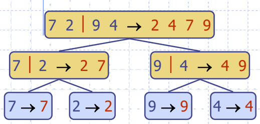
Lesson 10
Merge Sort: Collapsing Infinity To a Point



Wholeness of the Lesson

Merge Sort is a Divide and Conquer sorting algorithm that can sort lists in O(n logn) time, even in the worst case. The Divide and Conquer strategy is an example of the simple principle of "Do Less and Accomplish More."

Merge Sort 1

1

Divide-and-Conquer

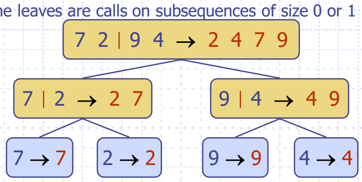
- Divide-and conquer is a general algorithm design strategy:
  - Divide: divide the input data S in two disjoint subsets S_1 and S_2
  - Recur: solve the subproblems associated with S_1 and S_2
  - Conquer: combine the solutions for S_1 and S_2 into a solution for S
- The base case for the recursion are typically subproblems of size 0 or 1

Merge Sort 2

2

Merge-Sort Tree

- An execution of merge-sort may be depicted by a binary tree
  - each node represents a recursive call of merge-sort and stores
    - unsorted sequence before the execution
    - its partition
    - sorted sequence at the end of the execution
  - the root is the initial call
  - the leaves are calls on subsequences of size 0 or 1

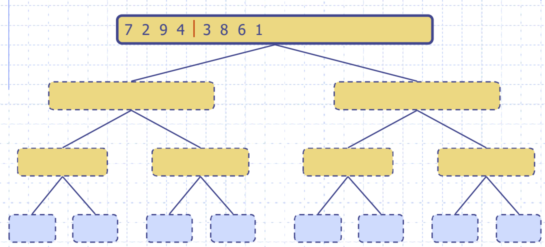


Merge Sort 3

3

Execution Example

- Partition

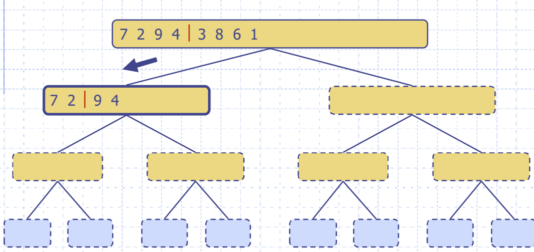


Merge Sort 4

4

Execution Example (cont.)

- Recursive call, partition

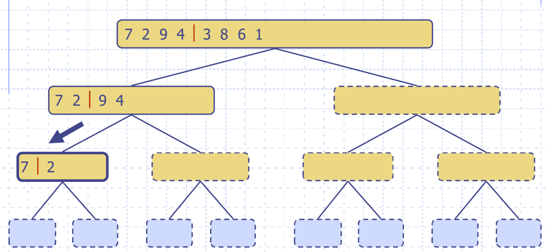


Merge Sort 5

5

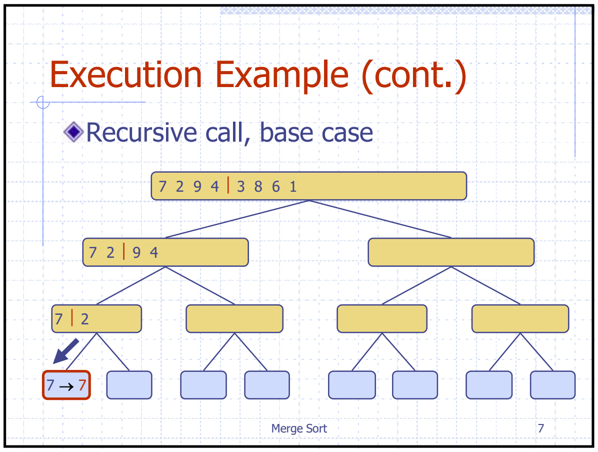
Execution Example (cont.)

- Recursive call, partition

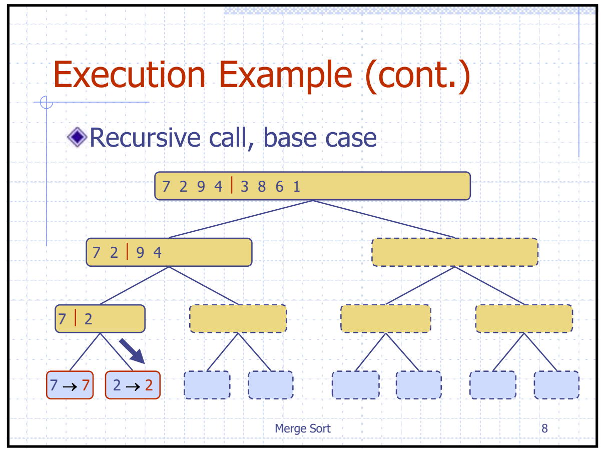


Merge Sort 6

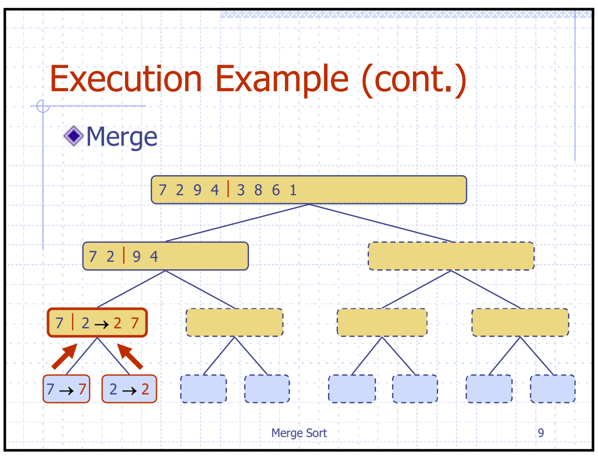
6


Execution Example (cont.)
Recursive call, base case
7 2 9 4 | 3 8 6 1
7 2 | 9 4
7 | 2
7 → 7
Merge Sort
7

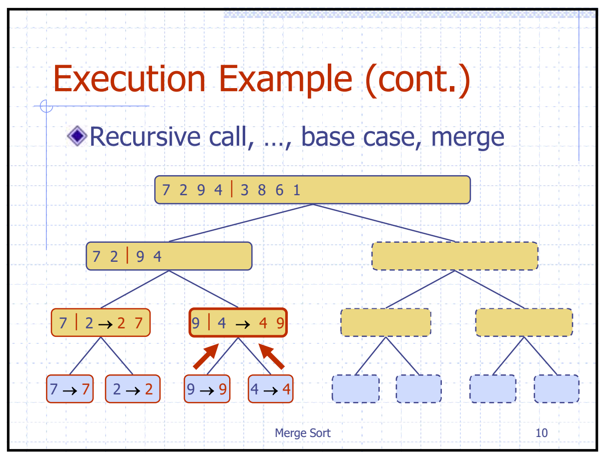
7


Execution Example (cont.)
Recursive call, base case
7 2 9 4 | 3 8 6 1
7 2 | 9 4
7 | 2
7 → 7
2 → 2
Merge Sort
8

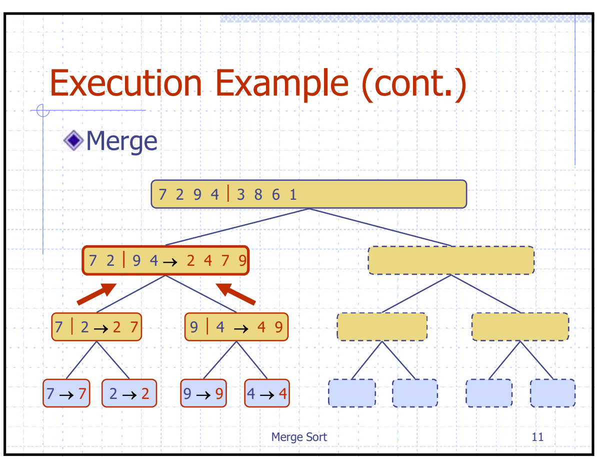
8


Execution Example (cont.)
Merge
7 2 9 4 | 3 8 6 1
7 2 | 9 4
7 | 2 → 2 7
7 → 7
2 → 2
Merge Sort
9

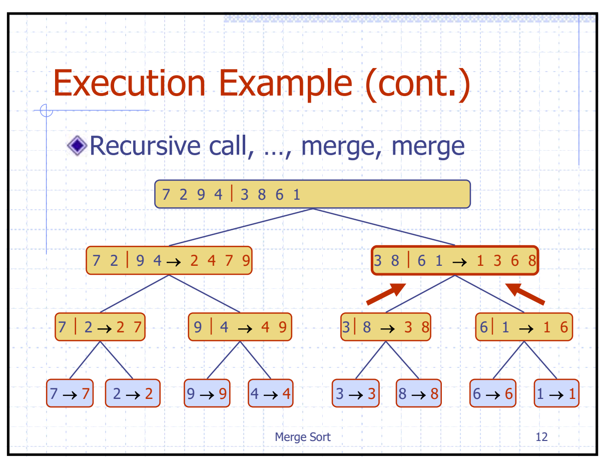
9


Execution Example (cont.)
Recursive call, …, base case, merge
7 2 9 4 | 3 8 6 1
7 2 | 9 4
7 | 2 → 2 7
9 | 4 → 4 9
7 → 7
2 → 2
9 → 9
4 → 4
Merge Sort
10

10


Execution Example (cont.)
Merge
7 2 9 4 | 3 8 6 1
7 2 | 9 4 → 2 4 7 9
7 | 2 → 2 7
9 | 4 → 4 9
7 → 7
2 → 2
9 → 9
4 → 4
Merge Sort
11

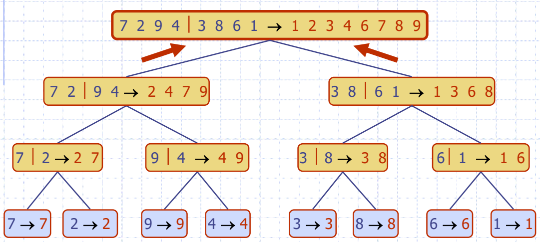
11


Execution Example (cont.)
Recursive call, …, merge, merge
7 2 9 4 | 3 8 6 1
7 2 | 9 4 → 2 4 7 9
3 8 | 6 1 → 1 3 6 8
7 | 2 → 2 7
9 | 4 → 4 9
3 | 8 → 3 8
6 | 1 → 1 6
7 → 7
2 → 2
9 → 9
4 → 4
3 → 3
8 → 8
6 → 6
1 → 1
Merge Sort
12

12

Execution Example (cont.)

- Merge



Merge Sort 13

13

Merge-Sort

- Merge-sort is a sorting algorithm based on the divide-and-conquer paradigm
- Merge-sort on an input sequence S with n integers consists of three steps:
  - Divide: partition S into two lists S_1 and S_2 of about $n/2$ elements each
  - Conquer: recursively sort S_1 and S_2
  - Combine: merge S_1 and S_2 into a single sorted list S

```
Algorithm mergeSort(S)
    Input List S with n integers
    Output List S sorted
    if S.size() > 1 then
        (S1, S2) ← partition(S)
        mergeSort(S1)
        mergeSort(S2)
        S ← merge(S1, S2, S)
    return S
```

Merge Sort 14

14

Merging Two Sorted Sequences

- The conquer step of merge-sort consists of merging two sorted lists A and B into a sorted list S containing the union of the elements of A and B
- Merging two sorted lists, each with $n/2$ elements and implemented by means of a doubly linked list, takes $O(n)$ time

```
Algorithm merge(A, B, S)
    Input Sorted lists A and B with n/2
        elements each and empty list S
    Output S contains sorted sequence of A ∪ B
    while A.size()>0 ∧ B.size()>0 do
      if ( B.first().element() <
                A.first().element() ) then
          S.insertLast(B.remove(B.first()))
      else
          S.insertLast(A.remove(A.first()))

    while A.size() > 0 do
        S.insertLast(A.remove(A.first()))

    while B.size() > 0 do
        S.insertLast(B.remove(B.first()))
    return S
```

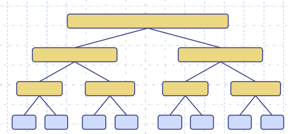
Merge Sort 15

15

Analysis of Merge-Sort

- The height h of the merge-sort tree is $O(\log n)$
  - at each recursive call we divide the sequence in half (n a power of 2)
- The overall amount of work done at each level is $O(n)$
- Thus, the total running time of merge-sort is $O(n \log n)$

depth	#seqs	size
0	1	n
1	2	$n/2$
i	2^i	$n/2^i$
…	…	…



Merge Sort 16

16

Merge-Sort of an Array

- Merge-sort of an array by partitioning into segments of the input array
- Merge-sort on an input sequence S with n integers consists of three steps:
  - Divide: partition S into two segments of about $n/2$ elements each (lo..mid) and (mid+1..hi)
  - Conquer: recursively sort the two segments
  - Combine: merges the two segments back into S in the merge step

```
Algorithm mergeSort(S, lo, hi, Temp)
    Input arrays S and Temp (work
        area), and indices lo, hi
    Output array S with elements
        between lo and hi in sorted order
    if hi - lo + 1 > 1 then
        mid ← floor((lo + hi)/2)
        mergeSort(S, lo, mid, Temp)
        mergeSort(S, mid+1, hi, Temp)
        merge(S, lo, mid, hi, Temp)
    return
```

Merge Sort 17

17

Merging Two Sorted Sequences

- The conquer step of merge-sort consists of merging two sorted segments of A back into A in sorted order
- Merging two sorted array segments, each with $n/2$ elements (where n=hi-lo+1) takes $O(n)$ time

```
Algorithm merge(A, lo, mid, hi, Temp)
    Input Sorted segments of array A between lo..mid and
        mid+1..hi and Temp array is working storage
    Output A contains elements sorted between lo..hi
    size ← hi - lo + 1
    t ← 0
    j ← lo
    k ← mid + 1
    while j ≤ mid ∧ k ≤ hi do
        if A[j] > A[k] then
            Temp[t] ← A[k]
            k ← k + 1
        else
            Temp[t] ← A[j]
            j ← j + 1
        t ← t + 1
    while j ≤ mid do
        Temp[t] ← A[j]; t ← t+1; k ← k+1;

    while k ≤ hi do
        Temp[t] ← A[k]; t ← t+1; k ← k+1;

    for i ← 0 to size - 1 do
        A[lo+i] ← Temp[i]
```

Merge Sort 18

18

Main Point

1. In merge-sort, the input is divided into two equal-sized subsequences, each of which is sorted separately. Then these sorted subsequences are merged together to form the sorted output.
Science of Consciousness: Creation arises from the collapse of the unbounded value of wholeness to a point; the re-emergence of wholeness results in the laws (algorithms of nature) that provide the balance, order, and efficiency in creation. Contact with this field improves the quality of life (order, balance, simplicity, efficiency) of the individual and society.

19

Summary of Sorting Algorithms

Algorithm	Time	Notes
selection-sort	$O(n^2)$	◆ slow ◆ in-place ◆ for small data sets (< 1K)
insertion-sort	$O(n^2)$	◆ slow ◆ in-place ◆ for small data sets (< 1K)
heap-sort	$O(n \log n)$	◆ fast ◆ in-place ◆ for large data sets (1K — 1M)
merge-sort	$O(n \log n)$	◆ fast ◆ sequential data access ◆ for huge data sets (> 1M)

20

Connecting the Parts of Knowledge With the Wholeness of Knowledge

Merge Sort

1. Simple sorting algorithms examine each successive element in the input array, then perform a further step to place this element in an already sorted area. This style of sorting involves an *incremental unfoldment.*
2. MergeSort proceeds by repeatedly collapsing (reducing) the wholeness of the current input into smaller parts, processing them separately, then synthesizing the parts into a sorted whole. This approach yields a much faster sorting algorithm.
3. *Transcendental Consciousness* is the silent field of *infinite correlation*, where "an impulse anywhere is an impulse everywhere," a field of "frictionless flow".
4. *Impulses within the Transcendental field.* Established in the transcendental field, action reaches fulfillment with minimum effort. Yoga is "skill in action" – efficiency in action, "doing less, accomplishing more", whereby little needs to be done to accomplish great goals.
5. *Wholeness moving within itself.* In Unity Consciousness, the field of action effortlessly unfolds as the play of one's own Self, one's own pure consciousness.

21