http://www.cs.cornell.edu/courses/cs1110/2022sp

Lecture 23: More Algorithms for Sorting

CS 1110

Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]

Announcements

Next Tuesday:

- Lecture is a review session.
- There will be no post-lecture office hours.

Course Staff also hosting additional review sessions (possibly during study days). Announcements forthcoming.

Search Algorithms

Recall from last lecture:

- Searching for data is a common task
  - Linear search: on the order of n
    - input doubles? → work **doubles**!
  - Binary search: on the order of log2 n
    - input doubles? → work **increases by just 1 unit**!
    - BUT data needs to be sorted…
- **Sorting** data now suddenly interesting…

Sorting Algorithms

- Sorting data is a common task
  - Insertion sort: on the order of n^2
    - input doubles? → work **quadruples**! (yikes)
- Today's topic:
  - Merge sort: *can we do better than Insertion Sort?*

Which algorithm does Python's `sort` use?

- Recursive algorithm that runs much faster than insertion sort for the same size list (when the size is big)!
- A variant of an algorithm called "merge sort"
- Based on the idea that sorting is hard, but *"merging" two already sorted lists is easy*.

L | 11 | 17 | 19 | R | 14 | 15 | 16 | 18 |

merge

L | 11 | 14 | 15 | 16 | 17 | 18 | 19 |

Merge sort: Motivation

Since merging is easier than sorting, if I had two helpers, I'd…

- Give each helper half the array to sort
- Then I get back their sorted subarrays and merge them.

What if those two helpers each had two sub-helpers?

And the sub-helpers each had two sub-sub-helpers? And…

Subdivide the sorting task

H E M G B K A Q F L P D R C J N

H E M G B K A Q   F L P D R C J N

Subdivide again

H E M G B K A Q   F L P D R C J N

H E M G   B K A Q   F L P D   R C J N

And again

H E M G   B K A Q   F L P D   R C J N

H E   M G   B K   A Q   F L   P D   R C   J N

And one last time

H E   M G   B K   A Q   F L   P D   R C   J N

H   E   M   G   B   K   A   Q   F   L   P   D   R   C   J   N

Now merge

E H   G M   B K   A Q   F L   D P   C R   J N

H   E   M   G   B   K   A   Q   F   L   P   D   R   C   J   N

And merge again

E G H M   A B K Q   D F L P   C J N R

E H   G M   B K   A Q   F L   D P   C R   J N

And again

A B E G H K M Q   C D F J L N P R

E G H M   A B K Q   D F L P   C J N R

And one last time

A B C D E F G H J K L M N P Q R

A B E G H K M Q   C D F J L N P R

Done!

A B C D E F G H J K L M N P Q R


```
def mergeSort(li):
    """Sort list li using Merge Sort"""
    if len(li) > 1:
        # Divide into two parts
        mid= len(li)//2
        left= li[:mid]
        right= li[mid:]

        # Recursive calls
        mergeSort(left)
        mergeSort(right)

        # Merge left & right back to li
        ???
    # base case does nothing!
    # a list with len 0 or 1 is sorted!
```

The central sub-problem is the **merging** of two sorted lists into one single sorted list

12	33	35	45

15	42	55	65	75

Approach:
keep comparing the smallest element of first list with smallest element of second list.

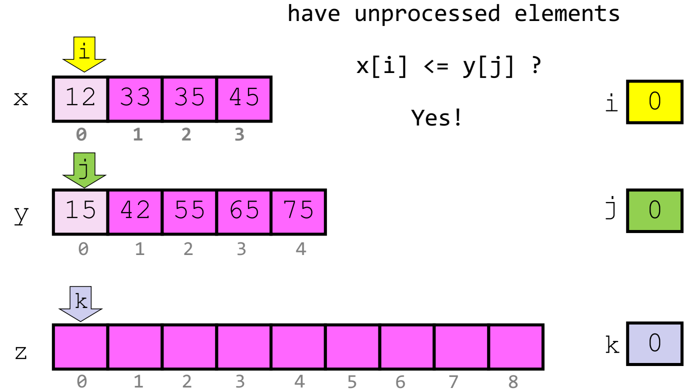
12	15	33	35	42	45	55	65	75

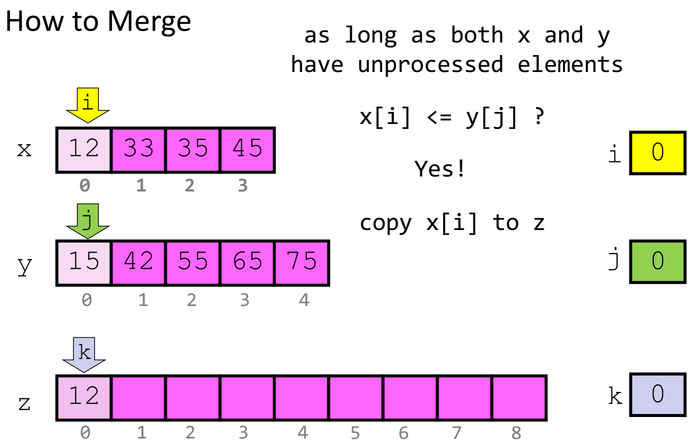
How to Merge

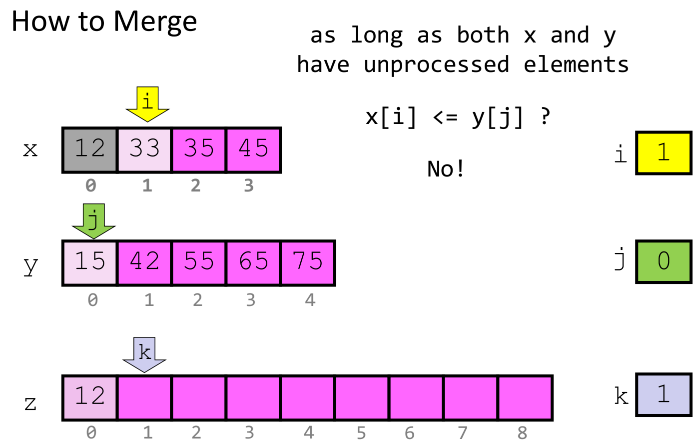
as long as both x and y have unprocessed elements

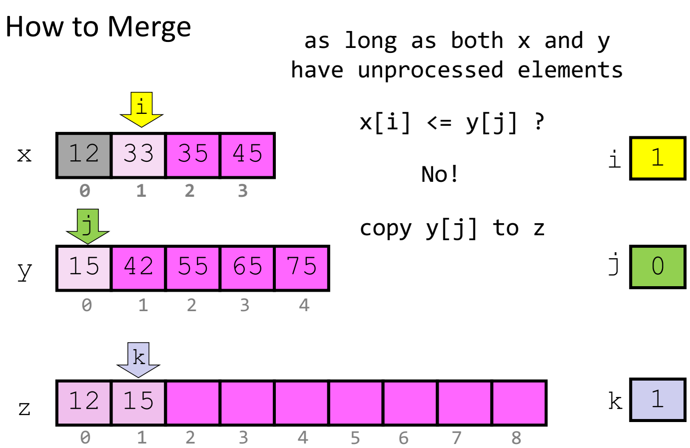
x[i] <= y[j] ?

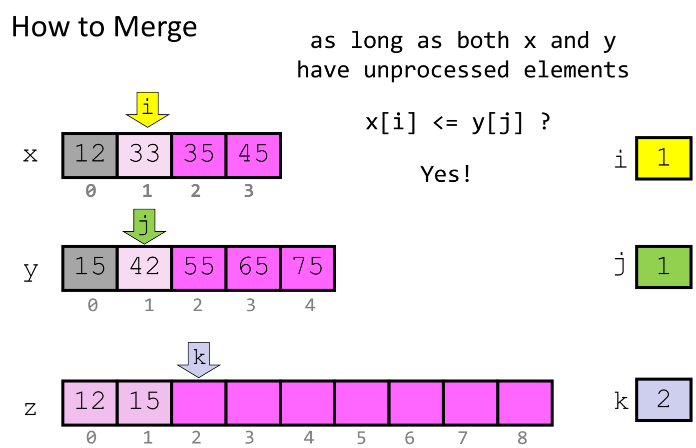
Yes!

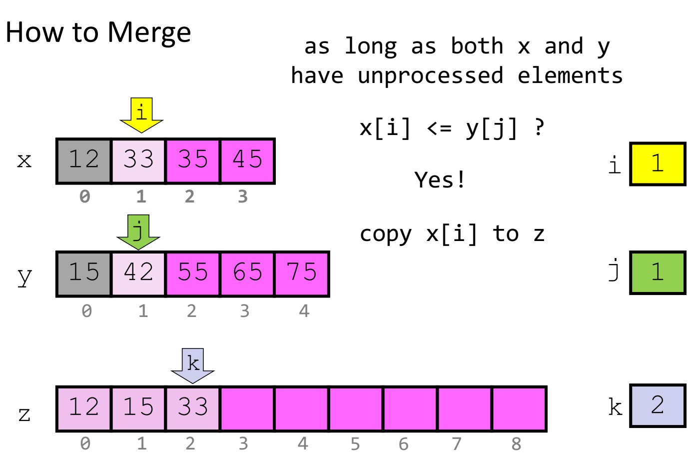


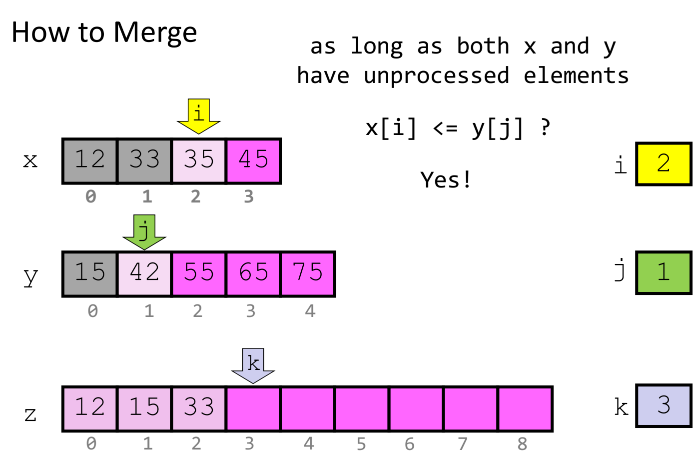

How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
Yes!
copy x[i] to z
x
12 33 35 45
y
15 42 55 65 75
z
12
i 0
j 0
k 0


How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
No!
x
12 33 35 45
y
15 42 55 65 75
z
12
i 1
j 0
k 1

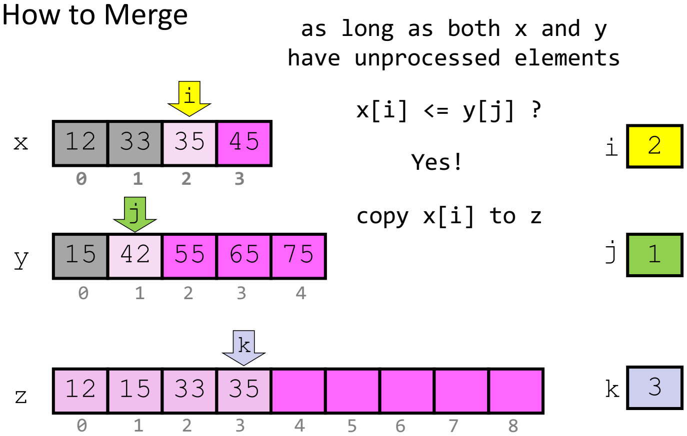

How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
No!
copy y[j] to z
x
12 33 35 45
y
15 42 55 65 75
z
12 15
i 1
j 0
k 1


How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
Yes!
x
12 33 35 45
y
15 42 55 65 75
z
12 15
i 1
j 1
k 2

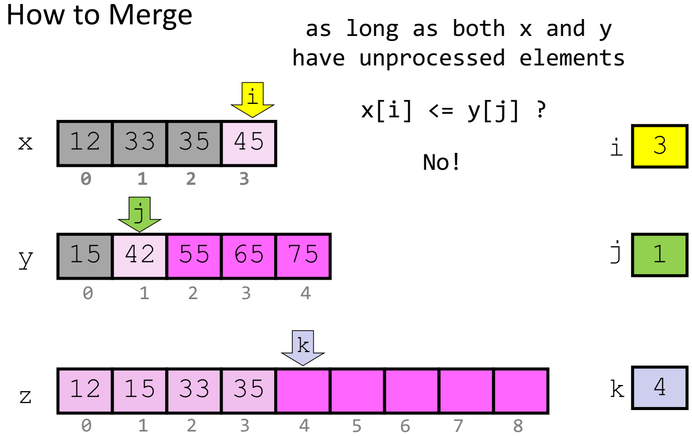

How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
Yes!
copy x[i] to z
x
12 33 35 45
y
15 42 55 65 75
z
12 15 33
i 1
j 1
k 2


How to Merge
as long as both x and y
have unprocessed elements
x[i] <= y[j] ?
Yes!
x
12 33 35 45
y
15 42 55 65 75
z
12 15 33
i 2
j 1
k 3

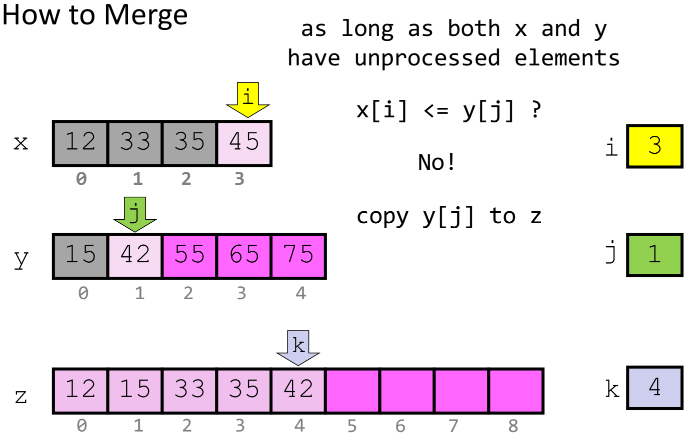
How to Merge



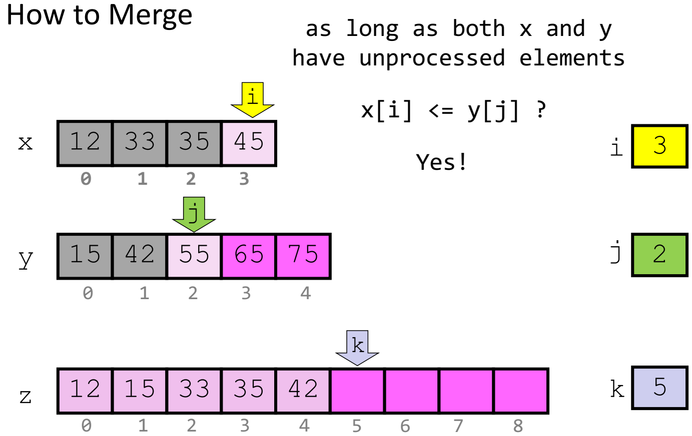
How to Merge



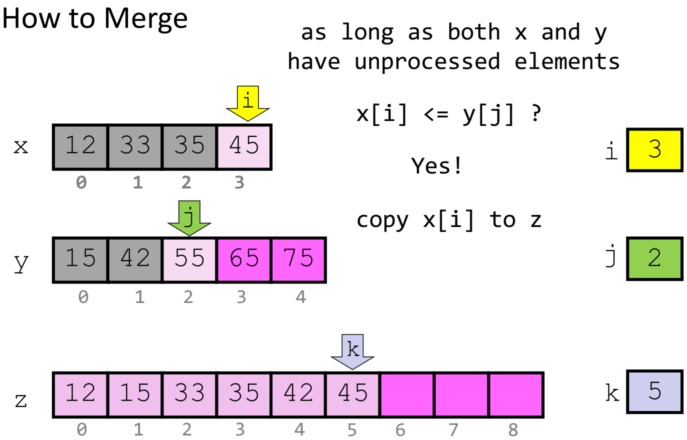
How to Merge



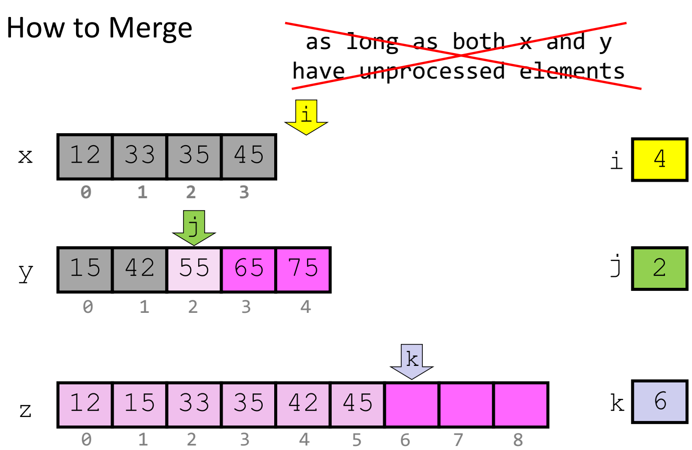
How to Merge



How to Merge



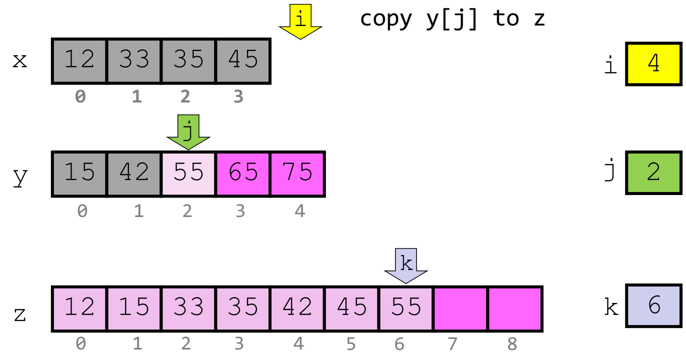
How to Merge



How to Merge

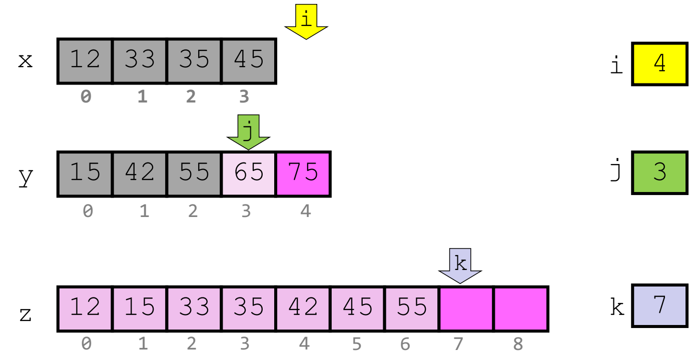
as long as y has
unprocessed elements

copy y[j] to z



How to Merge

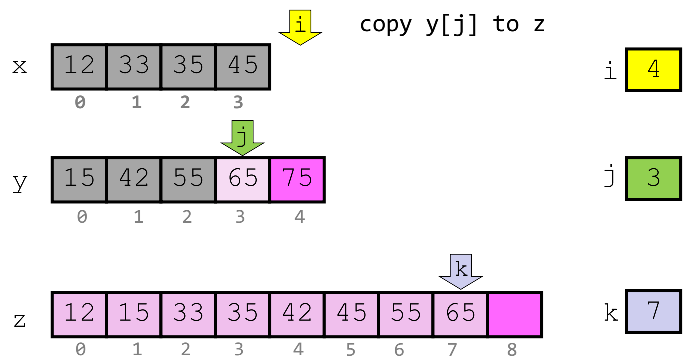
as long as y has
unprocessed elements



How to Merge

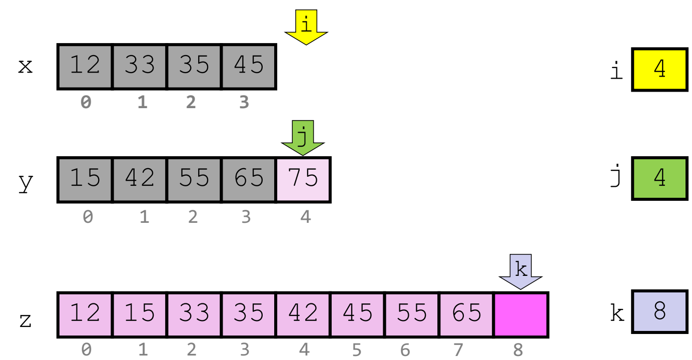
as long as y has
unprocessed elements

copy y[j] to z



How to Merge

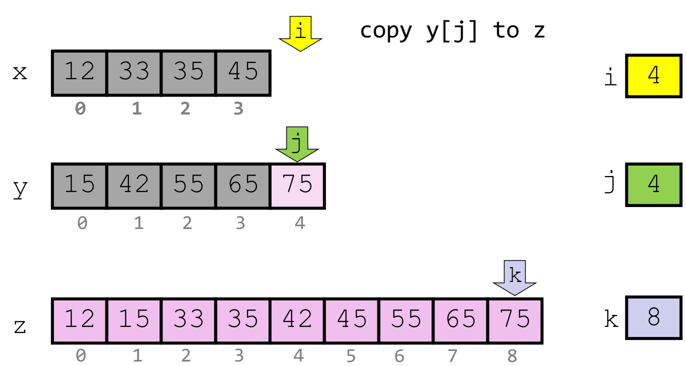
as long as y has
unprocessed elements



How to Merge

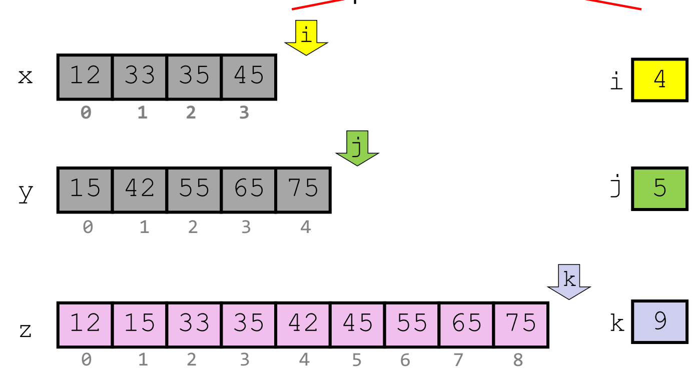
as long as y has
unprocessed elements

copy y[j] to z



How to Merge

~~as long as y has~~
~~unprocessed elements~~



(1/3)

```
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:





    # Deal with remaining values in x or y
```

(2/3)

```
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:
        if  x[i] <= y[j]:
            z[k]= x[i];  i=i+1
        else:
            z[k]= y[j];  j=j+1
        k=k+1
    # Deal with remaining values in x or y
```

(3/3)

```
def merge(x, y, z):
    # Given: sorted lists x and y
    # list z, has the combined length of x and y...
    nx = len(x); ny = len(y)

    i = 0; j = 0; k = 0;
    while i<nx and j<ny:
        if  x[i] <= y[j]:
            z[k]= x[i];  i=i+1
        else:
            z[k]= y[j];  j=j+1
        k=k+1
    # Deal with remaining values in x or y
    while i<nx:  # copy any remaining x-values
        z[k]= x[i];  i=i+1;  k=k+1

    while j<ny:  # copy any remaining y-values
        z[k]= y[j];  j=j+1;  k=k+1
```

```
def mergeSort(li):
    """Sort list li using Merge Sort"""

    if len(li) > 1:
        # Divide into two parts
        mid= len(li)/2
        left= li[:mid]
        right= li[mid:]

        # Recursive calls
        mergeSort(left)
        mergeSort(right)

        # Merge left & right back to li
        merge(left, right, li)
```


Sorting Algorithms

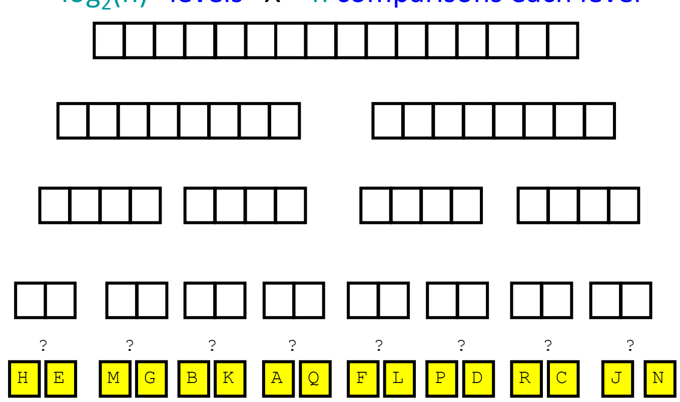
- Sorting data is a common task
  - Insertion sort: on the order of n^2
    - input doubles? → work **quadruples**! (yikes)
- Today's topic:
  - Merge sort: *did we do better than Insertion Sort?*

work = one comparison

How many comparisons do we make?

Merge sort:

~ $\log_2(n)$ “levels” X ~ n comparisons each level



Sorting Algorithms

- Sorting data is a common task
  - Insertion sort: on the order of n^2
    - input doubles? → work **quadruples**! (yikes)
  - Merge sort: on the order of $n \cdot \log_2(n)$

Order of magnitude difference

Should we always use merge sort then?

Python's **`sort`** *actually combines merge and insertion sort!*

For fun, check out the visualizations:
https://www.youtube.com/watch?v=xxcpvCGrCBc
https://www.youtube.com/watch?v=ZRPoEKHXTJg