Lecture 9: Sorting Algorithms

김수경

이화여자대학교 인공지능융합전공 소속

저작권 안내

(주)업스테이지가 제공하는 모든 교육 콘텐츠의 지식재산권은 운영 주체인 (주)업스테이지 또는 해당 저작물의 적법한 관리자에게 귀속되어 있습니다.

콘텐츠 일부 또는 전부를 복사, 복제, 판매, 재판매 공개, 공유 등을 할 수 없습니다.
유출될 경우 지식재산권 침해에 대한 책임을 부담할 수 있습니다.

유출에 해당하여 금지되는 행위의 예시는 다음과 같습니다.

- 콘텐츠를 재가공하여 온/오프라인으로 공개하는 행위
- 콘텐츠의 일부 또는 전부를 이용하여 인쇄물을 만드는 행위
- 콘텐츠의 전부 또는 일부를 녹취 또는 녹화하거나 녹취록을 작성하는 행위
- 콘텐츠의 전부 또는 일부를 스크린 캡쳐하거나 카메라로 촬영하는 행위
- 지인을 포함한 제3자에게 콘텐츠의 일부 또는 전부를 공유하는 행위
- 다른 정보와 결합하여 Upstage Education의 콘텐츠임을 알아볼 수 있는 저작물을 작성, 공개하는 행위
- 제공된 데이터의 일부 혹은 전부를 Upstage Education 프로젝트/실습 수행 이외의 목적으로 사용하는 행위

Insertion Sort in Life

[1]

The Great Dalmuti

[1] http://www.partyhatpotato.com/reviews/2017/7/great-dalmuti-review.html

01

Insertion Sort

upstage Education

Sorting Problem

- Input: a list of N numbers
- Output: permutation B[1, ⋯, n] of A such that B[1] ≤ B[2] ≤ ⋯ ≤ B[n]
  - In other words, a rearranged list of items in the input, such that they monotonically increase (or decrease).
- Example:
  - Input:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

  - Expected output:

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	-4	-1	2	4	5	6	10	15	21

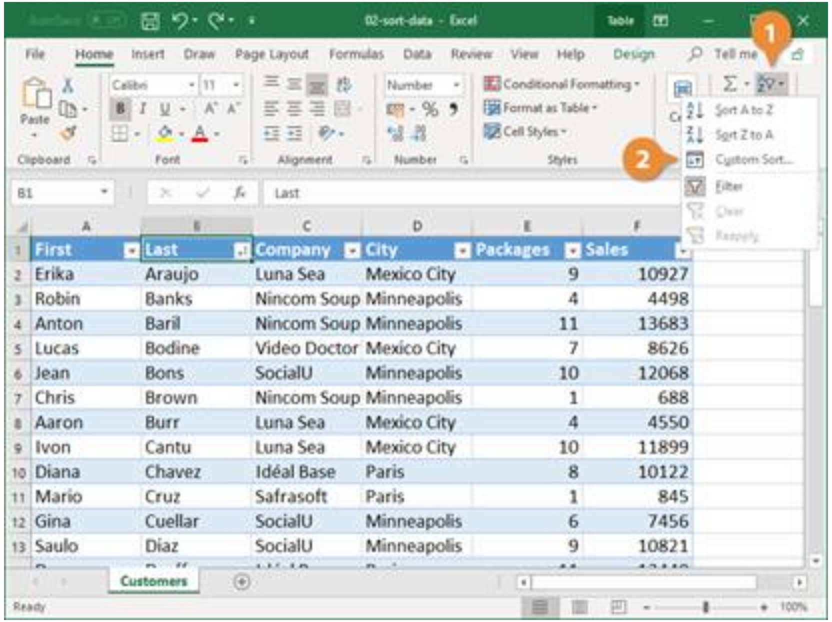
upstage Education

Why Sorting?

- Problems become easier once items are in sorted order:
  - Finding a median
  - Finding the closest pairs
  - Binary search
  - Identifying statistical outliers

- Various applications
  - Sorting a table by an attribute
  - Data compression: sorting finds duplicates.

[1]



First	Last	Company	City	Packages	Sales
Erika	Araujo	Luna Sea	Mexico City	9	10927
Robin	Banks	Nincom Soup	Minneapolis	4	4498
Anton	Baril	Nincom Soup	Minneapolis	11	13683
Lucas	Bodine	Video Doctor	Mexico City	7	8626
Jean	Bons	SocialU	Minneapolis	10	12068
Chris	Brown	Nincom Soup	Minneapolis	1	688
Aaron	Burr	Luna Sea	Mexico City	4	4550
Ivon	Cantu	Luna Sea	Mexico City	10	11899
Diana	Chavez	Idéal Base	Paris	8	10122
Mario	Cruz	Safrasoft	Paris	1	845
Gina	Cuellar	SocialU	Minneapolis	6	7456
Saulo	Diaz	SocialU	Minneapolis	9	10821

upstage Education

(Human-like) Insertion Sort

- Main idea:
  - Start from an empty “sorted list” and a pile of items to sort.
  - For each item on the pile, put in the right position in the “sorted list”.
  - When there’s no item left in the pile, you are done.

Pile

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted list

Index
Value

upstage Education

(Human-like) Insertion Sort

- Main idea:
  - Start from an empty “sorted list” and a pile of items to sort.
  - For each item on the pile, put in the right position in the “sorted list”.
  - When there’s no item left in the pile, you are done.

Pile

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted list

Index	0
Value	-7

upstage Education

(Human-like) Insertion Sort

- Main idea:
  - Start from an empty “sorted list” and a pile of items to sort.
  - For each item on the pile, put in the right position in the “sorted list”.
  - When there’s no item left in the pile, you are done.

Pile

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted list

Index	**0**	**1**
Value	-7	15

(Human-like) Insertion Sort

- Main idea:
  - Start from an empty "sorted list" and a pile of items to sort.
  - For each item on the pile, put in the right position in the "sorted list".
  - When there's no item left in the pile, you are done.

Pile

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted list

Index	0	1	2
Value	-7	2	15

upstage Education

(Human-like) Insertion Sort

- Main idea:
  - Start from an empty "sorted list" and a pile of items to sort.
  - For each item on the pile, put in the right position in the "sorted list".
  - When there's no item left in the pile, you are done.

Pile

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted list

Index	0	1	2	3	4	5	6	7	8
Value	-7	-4	-1	2	4	5	6	10	15

upstage Education

Insertion Sort

- Let's do this **in-place**, without using an additional list.
  - Start from an empty "sorted ~~list~~ region" and ~~a pile~~ the region of items to sort.
  - For each item ~~on the pile~~ in the unsorted region, put in the right position in the "sorted ~~list~~ region".
  - When there's no item left in the ~~pile~~ unsorted region, you are done.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted **Unsorted**

upstage Education

Insertion Sort

- Let's do this **in-place**, without using an additional list.
  - Start from an empty "sorted ~~list~~ region" and ~~a pile~~ the region of items to sort.
  - For each item ~~on the pile~~ in the unsorted region, put in the right position in the "sorted ~~list~~ region".
  - When there's no item left in the ~~pile~~ unsorted region, you are done.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted **Unsorted**

upstage Education

Insertion Sort

- Let's do this **in-place**, without using an additional list.
  - Start from an empty "sorted ~~list~~ region" and ~~a pile~~ the region of items to sort.
  - For each item ~~on the pile~~ in the unsorted region, put in the right position in the "sorted ~~list~~ region".
  - When there's no item left in the ~~pile~~ unsorted region, you are done.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted **Unsorted**

upstage Education

Insertion Sort

- Let's do this **in-place**, without using an additional list.
  - Start from an empty "sorted ~~list~~ region" and ~~a pile~~ the region of items to sort.
  - For each item ~~on the pile~~ in the unsorted region, put in the right position in the "sorted ~~list~~ region".
  - When there's no item left in the ~~pile~~ unsorted region, you are done.

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	2	15	6	-1	5	4	10	-4	21

Sorted | Unsorted

upstage Education

Insertion Sort

- Let's do this **in-place**, without using an additional list.
  - Start from an empty "sorted ~~list~~ region" and ~~a pile~~ the region of items to sort.
  - For each item ~~on the pile~~ in the unsorted region, put in the right position in the "sorted ~~list~~ region".
  - When there's no item left in the ~~pile~~ unsorted region, you are done.

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	2	6	15	-1	5	4	10	-4	21

Sorted | Unsorted

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-4	-1	2	4	5	6	10	15	21

Sorted

upstage Education

Insertion Sort: Implementation

```
def insertion_sort(list):
  for i in range(1, len(list)):
    key = list[i]
    j = i - 1
    while j >=0 and key < list[j]:
      list[j+1] = list[j]
      j -= 1
    list[j+1] = key
```

j i

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-1	2	5	6	15	4	10	-4	21

Sorted key

upstage Education

Time Complexity

Complexity?

- At the i-th iteration, its inner loop (while) does:
  - **Find** the location to put the next item among $\boxed{\ ?\ }$ items, — O(N) ← O(log N) if binary search used.
  - **Shift** all items on the right side by 1, — O(N)
  - **Put** the target item at the found position. — O(1)
- How many iterations? — O(N)
- Overall complexity? — O(N^2)
- If the input list is *almost* sorted, each iteration will be done by ≈O(1), so overall time complexity will be ≈O(N).

02

Selection Sort

Selection Sort

- Main idea: the opposite of insertion sort!
  - Instead of putting the next item in the right place,
  - Find the smallest item in the unsorted region, and
  - Swap it with the item in its right position.

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted Unsorted

Find the smallest among unsorted ones.

The smallest is already in the target place 😀

Selection Sort

- Main idea: the opposite of insertion sort!
  - Instead of putting the next item in the right place,
  - Find the smallest item in the unsorted region, and
  - Swap it with the item in its right position.

Index	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Value	-7	15	2	6	-1	5	4	10	-4	21

Sorted **Unsorted**

Find the smallest among unsorted ones.

Swap it with the one at the target position.

upstage Education

Selection Sort

- Main idea: the opposite of insertion sort!
  - Instead of putting the next item in the right place,
  - Find the smallest item in the unsorted region, and
  - Swap it with the item in its right position.

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-4	2	6	-1	5	4	10	15	21

Sorted Unsorted

Find the smallest among unsorted ones.

Swap it with the one at the target position.

upstage Education

Selection Sort

- Main idea: the opposite of insertion sort!
  - Instead of putting the next item in the right place,
  - Find the smallest item in the unsorted region, and
  - Swap it with the item in its right position.

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-4	-1	6	2	5	4	10	15	21

Sorted Unsorted

Find the smallest among unsorted ones.

Swap it with the one at the target position.

Repeat until there is no item left in the unsorted region!

upstage Education

Selection Sort: Implementation

```
def selection_sort(list):
  for i in range(len(list)):
    smallest = i
    for j in range(i+1, len(list)):
      if list[j] < list[smallest]:
        smallest = j
    list[i], list[smallest] = list[smallest], list[i]
```

i   swap

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-4	-1	6	10	5	4	2	15	21

Sorted   smallest

Q. Can you implement this with recursion?

upstage Education

Time Complexity

Complexity?

- At the i-th iteration, its inner loop (for) does:
  - **Find** the smallest items among $\boxed{\ ?\ }$ unsorted items, $O(N)$
  - **Swap** it with the item at the next position. $O(1)$
- How many iterations? $O(N)$
- Overall complexity? $O(N^2)$
- No benefit even though the input list is *almost* sorted.
  - When we find the next smallest item, we do not assume the remaining list is sorted.

03

Merge Sort

Motivation

- Insertion sort and selection sort work but too slow.
  - Time complexity is $O(N^2)$.
  - Does not matter when handling small data, but we want to handle **big data**!
- Any better idea?
  - Divide and conquer!

Merge Sort

- Main idea:
  - **Divide** the whole list into two sub-lists.
  - Sort the left and right sublists **separately**.
  - **Merge** the two sorted sublists into a single sorted one.

Index	0	1	2	3	4	5	6	7
Value	5	-2	0	10	-6	7	4	9

Sort

Index	0	1	2	3
Value	-2	0	5	10

Sort

Index	0	1	2	3
Value	-6	4	7	9

Merge

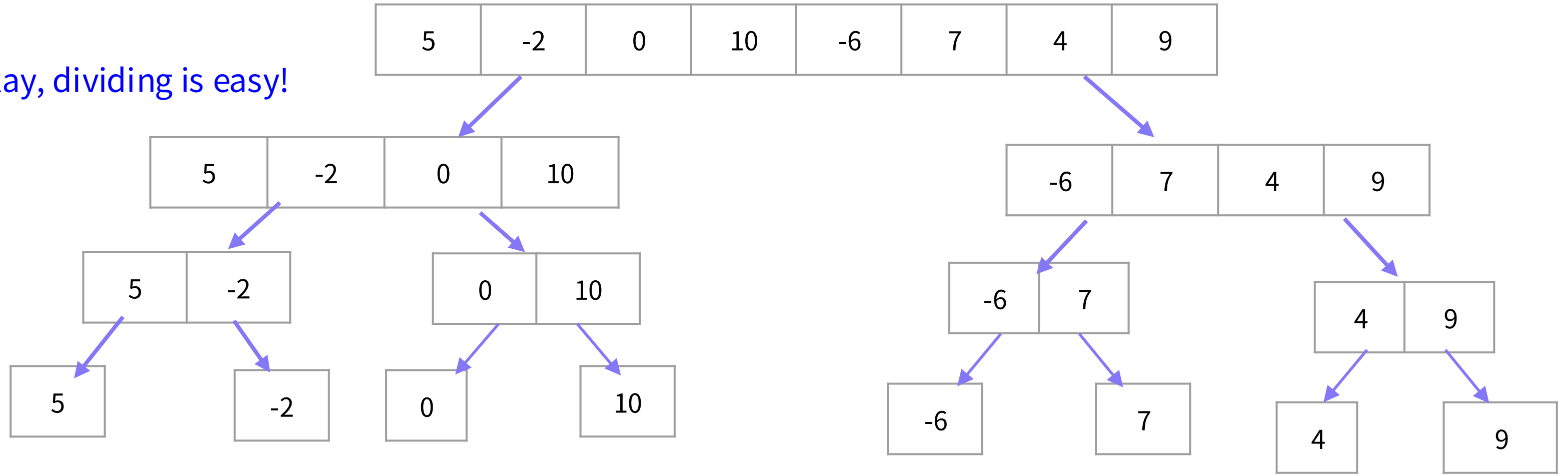
Index	0	1	2	3	4	5	6	7
Value	-6	-2	0	4	5	7	9	10

upstage Education

Merge Sort

- Further breakdown: each sublist is also sorted in a similar way!

Okay, dividing is easy!

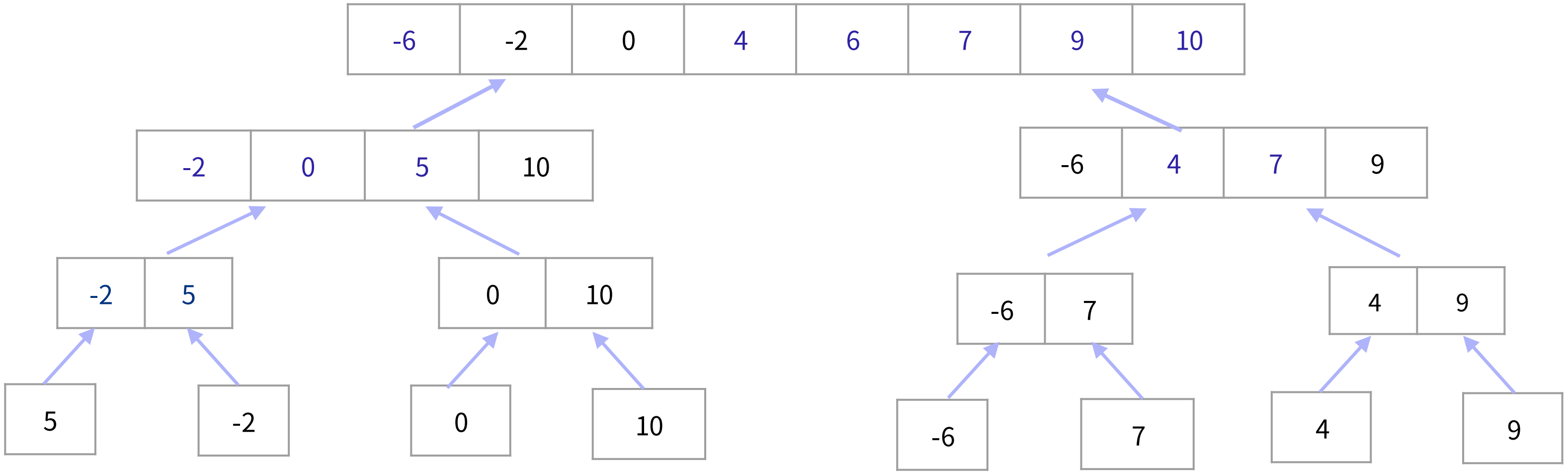


Sorting each separately is also easy, as they have just a single item in the end.

upstage Education

Merge Sort

- Further breakdown: each sublist is also sorted in a similar way!



What about merging?!

Merge Sort

- At each step, the only non-trivial task is **merging** two sorted arrays into a single sorted array.

-6	-2	0	4	6	7	9	10

-2	0	5	10

-6	4	7	9

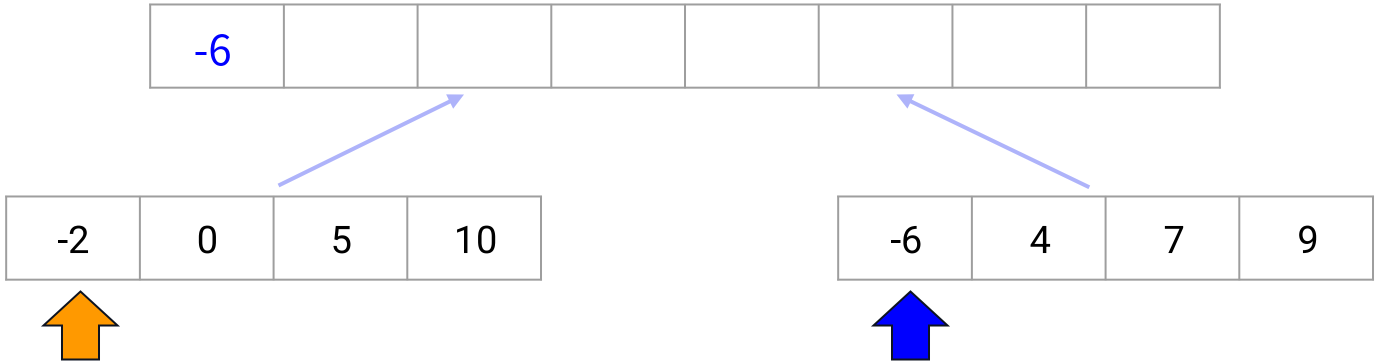
- First trial with brute force:
  - Concatenate the two list
  - Sort the entire list with insertion (or selection) sort.
  - Time complexity?
    - 1st split: $2 \times (N/2)^2$, 2st split: $4 \times (N/4)^2$, 3rd split: $8 \times (N/8)^2$, $\cdots$
    - In total, $O(N^2)$. No benefit from directly sorting entire matrix at once.

upstage Education

Merge Sort

- So, we need more efficient merging, taking advantage of the fact that the two sublists are already sorted.

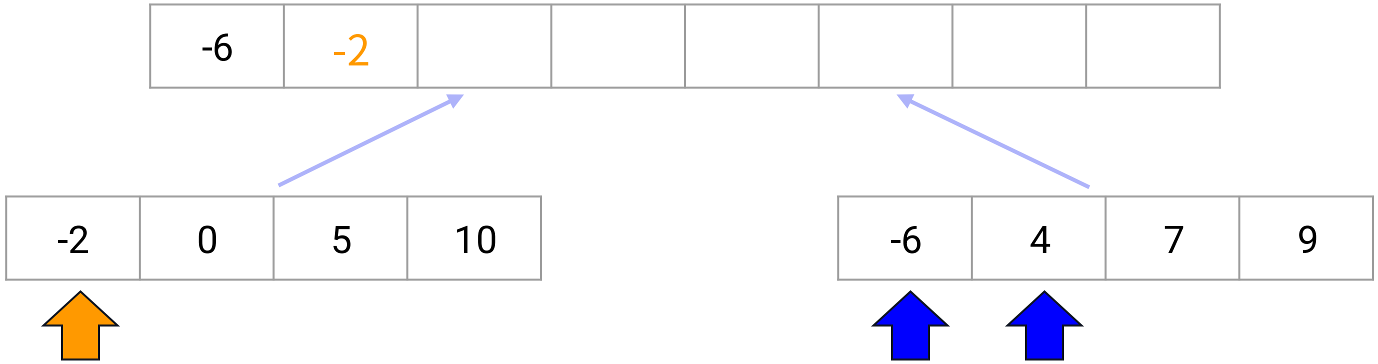
The first (smallest) item must be either the smallest item in the left sublist or the smallest one in the right sublist.



Merge Sort

- Then, what about the next one?

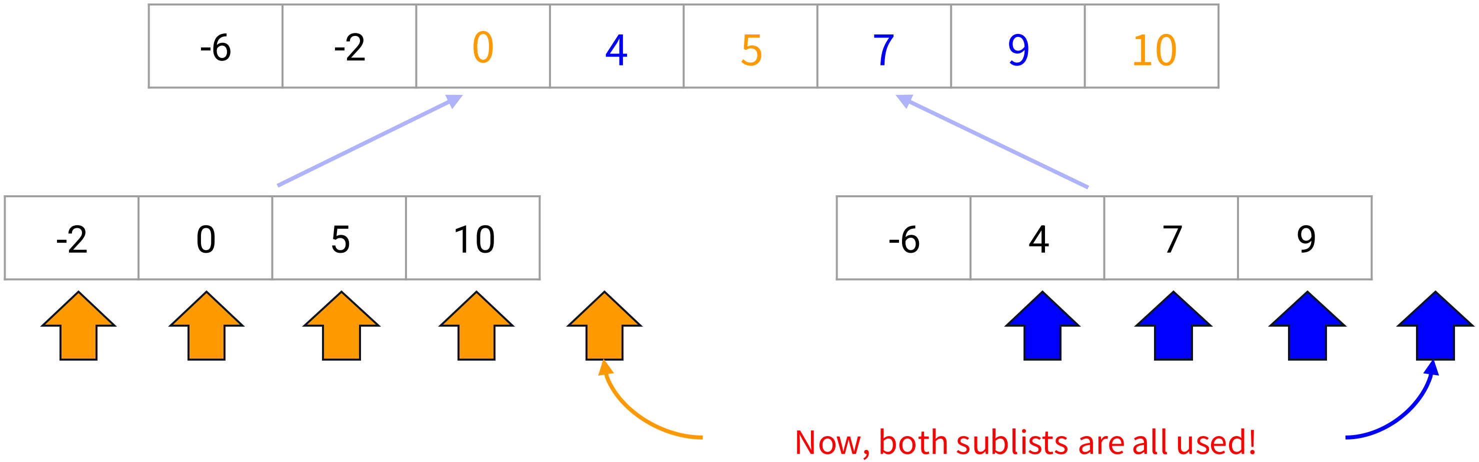
The second smallest item must be either the smallest item in the left sublist or the second smallest one in the right sublist (since -6 was already used).



upstage Education

Merge Sort

- In a similar way, we choose the smaller one between the smallest remaining items in each list at a time, until both lists are completely consumed.



upstage Education

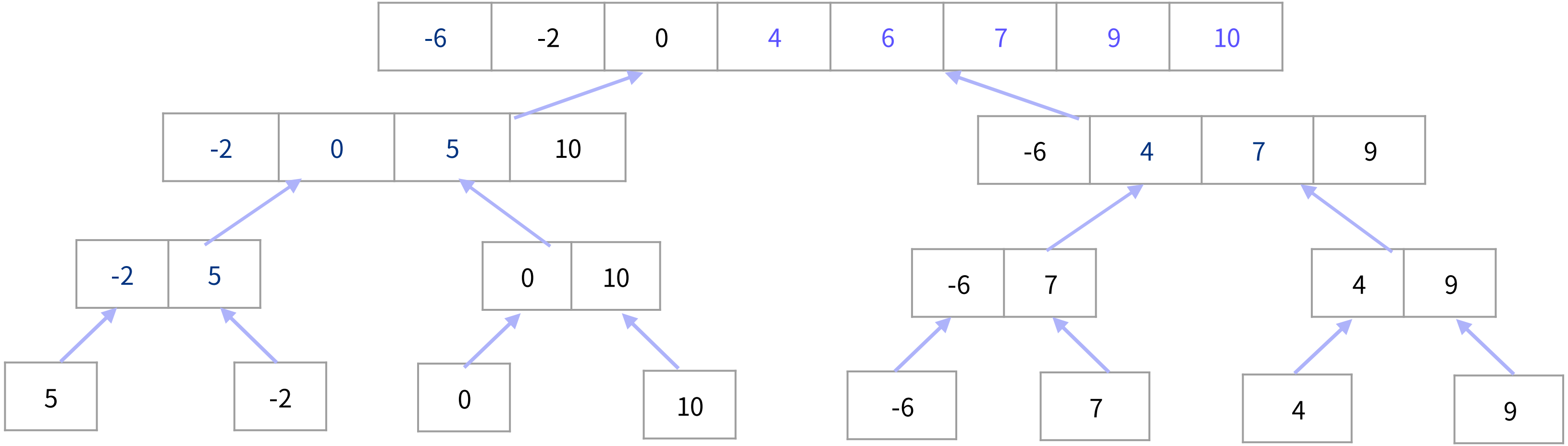
Merge Sort: Time Complexity

- Time complexity of this merging step?
  - At each time we fill the output, we
    - Compare two elements once → O(1)
    - Write the small one → O(1)
    - Move one pointer on the selected side → O(1)
  - How many times do we repeat this?
    - Same as the output list size!
    - 1st split: $2 \times (N/2)$, 2nd split: $4 \times (N/4)$, $\cdots \rightarrow$ **$O(N)$**

upstage Education

Merge Sort: Time Complexity

- How many steps do we have?
  - Same as the height of this tree: **O(log *N*)** So, overall time complexity is: **O(*N* log *N*)**



upstage Education

Merge Sort: Implementation

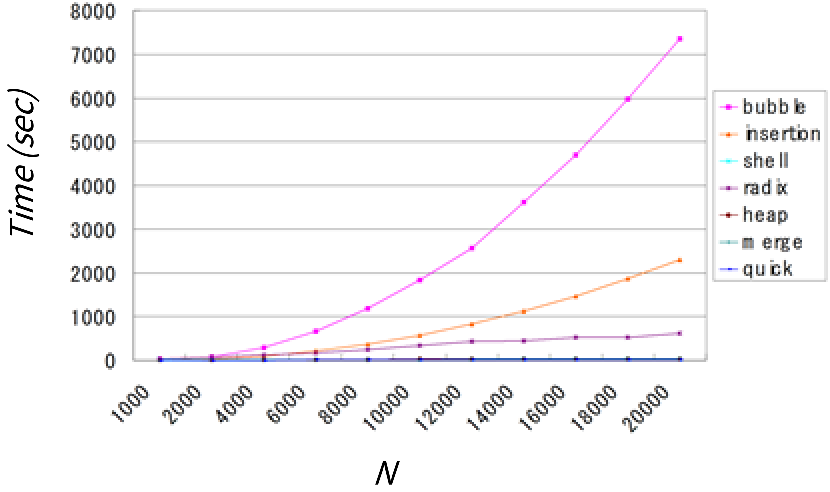
```
def merge_sort(list):
    if len(list) > 1:
        mid = len(list) // 2 # round down
        left = list[:mid]
        right = list[mid:]

        merge_sort(left)
        merge_sort(right)

        # TODO(students): merge left & right in the list
```

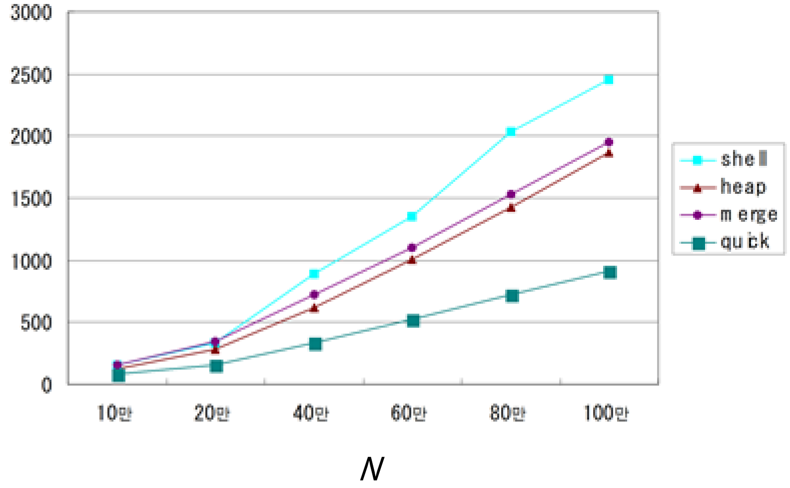
“else” for this is the base case, where the recursive calls are done.
We skip it here, since there’s no action item in the base case.

upstage Education

Merge Sort vs. Insertion Sort

- How different the speed is, between **O(N^2)** vs. **O(N log N)**?





upstage Education

Can we do better?

- Insertion / Selection sort takes **O(N^2)**.
- Merge sort takes **O(N log N)**.
- Can we do better?

What is the best possible time complexity for sorting problem?

O(N), because we need O(N) to read the input list anyway.

Then, can we sort in O(N)?

Yes, if we have some additional conditions.

(It was proved that O(N log N) is the best for comparison-based sorting algorithms.)

04

Linear Time Sorting: O (n)

upstage Education

Counting Sort

- A linear-time sorting algorithm under the condition that the input elements are always **integers** between 0 and ***k***.
- Example:

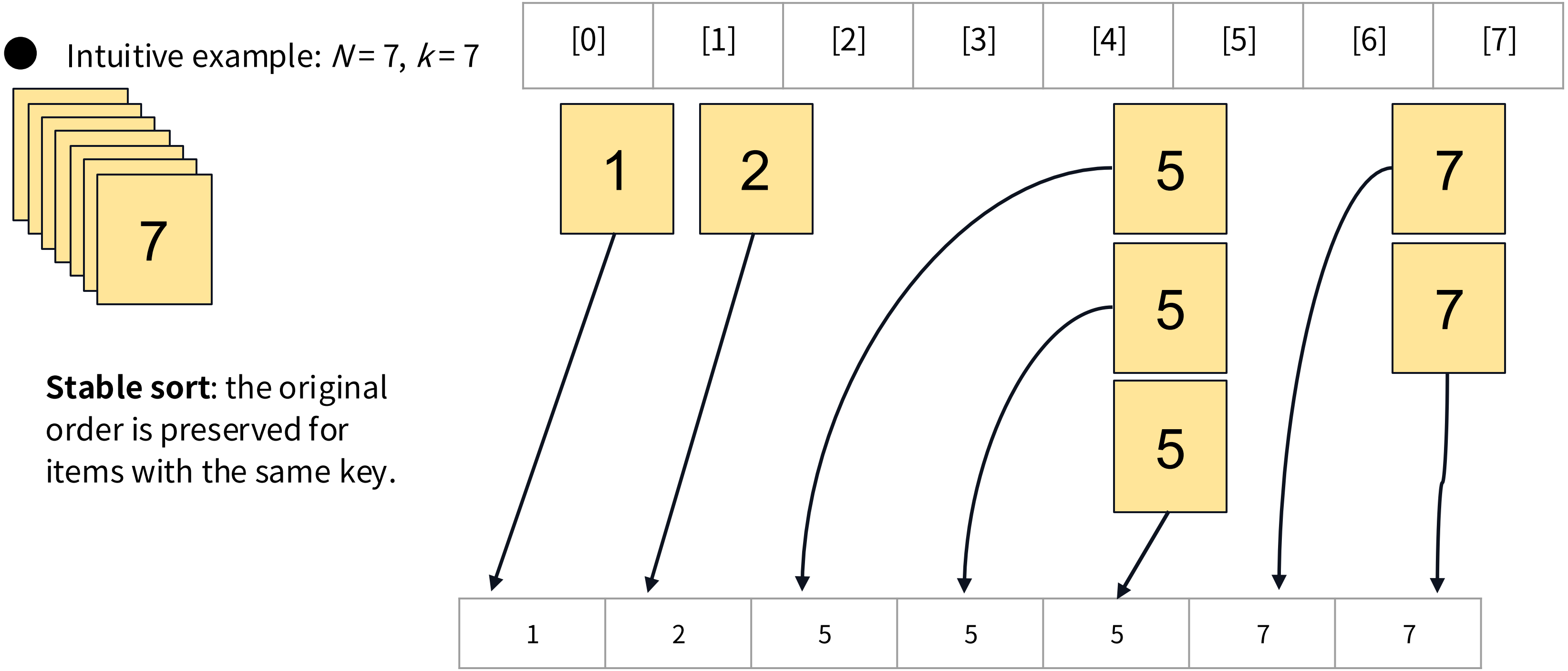
1	Order Id	Order Date	Employee Name	Reporting Manager	Customer Name	Customer Contact	Customer Email	Product Name	Product Price	Product Qty	Order Total
2	1	31-Oct-17	Abhay Gaurav	Aakash Harit	Chloe Jones	919-555-8658	lo@email.com	Apple	INR 67.00	81	INR 5,427.00
3	2	26-Jun-17	Nisha Kumari	Aakash Harit	Brett Newkirk	919-555-7653	newkb@email.com	Banana	INR 31.00	41	INR 1,271.00
4	3	20-Jan-17	Abhay Gaurav	Aakash Harit	Tracey Beckham	919-555-2314	beck@email.com	Banana	INR 93.00	31	INR 2,883.00
5	4	12-Dec-17	Nishu Kumari	Aakash Harit	Brett Newkirk	919-555-7653	newkb@email.com	Apple	INR 83.00	6	INR 498.00
6	5	18-Jul-17	Abhay Gaurav	Aakash Harit	Lucinda George	919-555-4534	lugeo@email.com	Apple	INR 98.00	47	INR 4,606.00
7	6	14-Mar-17	Abhay Gaurav	Aakash Harit	Jerrod Smith	919-555-4564	texj@email.com	Banana	INR 89.00	76	INR 6,764.00
8	7	21-Nov-17	Nishu Kumari	Aakash Harit	Lucinda George	919-555-4534	lugeo@email.com	Grapes	INR 80.00	26	INR 2,080.00
9	8	22-May-17	Nishu Kumari	Aakash Harit	Jerrod Smith	919-555-4564	texj@email.com	Grapes	INR 36.00	59	INR 2,124.00
10	9	10-Nov-17	Nishu Kumari	Aakash Harit	Chloe Jones	919-555-8658	lo@email.com	Apple	INR 85.00	15	INR 1,275.00
11	10	30-Oct-17	Abhay Gaurav	Aakash Harit	Tracey Beckham	919-555-2314	beck@email.com	Pineapple	INR 47.00	80	INR 3,760.00
12	11	30-May-17	Nishu Kumari	Aakash Harit	Brett Newkirk	919-555-7653	newkb@email.com	Pineapple	INR 38.00	47	INR 1,786.00
13	12	13-May-17	Nishu Kumari	Aakash Harit	Brett Newkirk	919-555-7653	newkb@email.com	Apple	INR 94.00	37	INR 3,478.00
14	13	02-Jan-17	Nisha Kumari	Aakash Harit	Lucinda George	919-555-4534	lugeo@email.com	Apple	INR 23.00	81	INR 1,863.00
15	14	16-Sep-17	Abhay Gaurav	Aakash Harit	Jerrod Smith	919-555-4564	texj@email.com	Banana	INR 56.00	65	INR 3,640.00
16	15	16-Sep-17	Abhay Gaurav	Aakash Harit	Jerrod Smith	919-555-4564	texj@email.com	Banana	INR 90.00	82	INR 7,380.00
17	16	22-May-17	Nishu Kumari	Aakash Harit	Jerrod Smith	919-555-4564	texj@email.com	Grapes	INR 66.00	17	INR 1,122.00
18	17	30-Jan-17	Vishal Kumar	Divya Sharma	Lucinda George	919-555-4534	lugeo@email.com	Pineapple	INR 56.00	36	INR 2,016.00
19	18	18-Jul-17	Abhay Gaurav	Aakash Harit	Lucinda George	919-555-4534	lugeo@email.com	Apple	INR 63.00	36	INR 2,268.00
20	19	28-Jul-17	Vishal Kumar	Divya Sharma	Tracey Beckham	919-555-2314	beck@email.com	Grapes	INR 38.00	15	INR 570.00
21	20	24-Mar-17	Vishal Kumar	Divya Sharma	Brett Newkirk	919-555-7653	newkb@email.com	Apple	INR 57.00	72	INR 4,104.00
22	21	01-Dec-17	Rajkumar Singh	Divya Sharma	Lucinda George	919-555-4534	lugeo@email.com	Grapes	INR 33.00	62	INR 2,046.00
23	22	01-Jun-17	Rajkumar Singh	Divya Sharma	Chloe Jones	919-555-8658	lo@email.com	Grapes	INR 45.00	81	INR 3,645.00
24	23	20-Nov-17	Rajkumar Singh	Divya Sharma	Tracey Beckham	919-555-2314	beck@email.com	Banana	INR 48.00	46	INR 2,208.00
25	24	09-Nov-17	Vishal Kumar	Divya Sharma	Brett Newkirk	919-555-7653	newkb@email.com	Grapes	INR 96.00	15	INR 1,440.00
26	25	09-Jun-17	Rajkumar Singh	Divya Sharma	Tracey Beckham	919-555-2314	beck@email.com	Banana	INR 70.00	34	INR 2,380.00
27	26	23-May-17	Rajkumar Singh	Divya Sharma	Chloe Jones	919-555-8658	lo@email.com	Banana	INR 65.00	5	INR 325.00
28	27	12-Jan-17	Rajkumar Singh	Divya Sharma	Chloe Jones	919-555-8658	lo@email.com	Grapes	INR 42.00	45	INR 1,890.00
29	28	26-Sep-17	Vishal Kumar	Divya Sharma	Jerrod Smith	919-555-4564	texj@email.com	Banana	INR 57.00	84	INR 4,788.00

[1]

upstage Education

Counting Sort

- Intuitive example: $N = 7$, $k = 7$



Stable sort: the original order is preserved for items with the same key.

upstage Education

Counting Sort

- Algorithm

7	5	1	5	2	7	5

  - Reading the entire input list, increase the number of occurrences of each key.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
0	1	1	0	0	3	0	2

  - Take cumulative sum of this counts.
    → This is the **ending index** of each number in the sorted output.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
0	1	2	2	2	5	5	7
0:0	0:1	1:2	2:2	2:2	2:5	5:5	5:7

upstage Education

Counting Sort

- Algorithm

7	5	1	5	2	7	5

  - Read the input list once again from backward,
  - Subtract the target index by 1,
  - Put the element there, and move on.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
0	1	**1**	2	2	**3**	5	**6**
	0:1	1:2	2:2	2:2	2:5	5:5	5:7

[0]	[1]	[2]	[3]	[4]	[5]	[6]
	2		5	5		7

Why backward?

→ For stable sort!

upstage Education

Counting Sort: Implementation

```
def counting_sort(list):
  output = [0] * (len(list))
  count = [0] * (max(list) + 1)

  for i in range(len(list)):
    count[list[i]] += 1

  for i in range(1, len(count)):
    count[i] += count[i-1]

  for i in range(len(list)):
    j = len(list) - 1 - i
    count[list[j]] -= 1
    index = count[list[j]]
    output[index] = list[j]

  return output
```

Time complexity?

Count occurrences of each key. $O(N)$

Cumulative sum $O(k)$

Locate each element at the right position in the output. $O(N)$

Overall, $O(N+k)$.
If $k \le N$, counting sort runs in linear time on the input size (N).

upstage Education

Sorting in Reality

- What sorting algorithm is used in Python library?
  - Timsort (2002): a hybrid sorting algorithm (merge sort + insertion sort)
    - A variant of merge sort (divide and conquer)
    - When a sublist becomes smaller than some threshold, it is sorted using insertion sort.
    - Insertion sort is faster than merge sort for a small list.

- If your data is small enough, insertion/selection sort may work okay.
- Otherwise, merge sort or quick sort is recommended.
- You may consider a linear-time sorting if your key is integer within a reasonable range. However, it may be hard to take advantage unless the dataset is really huge, and you implement efficiently.

upstage Education

Problem 1

Q. Can you implement this with selection sort with recursion?

```
def selection_sort(list):
  for i in range(len(list)):
    smallest = i
    for j in range(i+1, len(list)):
      if list[j] < list[smallest]:
        smallest = j
    list[i], list[smallest] = list[smallest], list[i]
```

i

swap

Index	0	1	2	3	4	5	6	7	8	9
Value	-7	-4	-1	6	10	5	4	2	15	21

Sorted

smallest

upstage Education

Problem 2

Q. Please implement merge sort. (# TODO section below)

```
def merge_sort(list):
  if len(list) > 1:
    mid = len(list) // 2
    left = list[:mid]
    right = list[mid:]

    merge_sort(left)
    merge_sort(right)

    # TODO(students): merge left & right in the list
```

"`else`" for this is the base case, where the recursive calls are done.
We skip it here, since there's no action item in the base case.

Building intelligence for the future of work