

CSE 326: Data Structures

Mergesort

Brian Curless
Spring 2008

Announcements (5/12/08)

- Homework due on Friday at the beginning of class.
- Reading for this lecture: Chapter 7.

2

Stability

A sorting algorithm is **stable** if:

- Items in the input with the same value end up in the same order as when they began.

Input		Unstable sort		Stable Sort	
Adams	1	Adams	1	Adams	1
Black	2	Smith	1	Smith	1
Brown	4	Washington	2	Black	2
Jackson	2	Jackson	2	Jackson	2
Jones	4	Black	2	Washington	2
Smith	1	White	3	White	3
Thompson	4	Wilson	3	Wilson	3
Washington	2	Thompson	4	Brown	4
White	3	Brown	4	Jones	4
Wilson	3	Jones	4	Thompson	4

3

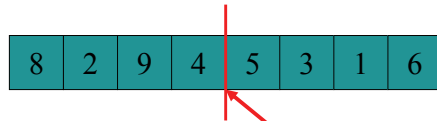
[Sedgewick]

“Divide and Conquer”

- Very important strategy in computer science:
 - Divide problem into smaller parts
 - Independently solve the parts
 - Combine these solutions to get overall solution
- **Idea 1**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves → known as **Mergesort**
- **Idea 2**: Partition array into small items and large items, then recursively sort the two sets → known as **Quicksort**

4

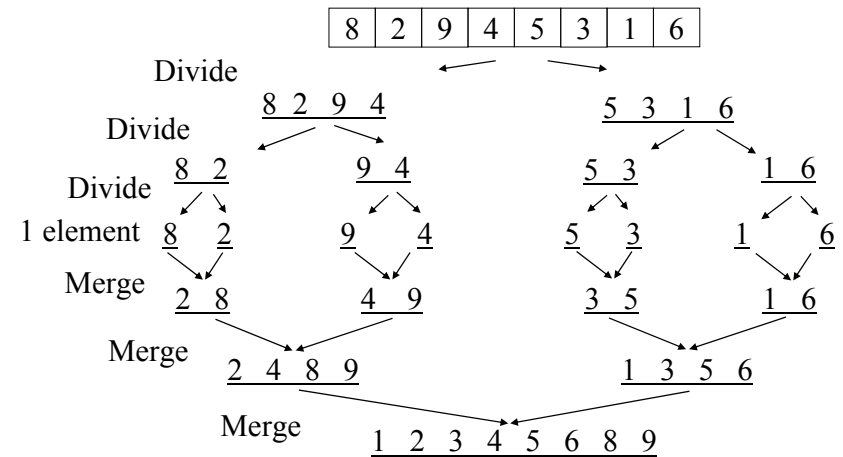
Mergesort



- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)
- Merge two halves together

5

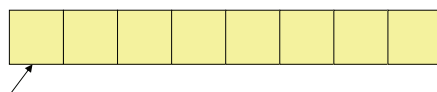
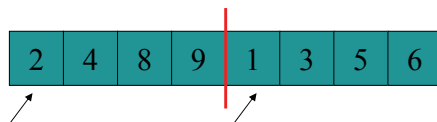
Mergesort Example



6

Merging: Two Pointer Method

- The merging requires an auxiliary array.

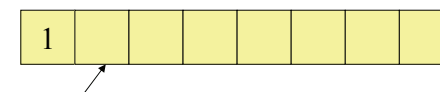
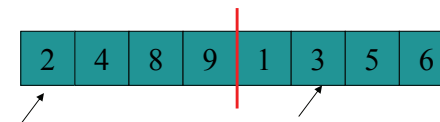


Auxiliary array

7

Merging: Two Pointer Method

- The merging requires an auxiliary array.

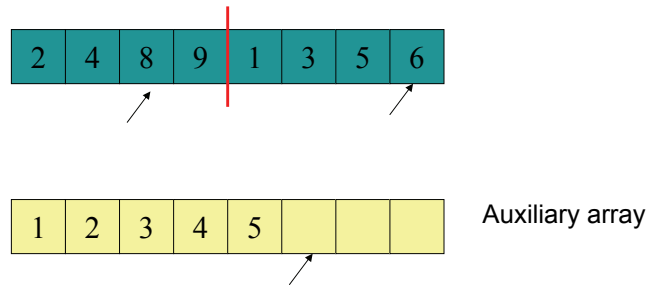


Auxiliary array

8

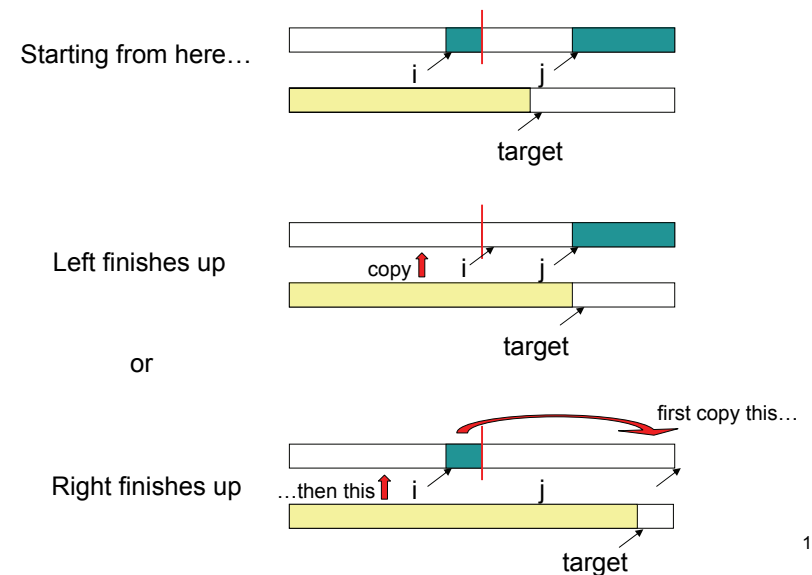
Merging: Two Pointer Method

- The merging requires an auxiliary array.



9

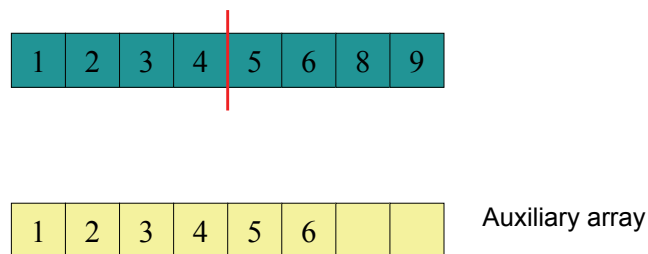
Merging: Finishing Up



10

Merging: Two Pointer Method

- Final result



Complexity?

Stability?

11

Merging

```

Merge(A[], Temp[], left, mid, right) {
    Int i, j, k, l, target
    i = left
    j = mid + 1
    target = left
    while (i ≤ mid && j ≤ right) {
        if (A[i] ≤ A[j])
            Temp[target] = A[i++]
        else
            Temp[target] = A[j++]
        target++
    }
    if (i > mid) //left completed//
        for (k = left to target-1)
            A[k] = Temp[k];
    if (j > right) //right completed//
        k = mid
        l = right
        while (k ≥ i)
            A[l--] = A[k--]
        for (k = left to target-1)
            A[k] = Temp[k]
    }

```

12

Recursive Mergesort

```

MainMergesort(A[1..n], n) {
    Array Temp[1..n]
    Mergesort(A, Temp, 1, n)
}

Mergesort(A[], Temp[], left, right) {
    if (left < right) {
        mid = (left + right)/2
        Mergesort(A, Temp, left, mid)
        Mergesort(A, Temp, mid+1, right)
        Merge(A, Temp, left, mid, right)
    }
}
    
```

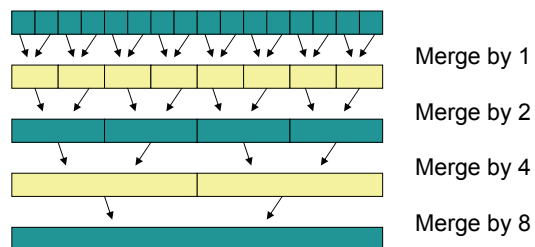
What is the recurrence relation?

13

Mergesort: Complexity

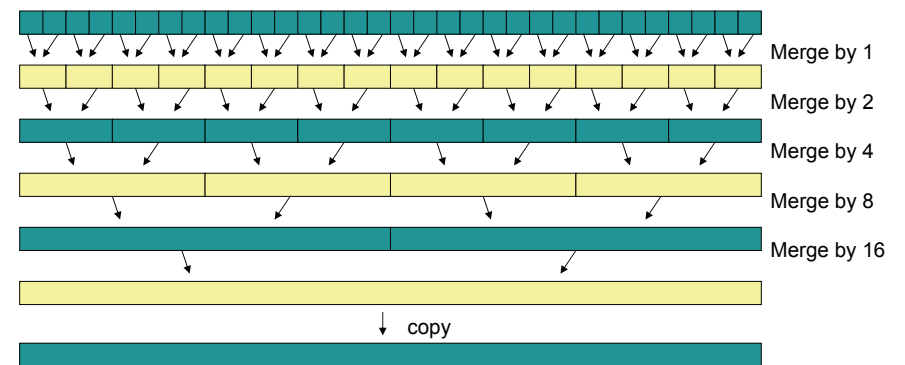
14

Iterative Mergesort



15

Iterative Mergesort



Iterative Mergesort reduces copying.
Complexity?

16

Properties of Mergesort

- In-place?
- Stable?
- Sorted list complexity?
- Nicely extends to handle linked lists.
- Multi-way merge is basis of big data sorting.
- Java uses Mergesort on Collections and on Arrays of Objects.