Algorithms and Data Structures CS 372

Merge Sort

(Based on slides by M. Nicolescu)

The Sorting Problem

- **Input:**
  - A sequence of n numbers $a_1, a_2, \ldots, a_n$
- **Output:**
  - A permutation (reordering) $a_1', a_2', \ldots, a_n'$ of the input sequence such that $a_1' \leq a_2' \leq \cdots \leq a_n'$

2

Insertion Sort - Summary

- Idea: like sorting a hand of playing cards
  - Start with an empty left hand and the cards facing down on the table.
  - Remove one card at a time from the table, and insert it into the correct position in the left hand
- Advantages
  - Good running time for "almost sorted" arrays $\Theta(n)$
- Disadvantages
  - $\Theta(n^2)$ running time in worst and average case

3

Divide-and-Conquer

- **Divide** the problem into a number of subproblems
  - Similar sub-problems of smaller size
- **Conquer** the sub-problems
  - Solve the sub-problems recursively
  - Sub-problem size small enough ⇒ solve the problems in straightforward manner
- **Combine** the solutions to the sub-problems
  - Obtain the solution for the original problem

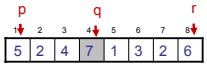
4

Merge Sort Approach

- To sort an array $A[p \,..\, r]$:
- **Divide**
  - Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each
- **Conquer**
  - Sort the subsequences recursively using merge sort
  - When the size of the sequences is 1 there is nothing more to do
- **Combine**
  - Merge the two sorted subsequences

5

Merge Sort



Alg.: MERGE-SORT(A, p, r)

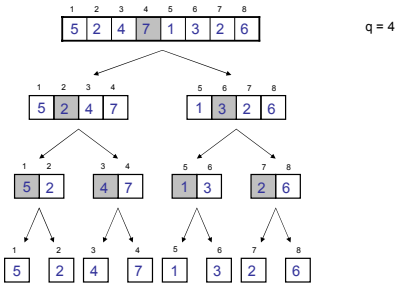
```
if p < r                          ▷ Check for base case
  then q ← ⌊(p + r)/2⌋            ▷ Divide
       MERGE-SORT(A, p, q)        ▷ Conquer
       MERGE-SORT(A, q + 1, r)    ▷ Conquer
       MERGE(A, p, q, r)          ▷ Combine
```

- Initial call: MERGE-SORT(A, 1, n)

6

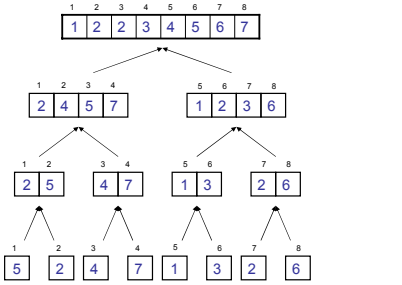
Example – n Power of 2

Example



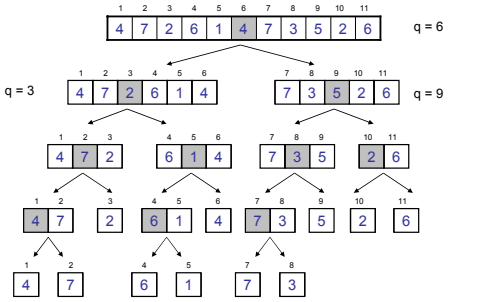
7

Example – n Power of 2



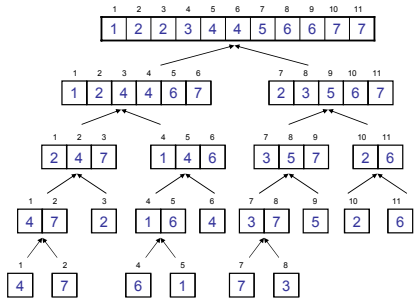
8

Example – n Not a Power of 2

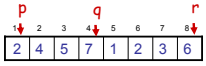


9

Example – n Not a Power of 2



Merging

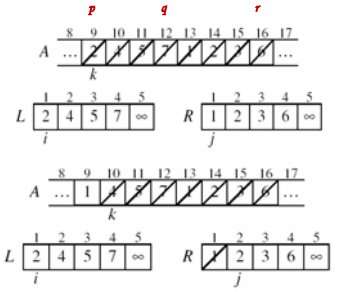


- **Input:** Array A and indices p, q, r such that $p \le q < r$
  - Subarrays $A[p\,..\,q]$ and $A[q+1\,..\,r]$ are sorted
- **Output:** One single sorted subarray $A[p\,..\,r]$

Merging

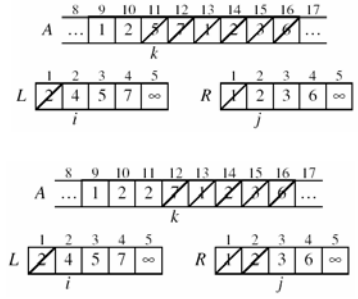
- Idea for merging:
  - Two piles of sorted cards
    - Choose the smaller of the two top cards
    - Remove it and place it face-down in the output pile
  - Repeat the process until one pile is empty
  - Take the remaining input pile and place it face-down onto the output pile

Example: MERGE(A, 9, 12, 16)



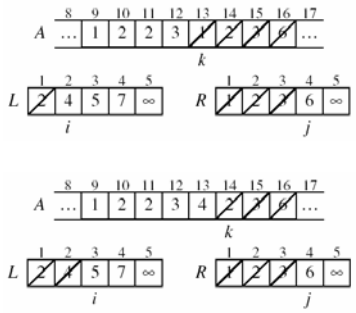
13

Example: MERGE(A, 9, 12, 16)



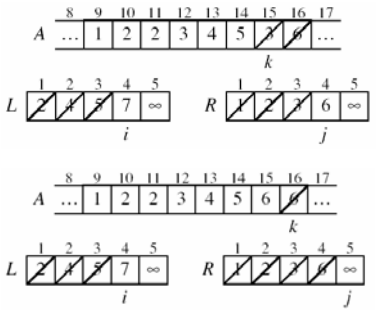
14

Example (cont.)

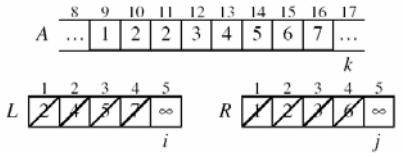


15

Example (cont.)



Example (cont.)

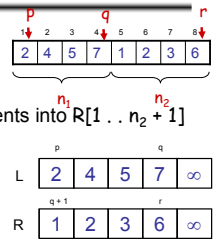


Done!

Merge - Pseudocode

Alg.: MERGE(A, p, q, r)

1. Compute n_1 and n_2
2. Copy the first n_1 elements into $L[1 . . n_1 + 1]$ and the next n_2 elements into $R[1 . . n_2 + 1]$
3. $L[n_1 + 1] \leftarrow \infty$; $R[n_2 + 1] \leftarrow \infty$
4. $i \leftarrow 1$; $j \leftarrow 1$
5. **for** $k \leftarrow p$ **to** r
6. **do if** $L[i] \le R[j]$
7. **then** $A[k] \leftarrow L[i]$
8. $i \leftarrow i + 1$
9. **else** $A[k] \leftarrow R[j]$
10. $j \leftarrow j + 1$



Running Time of Merge

- Initialization (copying into temporary arrays):
  - $\Theta(n_1 + n_2) = \Theta(n)$
- Adding the elements to the final array (the last **for** loop):
  - n iterations, each taking constant time $\Rightarrow \Theta(n)$
- Total time for Merge:
  - $\Theta(n)$

19

Sorting

- Insertion sort
  - Design approach: incremental
  - Sorts in place: Yes
  - Best case: $\Theta(n)$
  - Worst case $\Theta(n^2)$
- Merge Sort
  - Design approach: divide and conquer
  - Sorts in place: No
  - Running time: *Let's see!!*

20

Merge Sort Approach

- To sort an array $A[p \,..\, r]$:
- **Divide**
  - Divide the n-element sequence to be sorted into two subsequences of $n/2$ elements each
- **Conquer**
  - Sort the subsequences recursively using merge sort
  - When the size of the sequences is 1 there is nothing more to do
- **Combine**
  - Merge the two sorted subsequences

21

Analyzing Divide-and-Conquer Algorithms

- The recurrence is based on the three steps of the paradigm:
  - $T(n)$ – running time on a problem of size n
  - **Divide** the problem into a subproblems, each of size n/b: takes $D(n)$
  - **Conquer** (solve) the subproblems $aT(n/b)$
  - **Combine** the solutions $C(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le c \\ aT(n/b) + D(n) + C(n) & \text{otherwise} \end{cases}$$

22

MERGE-SORT Running Time

- **Divide:**
  - compute q as the average of p and r: $D(n) = \Theta(1)$
- **Conquer:**
  - recursively solve 2 subproblems, each of size $n/2$
    $\Rightarrow 2T(n/2)$
- **Combine:**
  - MERGE on an n-element subarray takes $\Theta(n)$ time
    $\Rightarrow C(n) = \Theta(n)$

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

23

Solve the Recurrence

$$T(n) = \begin{cases} c & \text{if } n = 1 \\ 2T(n/2) + cn & \text{if } n > 1 \end{cases}$$

Use Master's Theorem (Chapter 4.3):

Compare n with $f(n) = cn$
Case 2: $T(n) = \Theta(n \lg n)$

24

Merge Sort - Discussion

- Running time insensitive of the input
- Advantages:
  - Guaranteed to run in $\Theta(n \lg n)$
- Disadvantage
  - Requires extra space $\approx N$
- Applications
  - Maintain a large ordered data file
  - How would you use Merge sort to do this?

25

Sorting Challenge 1

Problem: Sort a huge randomly-ordered file of small records

Application: Process transaction record for a phone company

Which sorting method to use?

A. Mergesort guaranteed to run in time $\sim N \lg N$
B. Insertion sort

26

Sorting Huge, Randomly - Ordered Files

- Insertion sort?
  - NO, quadratic time for randomly-ordered keys
- Mergesort?
  - YES, it is designed for this problem

27

Sorting Challenge 2

Problem: sort a file that is already almost in order

Applications:

- Re-sort a huge database after a few changes
- Doublecheck that someone else sorted a file

Which sorting method to use?

A. Mergesort, guaranteed to run in time ~NlgN
B. Insertion sort

28

Sorting Files That are Almost in Order

- Insertion sort?
  - YES, takes linear time for most definitions of "almost in order"
- Mergesort?
  - Probably not: insertion sort simpler and faster

29