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.

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.

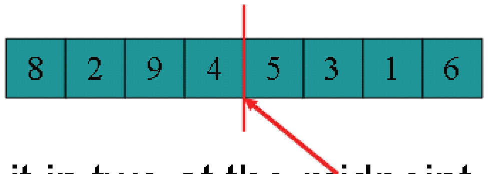
Input		**Unstable sort**		**Stable Sort**	
Adams	1	Adams	1	Adams	1
Black	2	Smith	1	Smith	1
Brown	4	Washington	2	Black	2
Jackson	2	Jackson	2	Jackson	2
Jones	4	Black	2	Washington	2
Smith	1	White	3	White	3
Thompson	4	Wilson	3	Wilson	3
Washington	2	Thompson	4	Brown	4
White	3	Brown	4	Jones	4
Wilson	3	Jones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

[Sedgewick]

"Divide and Conquer"

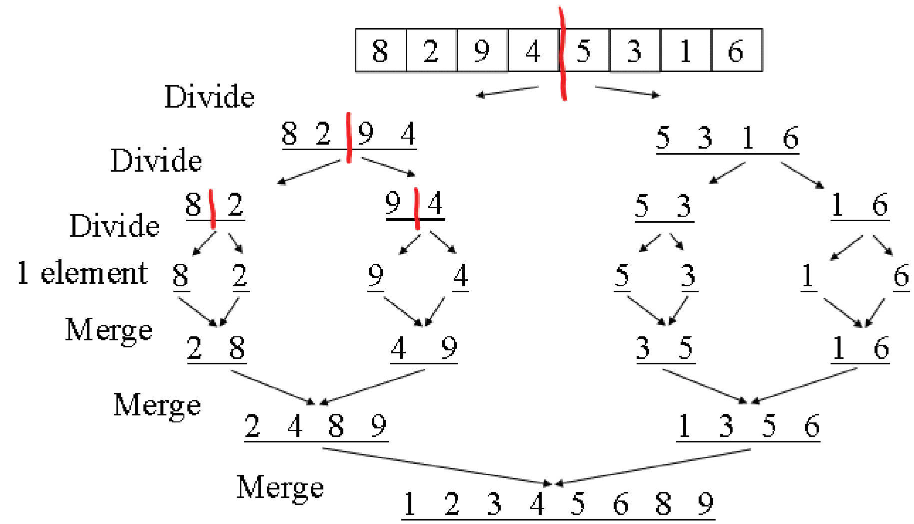
- Very important strategy in computer science:
  - Divide problem into smaller parts
  - Independently solve the parts
  - Combine these solutions to get overall solution
- **Idea 1**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves → known as Mergesort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

Mergesort



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

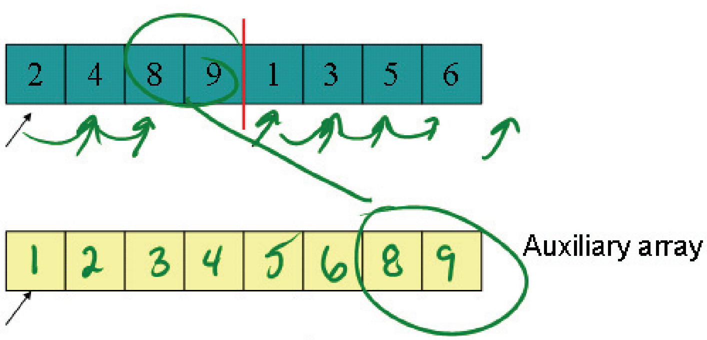
Mergesort Example



Merging: Two Pointer Method

(Two finger method)

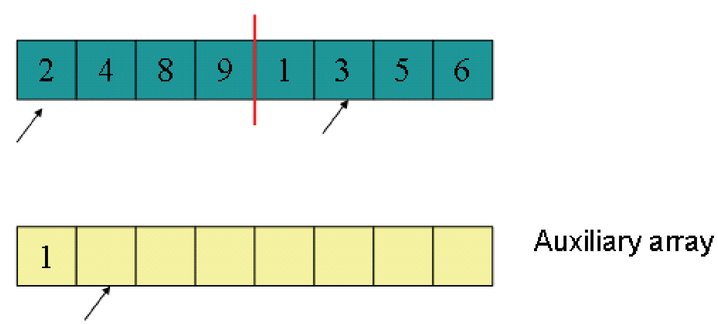
- The merging requires an auxiliary array.



$O(n)$

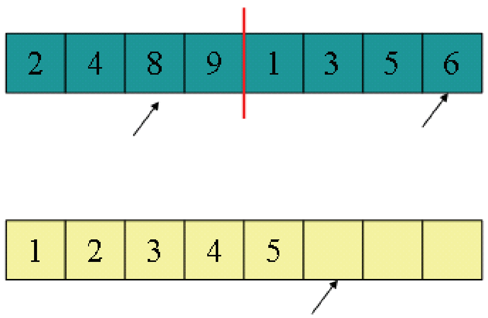
Merging: Two Pointer Method

- The merging requires an auxiliary array.



Merging: Two Pointer Method

- The merging requires an auxiliary array.



Auxiliary array

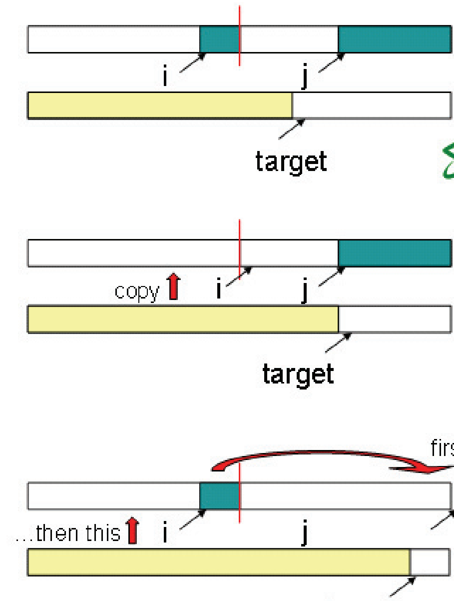
Merging: Finishing Up

Starting from here…

Left finishes up

or

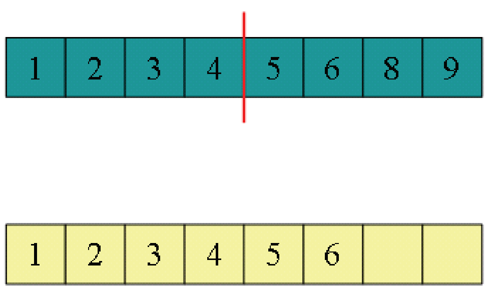
Right finishes up



Sorted arrays:
N/2
comparisons

Merging: Two Pointer Method

- Final result



Auxiliary array

Complexity? O(n)   Stability? Y if left wins ties

Merging

for stability

```
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]
}
```

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?

$T(1)=a \qquad T(n)=2T(n/2)+bn+c$

Mergesort: Complexity

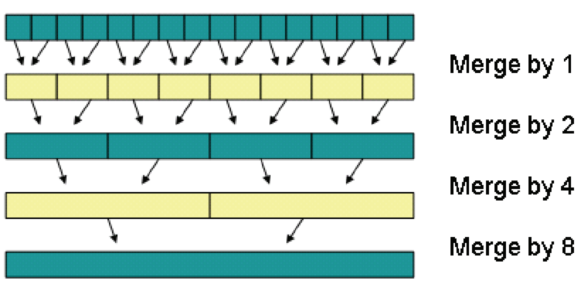
$T(1)=a$

$$
\begin{aligned}
T(n) &= 2T(n/2)+bn+c \\
&= 2\left[2T(n/4)+b\tfrac{n}{2}+c\right]+bn+c \\
&= 2\left[2\left[2T(n/8)+b\tfrac{n}{4}+c\right]+b\tfrac{n}{2}+c\right]+bn+c \\
&= 2^3T(n/2^3)+bn+bn+bn+2^2c+2^1c+2^0c \\
&\vdots \\
&= 2^kT(n/2^k)+bkn+\sum_{i=0}^{k-1}2^ic
\end{aligned}
$$

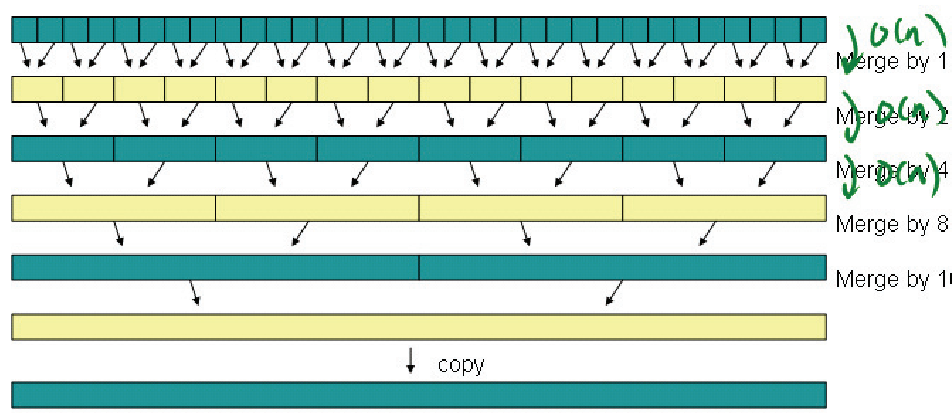
Assume $n=2^k \quad k=\log_2 n$

$$
\begin{aligned}
T(n) &= nT(1)+bn\log_2 n+(\underbrace{2^k}_{n}-1)c \\
&= an+bn\log_2 n+cn-c \in O(n\log n)
\end{aligned}
$$

Iterative Mergesort



Iterative Mergesort



Iterative Mergesort reduces copying.

Complexity? $O(n\log n)$

Properties of Mergesort

- In-place? N (not easy to do...)
- Stable? Y
- Sorted list complexity? O(n log n) .. O(n) w/ neat trick
- 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.