

Sorting algorithms

A Ammar, A Scemama, P Reinhardt, Y Damour

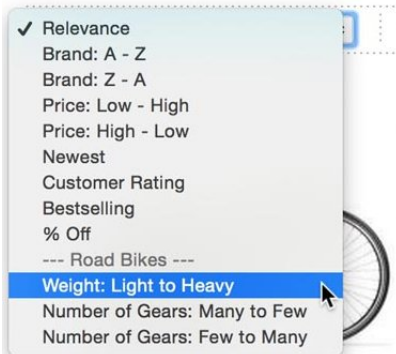
November 21, 2024

Sorting for Sorting

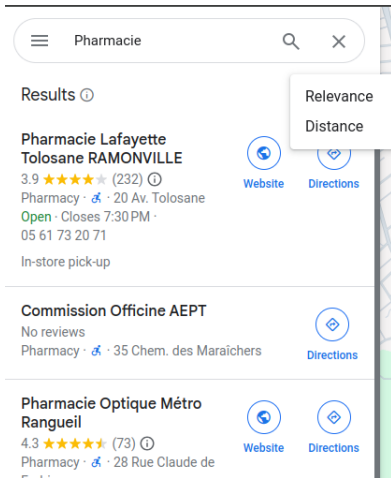
- Sorting algorithms are used in everyday applications

Sorting for Sorting

- Sorting algorithms are used in everyday applications



Sorting for Sorting

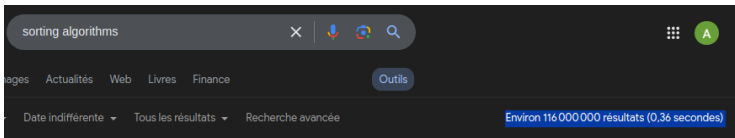


Sorting for Sorting

```
~/.../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 Makefile
~/.../slides$
```

Sorting for Efficiency

- Sorting algorithms are employed for more than just sorting



Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ time scaling $\mathcal{O}(N)$

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ **time scaling** $\mathcal{O}(N)$

5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ **time scaling** $\mathcal{O}(N)$

5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ **time scaling** $\mathcal{O}(N)$

5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ time scaling $\mathcal{O}(N)$

5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?

71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ time scaling $\mathcal{O}(N)$

5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?

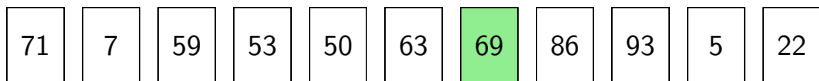
71	7	59	53	50	63	69	86	93	5	22
----	---	----	----	----	----	----	----	----	---	----

➡ time scaling $\mathcal{O}(N)$

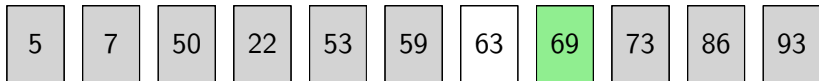
5	7	50	22	53	59	63	69	73	86	93
---	---	----	----	----	----	----	----	----	----	----

Linear vs Binary search

- Find 69 ?



➡ time scaling $\mathcal{O}(N)$



➡ time scaling $\mathcal{O}(\log N)$

🤖 $\log(116\,000\,000) \approx 18$!!

Example: CI wavefunction

0	0	0	0	1	1
---	---	---	---	---	---

 $\rightarrow D_{\text{HF}} = 3$

0	1	0	0	0	1
---	---	---	---	---	---

 $\rightarrow D = 17$

Example: CI wavefunction

0	0	0	0	1	1
---	---	---	---	---	---

 $\rightarrow D_{\text{HF}} = 3$

0	1	0	0	0	1
---	---	---	---	---	---

 $\rightarrow D = 17$

- Overlap between 2 CI wavefunctions

$$\left\{ \begin{array}{l} \Psi = \sum_{I=1}^N c_I D_I \\ \tilde{\Psi} = \sum_{I=1}^{\tilde{N}} \tilde{c}_I \tilde{D}_I \end{array} \right. \Rightarrow S = \langle \tilde{\Psi} | \Psi \rangle$$

Example: CI wavefunction

0	0	0	0	1	1
---	---	---	---	---	---

 $\rightarrow D_{\text{HF}} = 3$

0	1	0	0	0	1
---	---	---	---	---	---

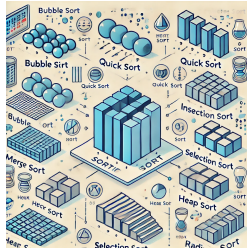
 $\rightarrow D = 17$

- Overlap between 2 CI wavefunctions

$$\begin{cases} \psi = \sum_{I=1}^N c_I D_I \\ \tilde{\psi} = \sum_{I=1}^{\tilde{N}} \tilde{c}_I \tilde{D}_I \end{cases} \Rightarrow S = \langle \tilde{\psi} | \psi \rangle$$

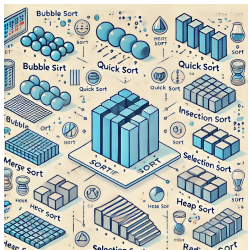
- Naive implementation: $\mathcal{O}(N\tilde{N})$
- Smart implementation: $\mathcal{O}(N \log(N) + \tilde{N} \log(\tilde{N}))$

Best Sorting Algorithm ?



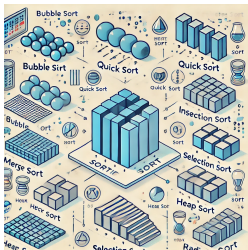
- Key considerations for sorting algorithms:

Best Sorting Algorithm ?



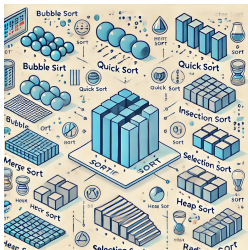
- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...

Best Sorting Algorithm ?



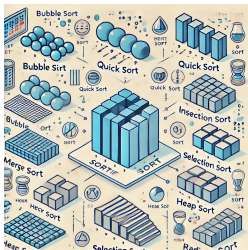
- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, ...

Best Sorting Algorithm ?



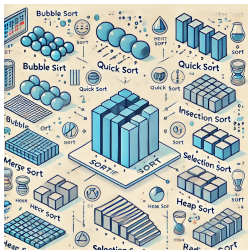
- Key considerations for sorting algorithms:
 - Time complexity: $\mathcal{O}(N^2)$, $\mathcal{O}(N \log N)$, $\mathcal{O}(kN)$, ...
 - Space complexity (memory usage): $\mathcal{O}(1)$, $\mathcal{O}(\log N)$, $\mathcal{O}(N)$, ...
 - Stability: ❷ ❷ \neq ❷ ❷

Best Sorting Algorithm ?



- 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: ❷ ❷ \neq ❷ ❷
 - Adaptiveness (best, worst, and average cases)

Best Sorting Algorithm ?



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

Illustration

First pass

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

Illustration

First pass

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

Illustration

First pass

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

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

Illustration

First pass

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

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

Illustration

First pass

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

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

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

Illustration

First pass

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

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

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

Illustration

First pass

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

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

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

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

Illustration

First pass

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

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

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

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

Illustration

First 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
----	---	---	----	----

Illustration

First 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
----	---	---	----	----

Illustration

First 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
----	---	---	----	----

Second pass

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

Illustration

First 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
----	---	---	----	----

Second pass

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

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

Illustration

First 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
----	---	---	----	----

Second pass

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

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

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

Illustration

First 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
----	---	---	----	----

Second pass

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

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

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

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

Naive implementation

Data: Array A of n elements

Result: Sorted array A

Naive implementation

Data: Array A of n elements

Result: Sorted array A

loop over passes

Naive implementation

Data: Array A of n elements

Result: Sorted array A

loop over passes

for $i = 0$ to $n - 2$ do

|

Naive implementation

Data: Array A of n elements

Result: Sorted array A

loop over passes

for $i = 0$ **to** $n - 2$ **do**

| # compares adjacent elements

Naive implementation

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**

Naive implementation

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

Naive implementation

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 $A[j + 1]$;

Illustration

First 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

Second pass

13	4	7	16	25
4	13	7	16	25
4	7	13	16	25
4	7	13	16	25

Illustration

First 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

Second pass

13	4	7	16	25
4	13	7	16	25
4	7	13	16	25
4	7	13	16	25

Third pass

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

Illustration

First 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

Second pass

13	4	7	16	25
4	13	7	16	25
4	7	13	16	25
4	7	13	16	25

Third pass

4	7	13	16	25
4	7	13	16	25

Illustration

First 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

Second pass

13	4	7	16	25
4	13	7	16	25
4	7	13	16	25
4	7	13	16	25

Third pass

4	7	13	16	25
4	7	13	16	25
4	7	13	16	25

Improved implementation

Data: Array A of n elements

Result: Sorted array A

for $i = 0$ **to** $n - 2$ **do**

$\text{is_sorted} = \text{true};$

for $j = 0$ **to** $n - 2 - i$ **do**

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

 Swap $A[j]$ and $A[j + 1];$

$\text{is_sorted} = \text{false};$

if is_sorted **then**

return;

Analysis

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

Analysis

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

Analysis

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

Analysis

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

Analysis

- Time complexity:
 - Outer loop (i): $0, 1, \dots, n - 2$
 - 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: $1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$

Analysis

- Time complexity:
 - Outer loop (i): $0, 1, \dots, n - 2$
 - 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: $1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n - 1))$

Analysis

- Time complexity:
 - Outer loop (i): $0, 1, \dots, n - 2$
 - 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: $1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n - 1)) = \mathcal{O}(n^2 - n)$

Analysis

- Time complexity:
 - Outer loop (i): $0, 1, \dots, n - 2$
 - 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: $1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$
- Total time: $\mathcal{O}(c n(n - 1)) = \mathcal{O}(n^2 - n) = \mathcal{O}(n^2)$

Analysis

- Time complexity:
 - Outer loop (i): $0, 1, \dots, n - 2$
 - 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: $1 + 2 + \dots + n - 1 = \frac{n(n-1)}{2}$
- 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

Idea

- 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

Idea

- 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 🤔):
 - Time Complexity: $\mathcal{O}(n \log n)$
 - Space Complexity: $\mathcal{O}(n)$

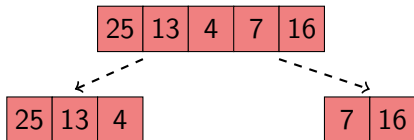
Illustration

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

Divide

Conquer

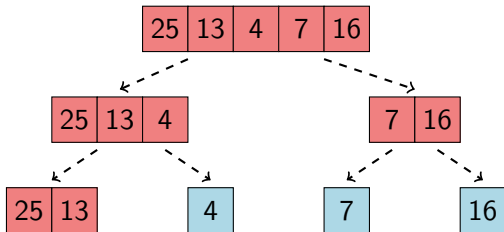
Illustration



Divide

Conquer

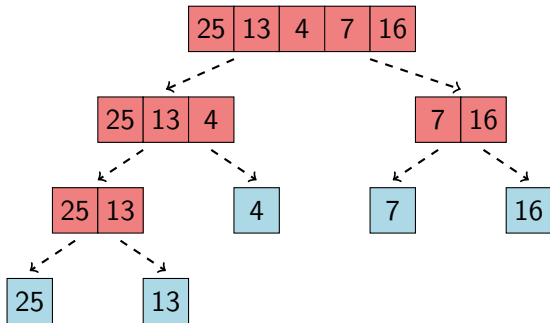
Illustration



Divide

Conquer

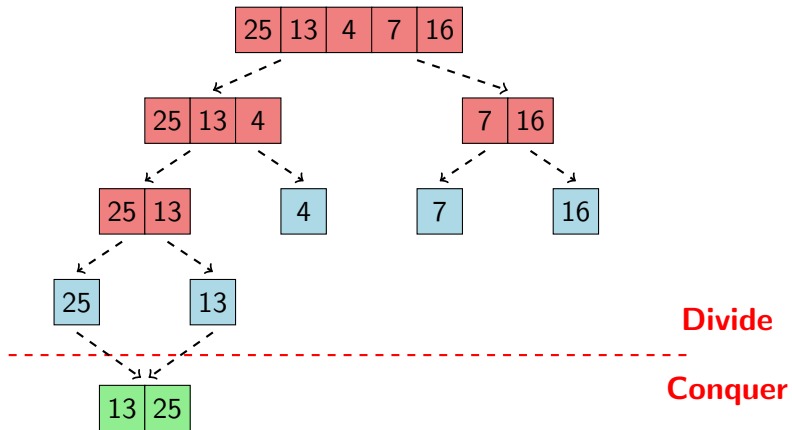
Illustration



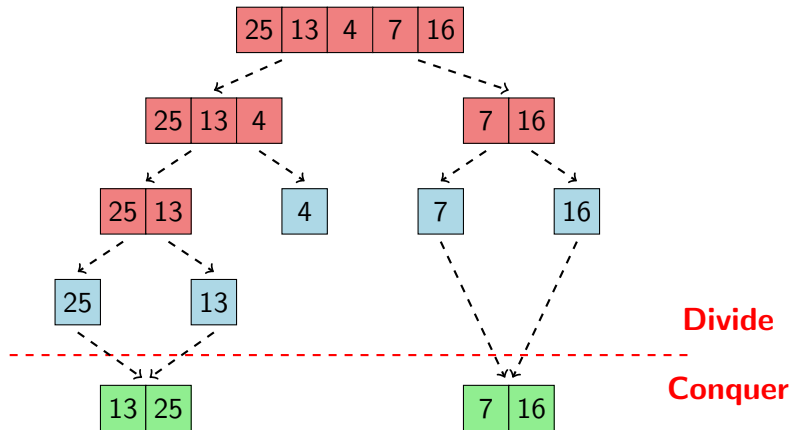
Divide

Conquer

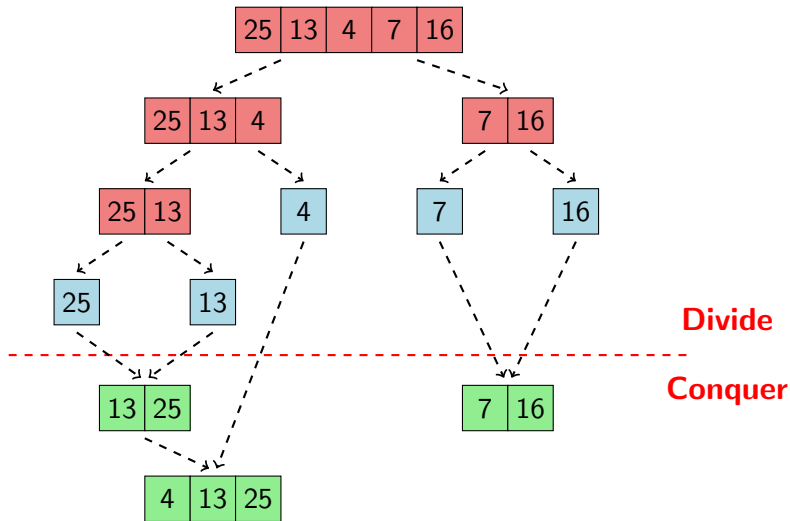
Illustration



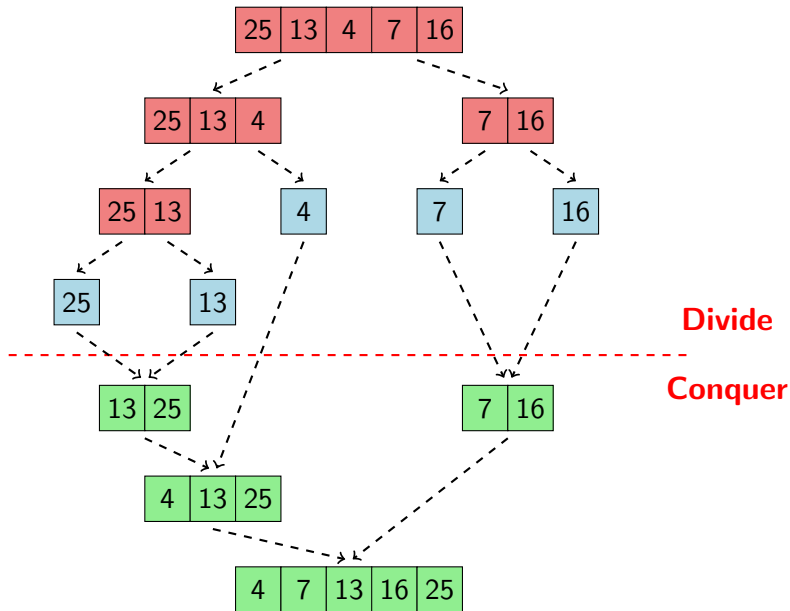
Illustration



Illustration



Illustration



MergeSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

MergeSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

to stop recursion

if $\text{left} < \text{right}$ **then**

|

MergeSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

to stop recursion

if left < right **then**

 # middle index

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

MergeSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

to stop recursion

if left < right **then**

middle index

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

recursively sort the two halves

 MergeSort(A , left, mid);

 MergeSort(A , mid+1, right);

MergeSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

to stop recursion

if left < right **then**

middle index

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

recursively sort the two halves

 MergeSort(A , left, mid);

 MergeSort(A , mid+1, right);

merge the two halves

 merge(A , left, mid, right);

Merge step

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

Merge step

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

Merge step

4	13	25
---	----	----

4	13	25
---	----	----

7	16
---	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

Merge step

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

4	13	25
---	----	----

7	16
---	----

4				
---	--	--	--	--

Merge step

4	13	25
---	----	----

4	13	25
---	----	----

4	13	25
---	----	----

7	16
---	----

7	16
---	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

4	7			
---	---	--	--	--

Merge step

4	13	25
---	----	----

4	13	25
---	----	----

4	13	25
---	----	----

7	16
---	----

7	16
---	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

4	7			
---	---	--	--	--

Merge step

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

4	7			
---	---	--	--	--

4	7	13		
---	---	----	--	--

Merge step

4	13	25
---	----	----

4	13	25
---	----	----

4	13	25
---	----	----

4	13	25
---	----	----

7	16
---	----

7	16
---	----

7	16
---	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

4	7			
---	---	--	--	--

4	7	13		
---	---	----	--	--

Merge step

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

4				
---	--	--	--	--

4	7			
---	---	--	--	--

4	7	13		
---	---	----	--	--

4	7	13	16	
---	---	----	----	--

Merge step

4	13	25
---	----	----

7	16
---	----

--	--	--	--	--

4	13	25
---	----	----

7	16
---	----

4				
---	--	--	--	--

4	13	25
---	----	----

7	16
---	----

4	7			
---	---	--	--	--

4	13	25
---	----	----

7	16
---	----

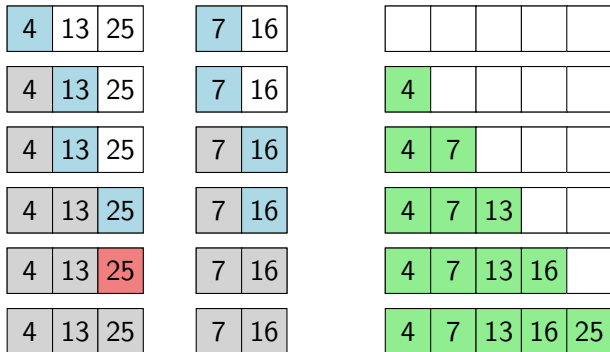
4	7	13		
---	---	----	--	--

4	13	25
---	----	----

7	16
---	----

4	7	13	16	
---	---	----	----	--

Merge step



Quick sort

Idea

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

Idea

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

Idea

- Quick Sort is a divide-and-conquer algorithm
 - ① Divide Step: choose a “pivot”
 - the pivot divides the array into elements $< p$ and those $\geq p$
 - 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 🤔):
 - Time Complexity: $\mathcal{O}(n \log n)$
 - Space Complexity: $\mathcal{O}(\log n)$

QuickSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

QuickSort Algorithm

Data: Array A , left and right indices

Result: Sorted array A

to stop recursion

if left < right **then**

|

QuickSort Algorithm

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);

QuickSort Algorithm

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);

Illustration

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

Illustration

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

Illustration

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

--	--	--	--	--	--	--	--	--

Illustration

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

10								
----	--	--	--	--	--	--	--	--

Illustration

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

10								
----	--	--	--	--	--	--	--	--

Illustration

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

10								
----	--	--	--	--	--	--	--	--

Illustration

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

10								
----	--	--	--	--	--	--	--	--

Illustration

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

10	1							
----	---	--	--	--	--	--	--	--

Illustration

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

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

Illustration

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

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

Illustration

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

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

Illustration

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

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

Illustration

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

pivot

10	1	22		61				
----	---	----	--	----	--	--	--	--

Illustration

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

pivot

10	1	22		61				46
----	---	----	--	----	--	--	--	----

Illustration

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

pivot

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

Illustration

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

pivot

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

Illustration

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

pivot

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

Illustration

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

pivot

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

pivot

pivot

1		22		46				
---	--	----	--	----	--	--	--	--

Illustration

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

pivot

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

pivot

pivot

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

Illustration

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

pivot

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

pivot

pivot

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

Illustration

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

pivot

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

pivot

pivot

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

Illustration

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

pivot

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

pivot

pivot

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

pivot

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

Illustration

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

pivot

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

pivot

pivot

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

pivot

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

Illustration

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

pivot

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

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	51	55	56	61
---	----	----	----	----	----	----	----	----

Illustration

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

pivot

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

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	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, ...)

- 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 😊):
 - Time Complexity: $\mathcal{O}(nk)$, (k : # of bits of largest number)
 - Space Complexity: $\mathcal{O}(n + k)$, $\mathcal{O}(nk)$, ...

Illustration

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

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
-------------	-------------	-------------	-------------

1000

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
-------------	-------------	-------------	-------------

1000

0011	0010
-------------	-------------

0110	0111
-------------	-------------

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
-------------	-------------	-------------	-------------

1000

0011	0010
-------------	-------------

0110	0111
-------------	-------------

1000

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
------	------	------	------

1000

0011	0010
------	------

0110	0111
------	------

1000

0010

0011

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
------	------	------	------

1000

0011	0010
------	------

0110	0111
------	------

1000

0010

0011

0110

0111

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
------	------	------	------

1000

0011	0010
------	------

0110	0111
------	------

1000

0010

0011

0110

0111

1000

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
------	------	------	------

1000

0011	0010
------	------

0110	0111
------	------

1000

0010

0011

0110

0111

1000

0010

0011

0110

0111

1000

Illustration

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

0011	0110	0111	1000	0010
------	------	------	------	------

0011	0110	0111	0010
------	------	------	------

1000

0011	0010
------	------

0110	0111
------	------

1000

0010

0011

0110

0111

1000

0010

0011

0110

0111

1000

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

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=kPRA0W1kECg>