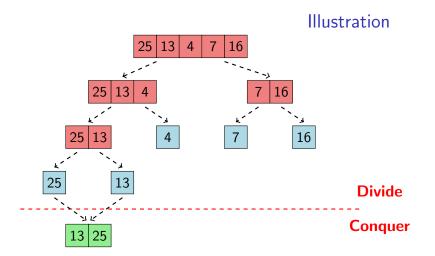
Divide

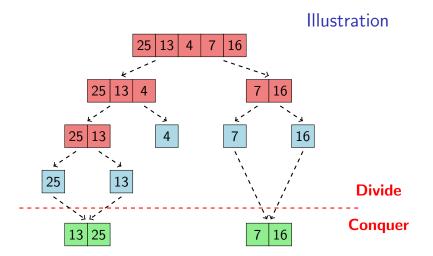
Illustration

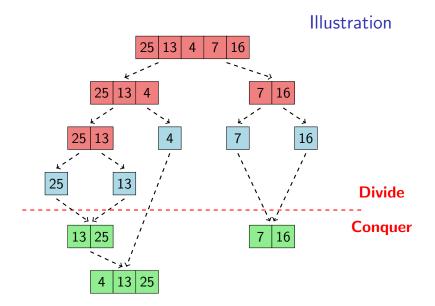


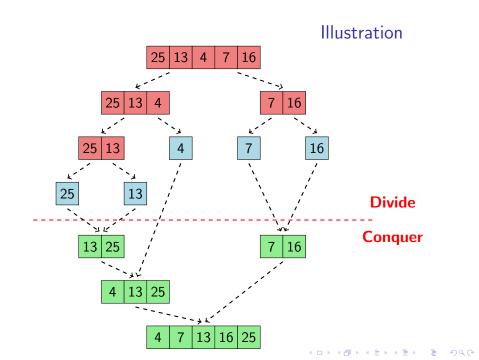
Divide











Data: Array A, left and right indices

Result: Sorted array A

Data: Array A, left and right indices

Result: Sorted array A

to stop recursion

if left < right then</pre>

```
Data: Array A, left and right indices Result: Sorted array A
# to stop recursion
if left < right then

# middle index

mid = left + \left\lfloor \frac{\text{right-left}}{2} \right\rfloor;
```

```
Data: Array A, left and right indices Result: Sorted array A # to stop recursion if left < right then | # middle index | mid = left + \left\lfloor \frac{\text{right-left}}{2} \right\rfloor; # recursively sort the two halves MergeSort(A, left, mid); MergeSort(A, mid+1, right);
```

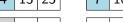
```
Data: Array A, left and right indices
Result: Sorted array A
# to stop recursion
if left < right then
   # middle index
   mid = left + \left| \frac{right - left}{2} \right|;
   # recursively sort the two halves
   MergeSort(A, left, mid);
   MergeSort(A, mid+1, right);
   # merge the two halves
   merge(A, left, mid, right);
```



4 13 25 7 16

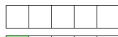


16



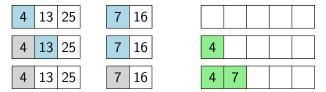


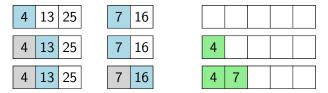
7 16

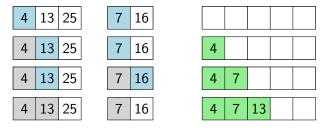


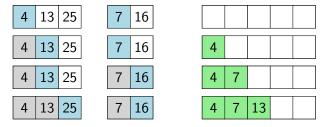
4 13 25

7 16









| 4 | 13 25 | 7 16 | | | | | |
|---|-------|------|---|---|----|----|--|
| 4 | 13 25 | 7 16 | 4 | | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | 16 | |

| 4 | 13 25 | 7 16 | | | | | |
|---|-------|------|---|---|----|----|--|
| 4 | 13 25 | 7 16 | 4 | | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | 16 | |

| 4 | 13 25 | 7 16 | | | | | |
|---|-------|------|---|---|----|----|----|
| 4 | 13 25 | 7 16 | 4 | | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | 16 | |
| 4 | 13 25 | 7 16 | 4 | 7 | 13 | 16 | 25 |

Quick sort

- Quick Sort is a divide-and-conquer algorithm
 - 1 Divide Step: choose a "pivot"
 - the pivot divides the array into elements < p and those $\ge p$
 - 2 Conquer Step: Do nothing!

- Quick Sort is a divide-and-conquer algorithm
 - 1 Divide Step: choose a "pivot"
 - the pivot divides the array into elements < p and those $\ge p$
 - 2 Conquer Step: Do nothing!
- Merge Sort spends more time in the conquer step, while Quick Sort spends more time in the divide step

- Quick Sort is a divide-and-conquer algorithm
 - 1 Divide Step: choose a "pivot"
 - the pivot divides the array into elements < p and those $\ge p$
 - 2 Conquer Step: Do nothing!
- Merge Sort spends more time in the conquer step, while Quick Sort spends more time in the divide step
- Analysis (homework 6):
 - Time Complexity: $\mathcal{O}(n \log n)$
 - Space Complexity: $\mathcal{O}(\log n)$

Data: Array A, left and right indices

Result: Sorted array A

Data: Array A, left and right indices

Result: Sorted array A

to stop recursion

if left < right then</pre>

```
Data: Array A, left and right indices

Result: Sorted array A

# to stop recursion

if left < right then

# partition the array

piv_index = partition(A, left, right);

# recursively sort the two halves

QuickSort(A, left, piv_index - 1);

QuickSort(A, piv_index + 1, right);
```

| 10 61 46 56 1 35 | 51 55 2 | 2 |
|------------------|---------|---|
|------------------|---------|---|

| 10 61 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|----------|----|---|----|----|----|----|
|----------|----|---|----|----|----|----|

| 10 61 46 56 | 35 | 51 55 | 22 |
|-------------|----|-------|----|
|-------------|----|-------|----|

| 1 | l | l . | | l | |
|---|---|-----|--|---|--|
| 1 | l | l . | | l | |
| 1 | l | l | | l | |
| | | | | | |

| 10 61 46 56 1 | 35 51 55 22 |
|---------------|-------------|
|---------------|-------------|

| 10 61 46 | 56 1 | 35 51 | 55 | 22 |
|----------|------|-------|----|----|
|----------|------|-------|----|----|

| 10 61 46 5 | 5 1 35 | 51 55 | 22 |
|------------|--------|-------|----|
|------------|--------|-------|----|

| 10 |
|----|
|----|

| 10 61 46 56 | 1 | 35 | 51 | 55 | 22 |
|-------------|---|----|----|----|----|
|-------------|---|----|----|----|----|

| 10 | |
|----|--|
|----|--|



| 10 1 |
|------|
|------|

| 10 61 46 56 | 1 | 35 | 51 | 55 | 22 |
|-------------|---|----|----|----|----|
|-------------|---|----|----|----|----|

| 10 61 46 56 1 | 35 51 | 55 22 | 2 |
|---------------|-------|-------|---|
|---------------|-------|-------|---|

| 10 1 | 61 | |
|------|----|--|
|------|----|--|

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|----|----|----|----|---|----|----|----|----|
|----|----|----|----|---|----|----|----|----|

| 10 1 61 |
|---------|
|---------|

| 10 61 46 56 1 | 35 51 55 22 |
|---------------|-------------|
|---------------|-------------|

| 10 | 1 | | 61 | | | | |
|----|---|--|----|--|--|--|--|
|----|---|--|----|--|--|--|--|

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 | |
|-------|----|----|----|----|----|----|----|----|--|
| pivot | | | | | | | | | |
| 10 | 1 | 22 | | 61 | | | | | |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 | |
|-------|----|----|----|----|----|----|----|----|--|
| pivot | | | | | | | | | |
| 10 | 1 | 22 | | 61 | | | | 46 | |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 | |
|-------|----|----|----|----|----|----|----|----|--|
| pivot | | | | | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 | |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|----|----|------|----|----|----|----|----|----|
| | F | oivo | t | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|----|----|------|----|----|----|----|----|----|
| | þ | ovio | t | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|------|----|------|----|------|----|----|----|----|
| | þ | oivo | t | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| oivo | t | | F | oivo | t | | | |
| 1 | | 22 | | 46 | | | | |

| | 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|---|-----|----|------|----|------|----|----|----|----|
| | | þ | oivo | t | | | | | |
| | 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| p | ivo | t | | F | oivo | t | | | |
| | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |

| | 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|---|-----|----|------|----|------|----|----|----|----|
| | | þ | oivo | t | | | | | |
| | 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| p | ivo | t | | F | oivo | t | | | |
| | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|------|----|------|----|------|----|----|----|----|
| | þ | oivo | t | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| ovic | t | | F | ovio | t | | | |
| 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |

| | 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|---|------|----|------|----|------|----|----|----|-------|
| | | þ | ovio | t | | | | | |
| | 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| p | oivo | t | | þ | oivo | t | | | |
| | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| | | | | | | | | þ | oivot |
| | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |

| 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
|------|-------------------|-----------------------|-----------------------------|--------------------------|---|---|--|------|
| | þ | ovio | t | | | | | |
| 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| ovio | t | | þ | ovio | t | | | |
| 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| | | | | | | | þ | ovio |
| 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| | 10 pivo | 10 1 bivot 1 10 | pivo 10 1 22 iivot 1 10 22 | pivot 10 1 22 56 iivot | pivot 10 1 22 56 61 pivot pivo 1 10 22 35 46 | pivot 35 56 61 35 56 61 10 10 22 35 46 56 61 61 61 61 61 61 6 | pivot 10 1 22 56 61 35 51 pivot pivot 1 10 22 35 46 56 51 | 10 |

| 61 | | | | | | | |
|------|-------|-------------|------------------------|--|----------------------------------|---|--|
| , 01 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
| ı | oivo | t | | | | | |
| 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| ot | | F | oivo | t | | | |
| 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| • | | • | | • | | þ | oivo |
| 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| | | | | þ | oivo | t | |
| 10 | 22 | 35 | 46 | 51 | 55 | 56 | 61 |
| | 10 10 | 10 22 10 22 | ot p 10 22 35 10 22 35 | 1 22 56 61 pivo 10 22 35 46 10 22 35 46 | 10 22 35 46 56 10 22 35 46 56 | 10 22 35 46 56 51 10 22 35 46 56 51 10 22 35 46 56 51 pivo | pivot 1 22 35 46 56 51 55 10 22 35 46 56 51 55 |

| 10 61 46 56 1 35 51 55 22 pivot 10 1 22 56 61 35 51 55 46 pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 51 55 56 61 1 10 22 35 46 51 55 56 61 | | | | | | | | | | |
|---|---|-----|----|------|----|------|----|------|----|------|
| 10 | | 10 | 61 | 46 | 56 | 1 | 35 | 51 | 55 | 22 |
| pivot pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 56 51 55 61 pivot | | | F | oivo | t | | | | | |
| 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 51 55 56 61 | | 10 | 1 | 22 | 56 | 61 | 35 | 51 | 55 | 46 |
| pivot 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 51 55 56 61 | ם | ivo | t | | þ | oivo | t | | | |
| 1 10 22 35 46 56 51 55 61 pivot 1 10 22 35 46 51 55 56 61 | | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| pivot 1 10 22 35 46 51 55 56 61 | | | | | | | | | þ | oivo |
| 1 10 22 35 46 51 55 56 61 | | 1 | 10 | 22 | 35 | 46 | 56 | 51 | 55 | 61 |
| | | | | | | | þ | oivo | t | |
| 1 10 22 35 46 51 55 56 61 | | 1 | 10 | 22 | 35 | 46 | 51 | 55 | 56 | 61 |
| 1 10 22 35 46 51 55 56 61 | | | | | | | | | | |
| | | 1 | 10 | 22 | 35 | 46 | 51 | 55 | 56 | 61 |

Radix sort

Idea

- Radix (root): base in which we express an integer
 - Radix 10, Radix 2, ...
 - from right(LSD \rightarrow MSD), from left (MSD \rightarrow LSD)

Idea

- Radix (root): base in which we express an integer
 - Radix 10, Radix 2, ...
 - from right(LSD \rightarrow MSD), from left (MSD \rightarrow LSD)
- Counting Sort:
 - Non-comparative sorting (no direct use of <, >)
 - treats data as a "character" string (digit, bit, ...)

Idea

- Radix (root): base in which we express an integer
 - Radix 10, Radix 2, ...
 - from right(LSD \rightarrow MSD), from left (MSD \rightarrow LSD)
- Counting Sort:
 - Non-comparative sorting (no direct use of <, >)
 - treats data as a "character" string (digit, bit, ...)
- Analysis (homework 6):
 - Time Complexity: O(nk), (k: # of bits of largest number)
 - Space Complexity: $\mathcal{O}(n+k)$, $\mathcal{O}(nk)$, ...

| 3 | 6 | 7 | 8 | 2 |
|---|---|---|---|---|
| | | | | |

| • • • = |
|---------------|
|---------------|

3 6 7 8 2

00110110011110000010

0011**0**110**0**111**0**010

3 6 7 8 2

0011 0110 0111 1000 0010

 $\mathbf{0}011 \, \mathbf{0}110 \, \mathbf{0}111 \, \mathbf{0}010$

1000

0**0**110**0**10

3 6 7 8 2

0011 0110 0111 1000 0010

 $\mathbf{0}011\mathbf{0}110\mathbf{0}111\mathbf{0}010$

1000

0**0**110**0**10

0**1**10<mark>01</mark>11

3 6 7 8 2

011**0**110**0**111**0**010

000

0110**0**10

110**01**11

0

3 6 7 8 2

011**0**110**0**111**0**010

000

0110**0**10

110**01**11

0

1

0

3 6 7 8 2

 $\mathbf{0}011\mathbf{0}110\mathbf{0}111\mathbf{0}010$

000

0110**0**10

110**01**11

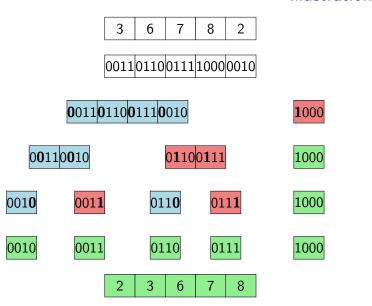
0

1

0

1

| | 3 | 6 | 7 | 8 | 2 | | |
|-----------------------------|------|----------------------------------|------------|------|------------|------|--|
| | 001 | 10110 | 0111 | 1000 | 0010 | | |
| | 1000 | | | | | | |
| 0 0 11 0 0 10 | | 0 1 10 <mark>01</mark> 11 | | | | 1000 | |
| 001 0 | 0011 | 01 | 1 0 | 01 | 1 1 | 1000 | |
| 0010 | 0011 | 01 | .10 | 01 | 11 | 1000 | |



Ref

- Introduction to Algorithms. T. Cormen, C. Leiserson, R. Rivest, C. Stein
- The Art of Computer Programming, Vol. 3. D. Knuth
- for fun
 - https://www.toptal.com/developers/sorting-algorithms
 - https://www.youtube.com/watch?v=kPRAOW1kECg