Merge Sort

Basics

- Merge sort is usually described recursively.
  - The recursive calls work on smaller and smaller part of the list.
- Idea:
  1. Split the list into two halves.
  2. Recursively sort each half.
  3. "Merge" the two halves into a single sorted list.

Example

0	1	2	3	4	5	6	7	8	9	10	11	12
17	8	75	23	14	95	29	87	74	-1	-2	37	81

1. Split:

0	1	2	3	4	5	6
17	8	75	23	14	95	29

7	8	9	10	11	12
87	74	-1	-2	37	81

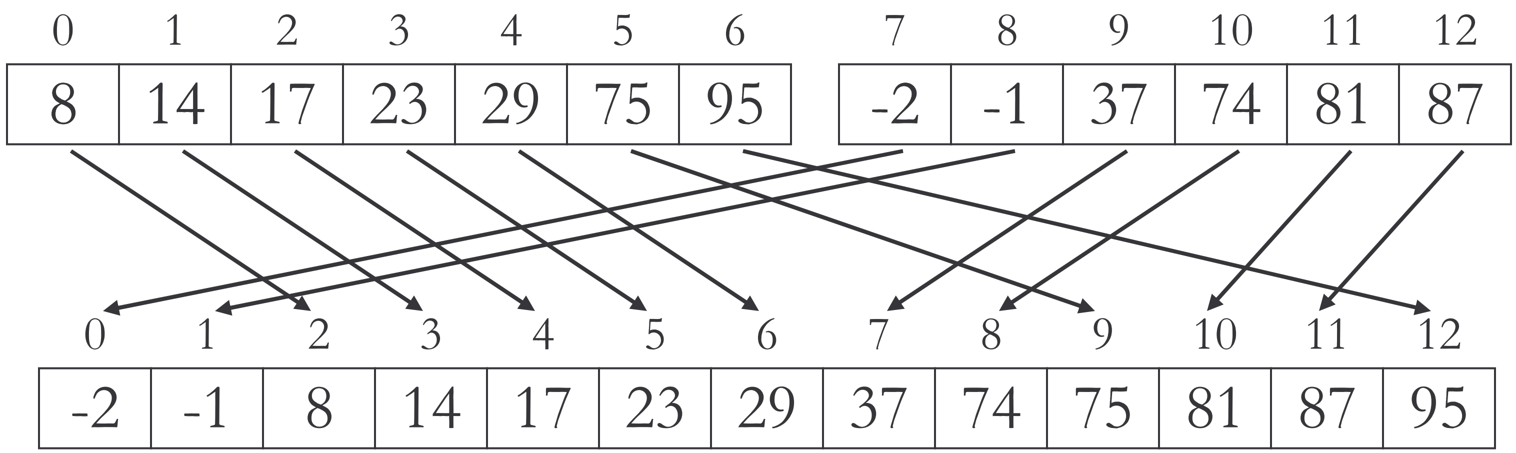
2. Recursively sort:

0	1	2	3	4	5	6
8	14	17	23	29	75	95

7	8	9	10	11	12
-2	-1	37	74	81	87

3. Merge?

Example Merge



- Must put the next-smallest element into the merged list at each step.
- each "next-smallest" could come from either half

Merge Sort Algorithm

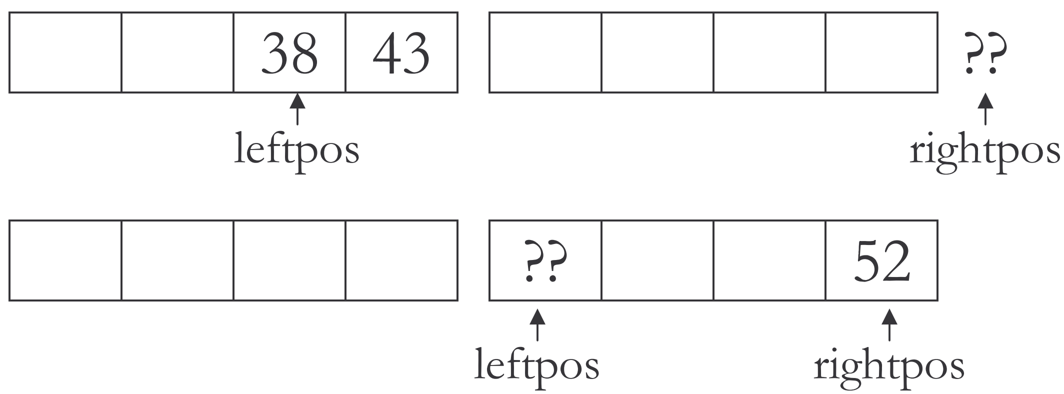
```
mergeSort(array, first, last):
    // sort array[first] to array[last-1]
    if last - first ≤ 1:
        return  // arrays of length 0, 1 are already sorted
    mid = (first + last)/2
    mergeSort(array, first, mid)  // recursive call 1
    mergeSort(array, mid, last)  // recursive call 2
    merge(array, first, mid, last)
```

Merge Algorithm (incorrect)

```
merge(array, first, mid, last):
    // merge array[first to mid-1] and array[mid to last-1]
    leftpos = first
    rightpos = mid
    for newpos from 0 to last-first:
        if array[leftpos] ≤ array[rightpos]:
            newarray[newpos] = array[leftpos]
            leftpos++
        else:
            newarray[newpos] = array[rightpos]
            rightpos++
    copy newarray to array[first to last-1]
```

Problem?

- This algorithm starts correctly, but has an error as it finishes.
  - Eventually, one of the halves will empty.
- Then, the "if" will compare against ???
  - the element past the end of one of the halves
  - one of:



Solution

- Must prevent this: we can only look at the correct parts of the array.
- So, compare only until we reach the end of one half.
  - Then, just copy the rest over.

Corrected Merge Algorithm (1)

```
merge(array, first, mid, last):
    leftpos = first
    rightpos = mid
    newpos = 0
    while leftpos < mid and rightpos ≤ last-1:
        if array[leftpos] ≤ array[rightpos]:
            newarray[newpos] = array[leftpos]
            leftpos++; newpos++
        else:
            newarray[newpos] = array[rightpos]
            rightpos++; newpos++
```

(continues)

Corrected Merge Algorithm (2)

```
merge(array, first, mid, last):
    … // code from last slide

    while leftpos < mid:   // copy rest of left half (if any)
        newarray[newpos] = array[leftpos]
        leftpos++; newpos++
    while rightpos ≤ last-1:  // copy rest of right half (if any)
        newarray[newpos] = array[rightpos]
        rightpos++; newpos++

    copy newarray to array[first to (last-1)]
```

Example 1

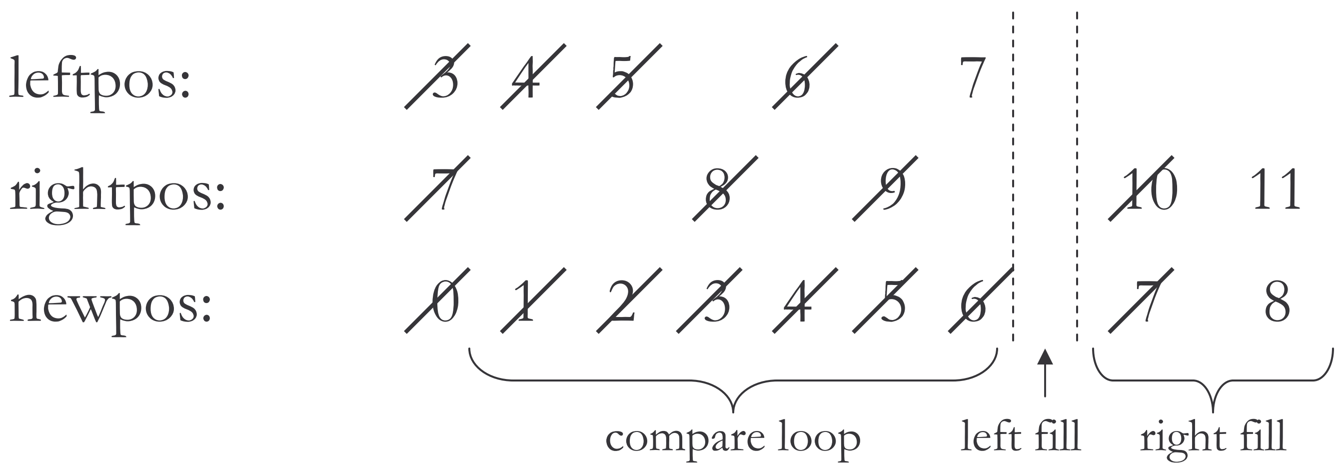
merge(array, 3, 7, 11):

array:

3	4	5	6	7	8	9	10
10	20	40	50	30	40	60	70

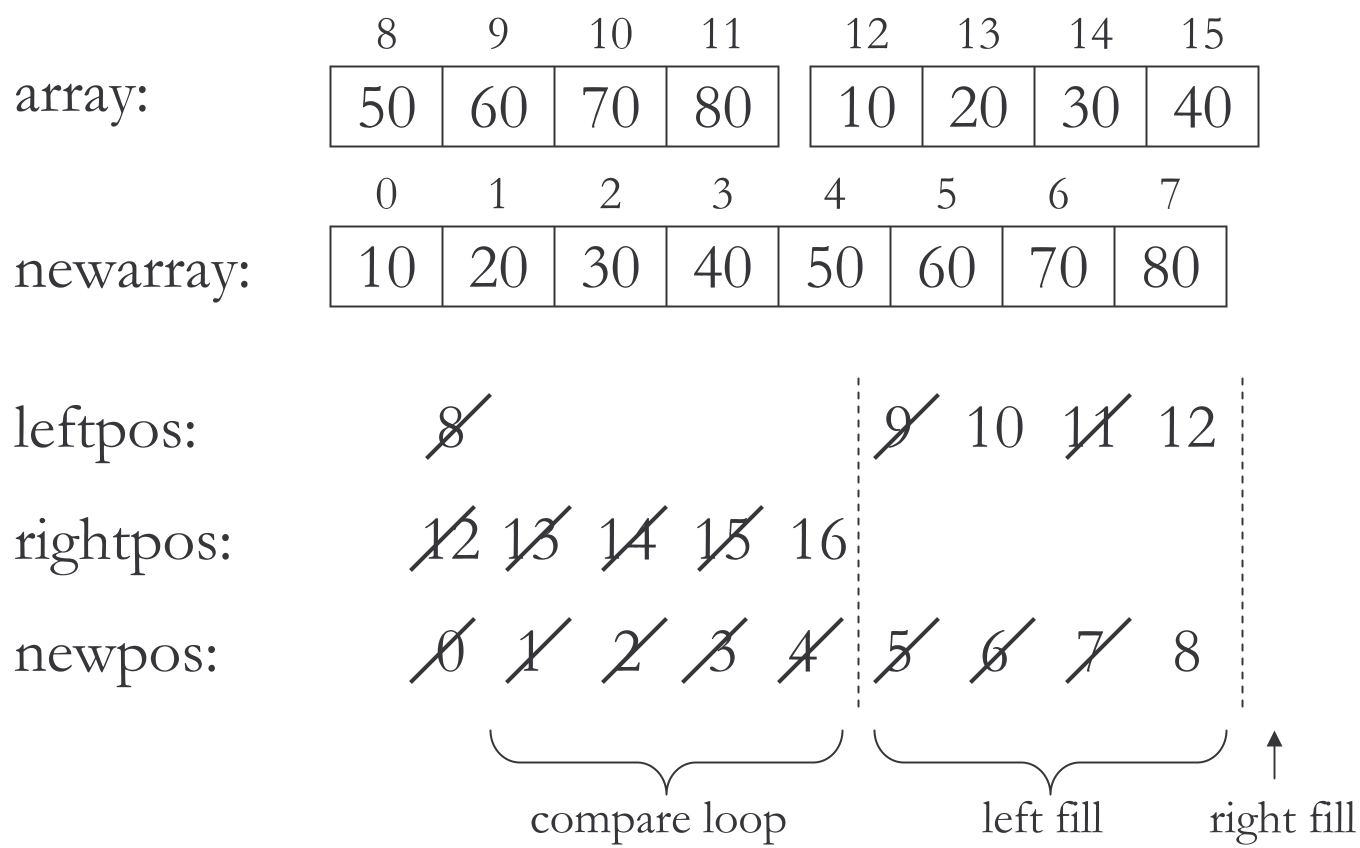
newarray:

0	1	2	3	4	5	6	7
10	20	30	40	40	50	60	70



Example 2

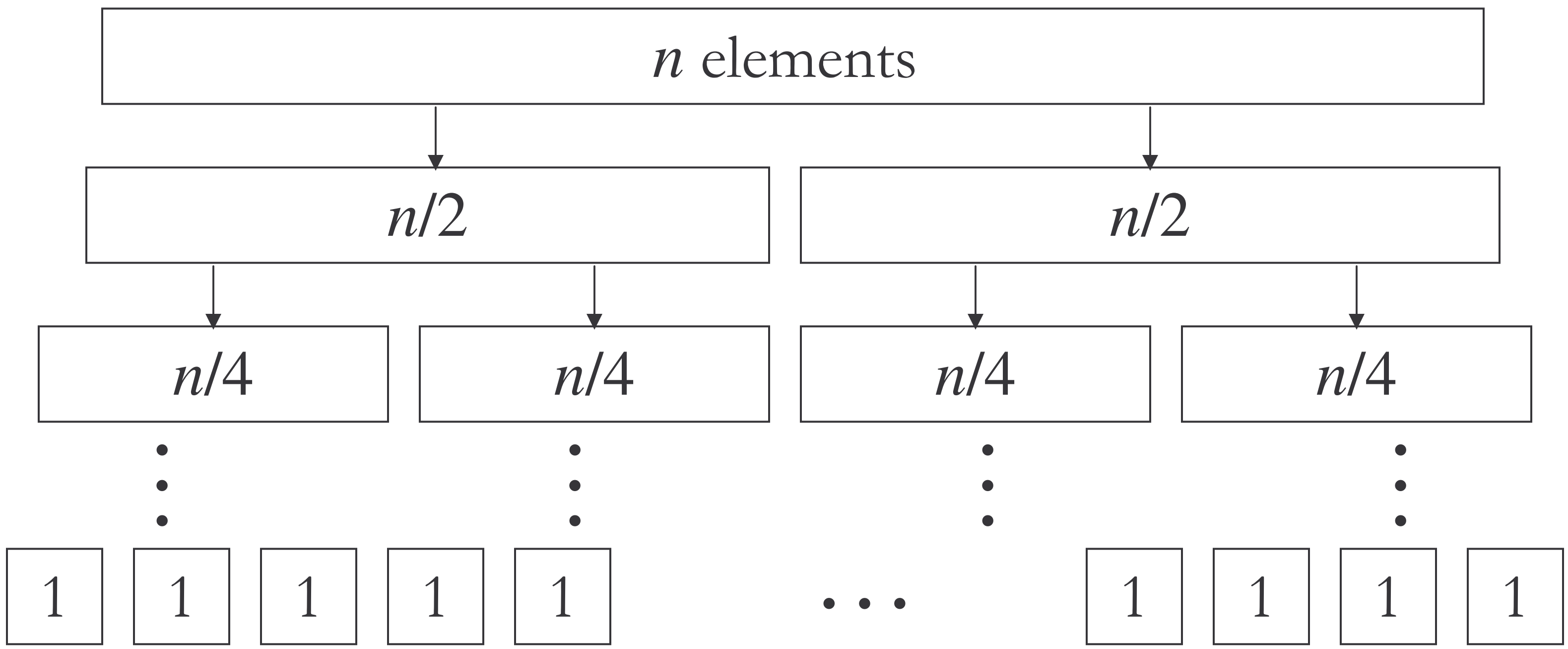
merge(array, 8, 12, 15):



Running Time

- What is merge sort's running time?
- recursive calls × work per call?
  - yes, but overly-simplified.
  - Work per call changes.
- We know: the merge algorithm takes O(m) work to merge a total of m elements.

Merge Sort Recursive Calls



- Each level has a total of *n* elements.

Running Time

- $O(n)$ total time to merge each level
- $O(\log n)$ levels
- Total time for merge sort: $O(n \cdot \log n)$
  - Much faster than insertion sort, which takes $O(n^2)$.
- In general, no sorting algorithm can do better than $O(n \cdot \log n)$.
  - There are some algorithms that are faster for limited cases.

In-place Sorting

- Merging requires extra storage.
  - an extra array with *n* elements (can be reused by all merges)
  - Insertion sort requires no extra storage (except a few numeric variables).
- An algorithm that uses at most O(1) extra storage is called “in-place”.
  - Insertion sort is in-place; merge sort isn’t.

Stable Sorting

- In merge sort, equal elements are kept in-order.
  - Think of sorting a spreadsheet with other columns.
  - Rows with equal values in the sort column stay in order.
- A "stable" sorting algorithm has this properly.
  - Our implementations of insertion sort & merge sort are both stable.