CS 401: Computer Algorithm I

Divide and Conquer

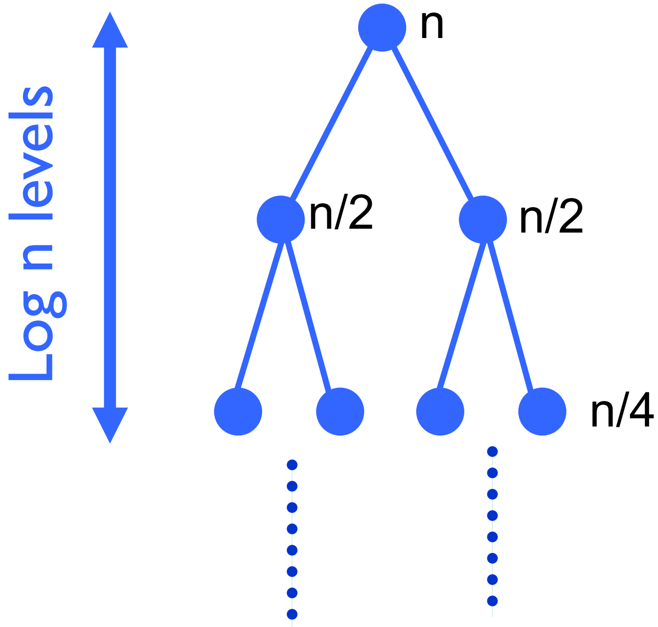
Xiaorui Sun

Divide and Conquer

Divide and Conquer

Divide: We reduce a problem to several subproblems.

Typically, each sub-problem is at most a constant c < 1 fraction of the size of the original problem



Conquer: Recursively solve each subproblem

Combine: Merge the solutions

Examples:

- Mergesort, Binary Search, Strassen's Algorithm,

Mergesort

Sorting

Sorting. Given n elements, rearrange in ascending order.

Obvious sorting applications.
- List files in a directory.
- Organize a playlist.
- List names in address book.
- Display Google PageRank results.

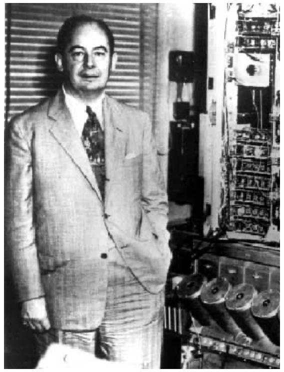
Problems become easier once sorted.
- Find the median.
- Greedy algorithms.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

Non-obvious sorting applications.
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
- . . .

Mergesort

Mergesort

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.



Jon von Neumann (1945)

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R	I	T	H	M	S	divide	$O(1)$
A	G	L	O	R	H	I	M	S	T	sort	$2T(n/2)$

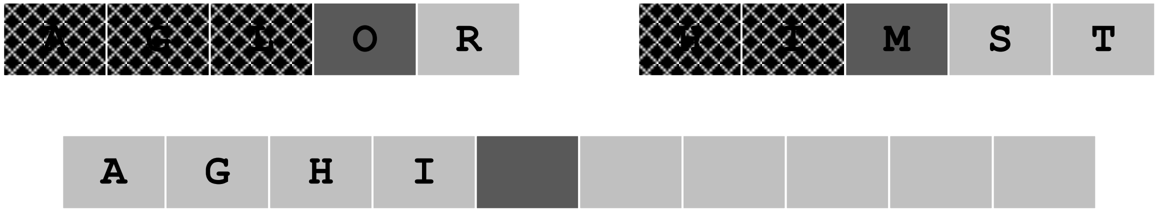
A	G	H	I	L	M	O	R	S	T	merge	$O(n)$

Merging

Merging: Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?

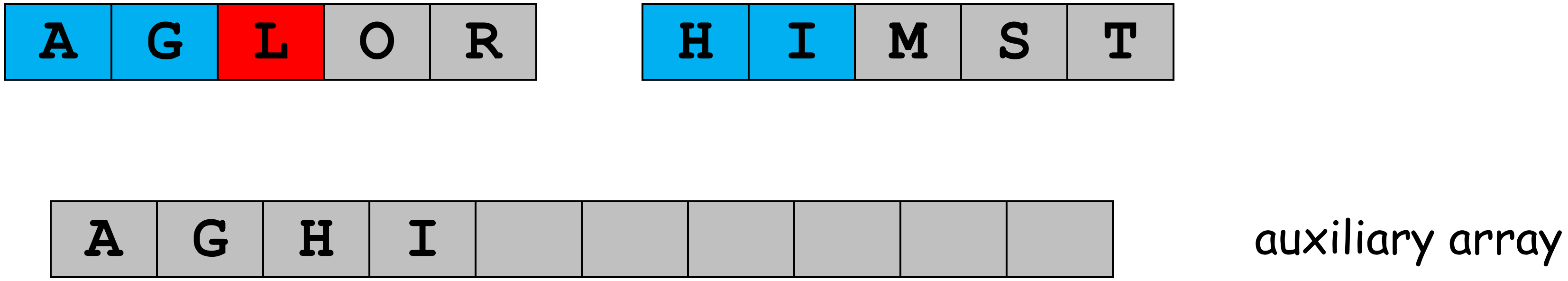
- Linear number of comparisons.
- Use auxiliary array.



Merging

Merge.

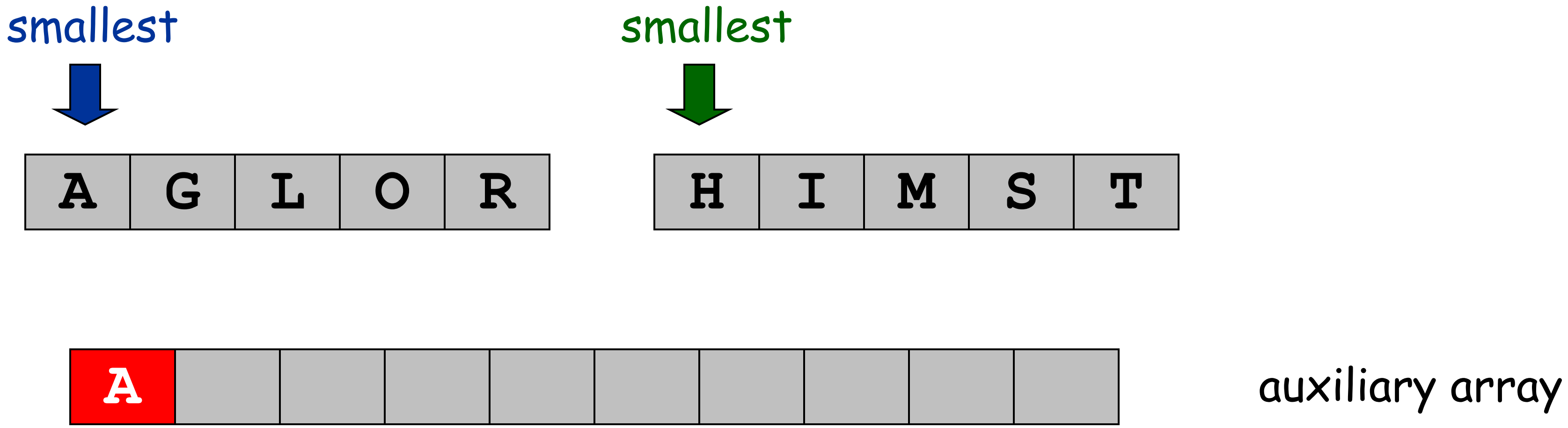
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

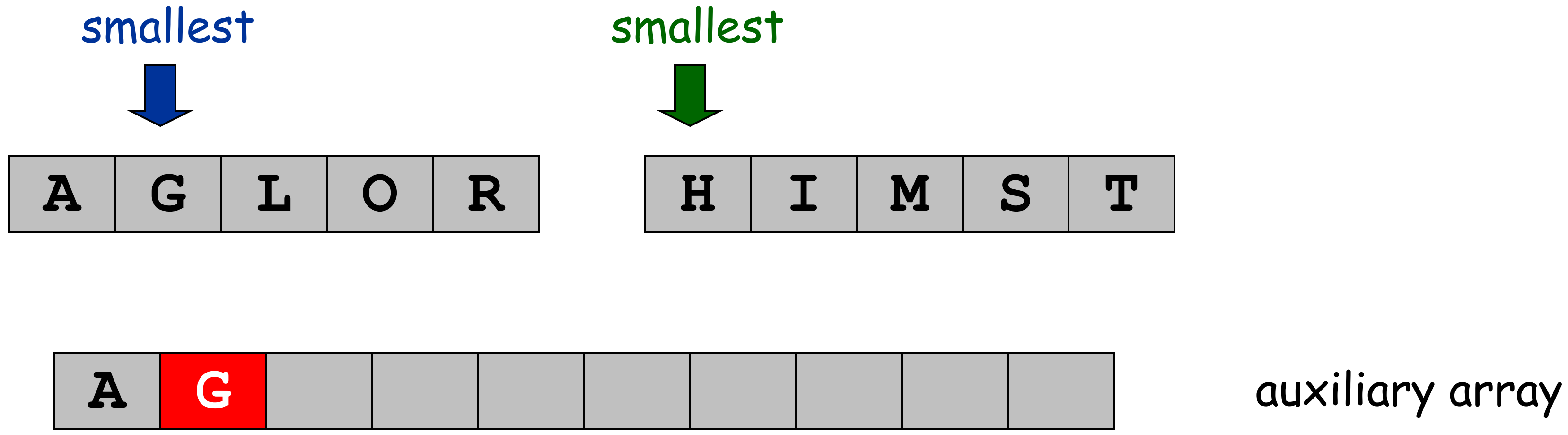
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

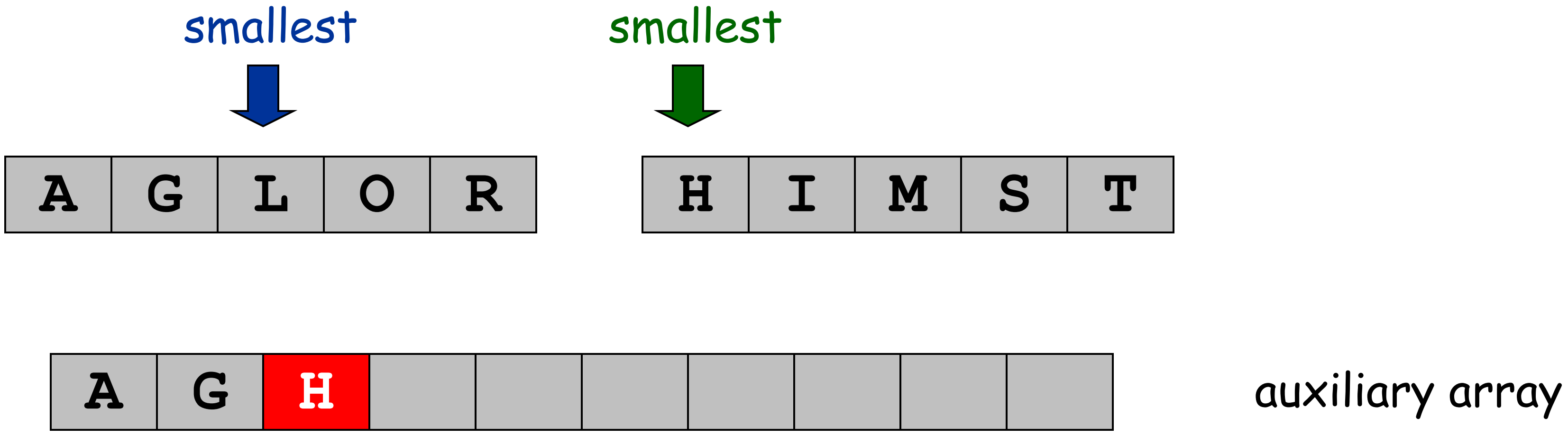
Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

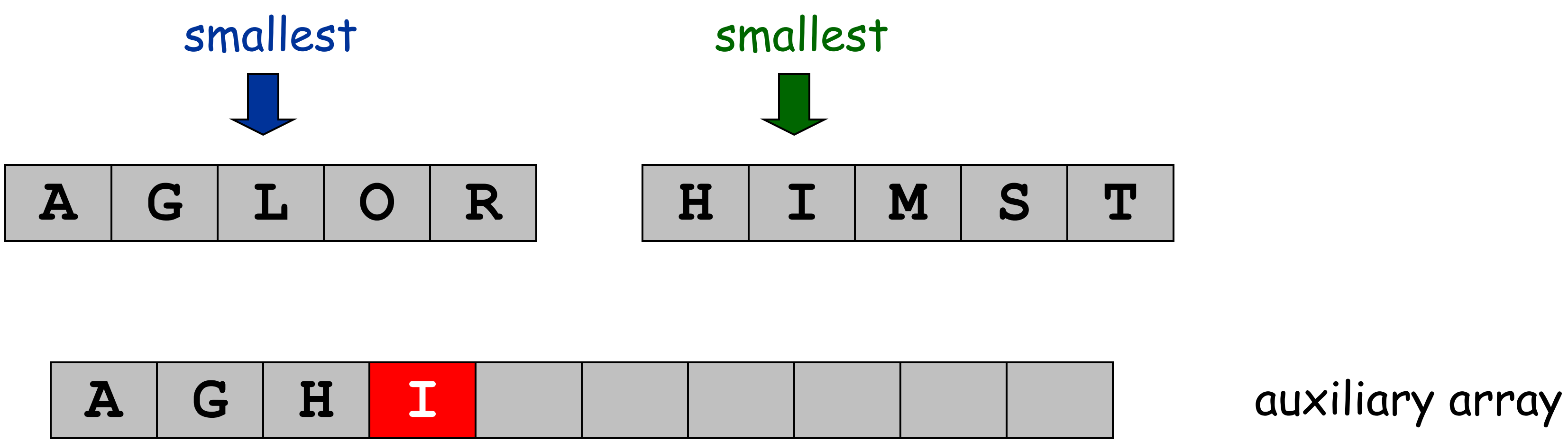
Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

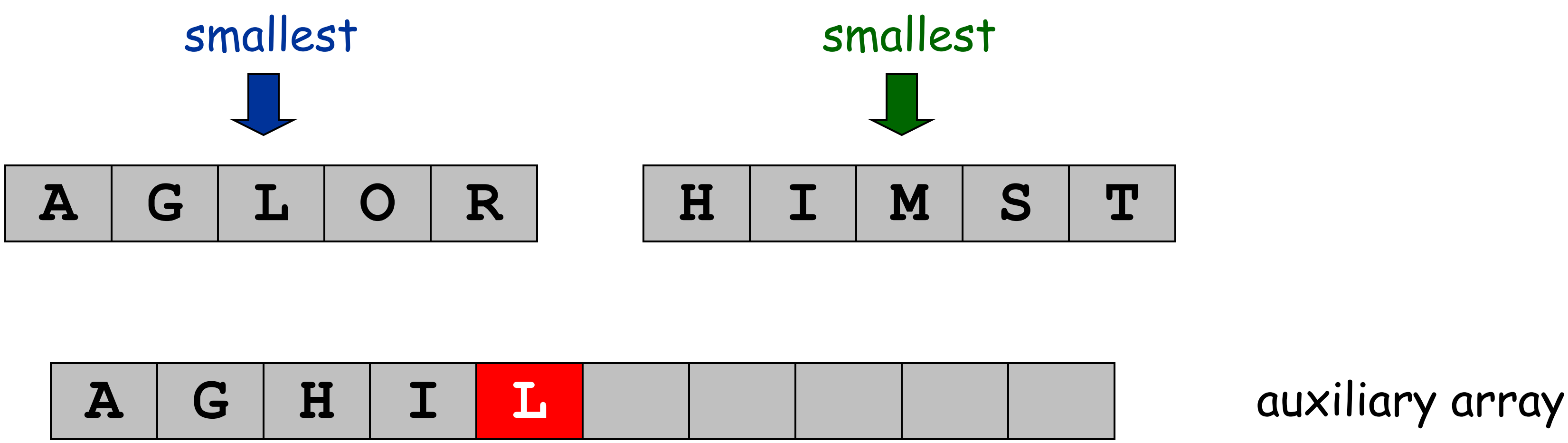
Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

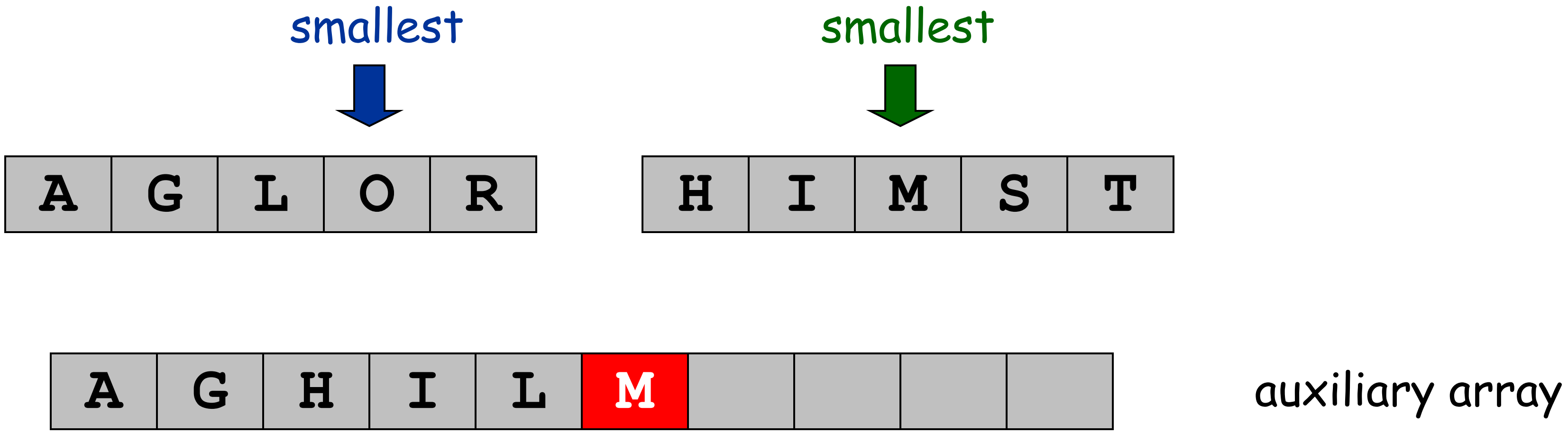
Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

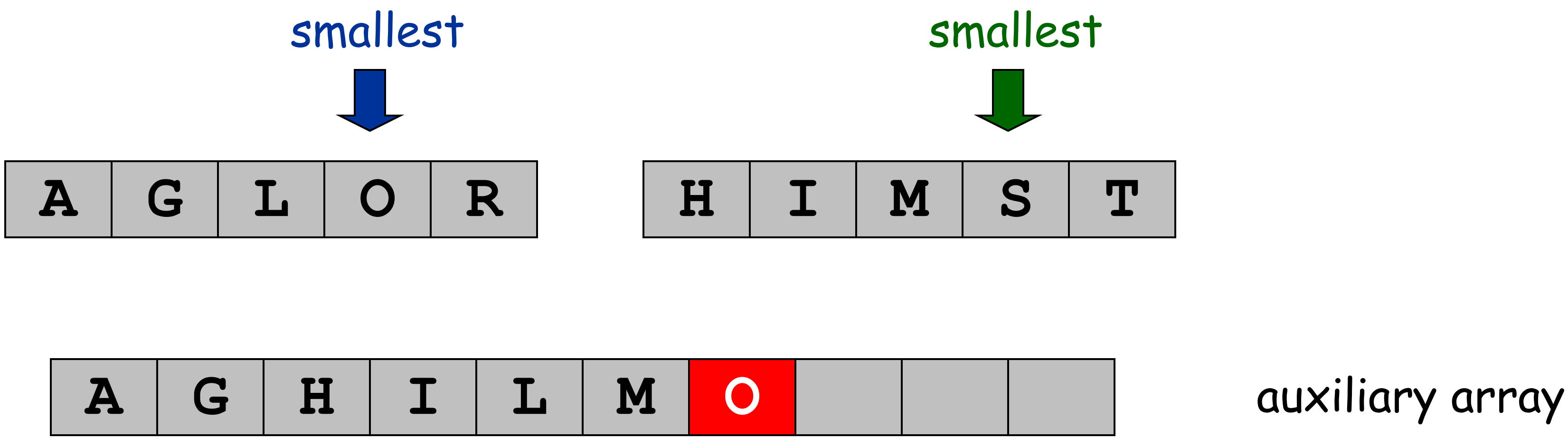
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

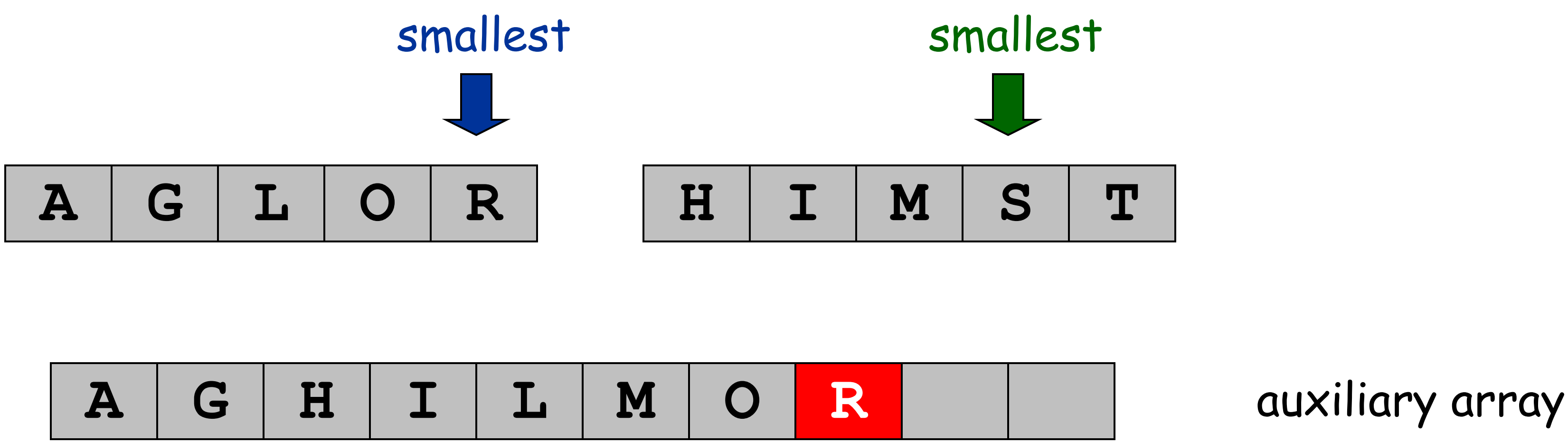
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

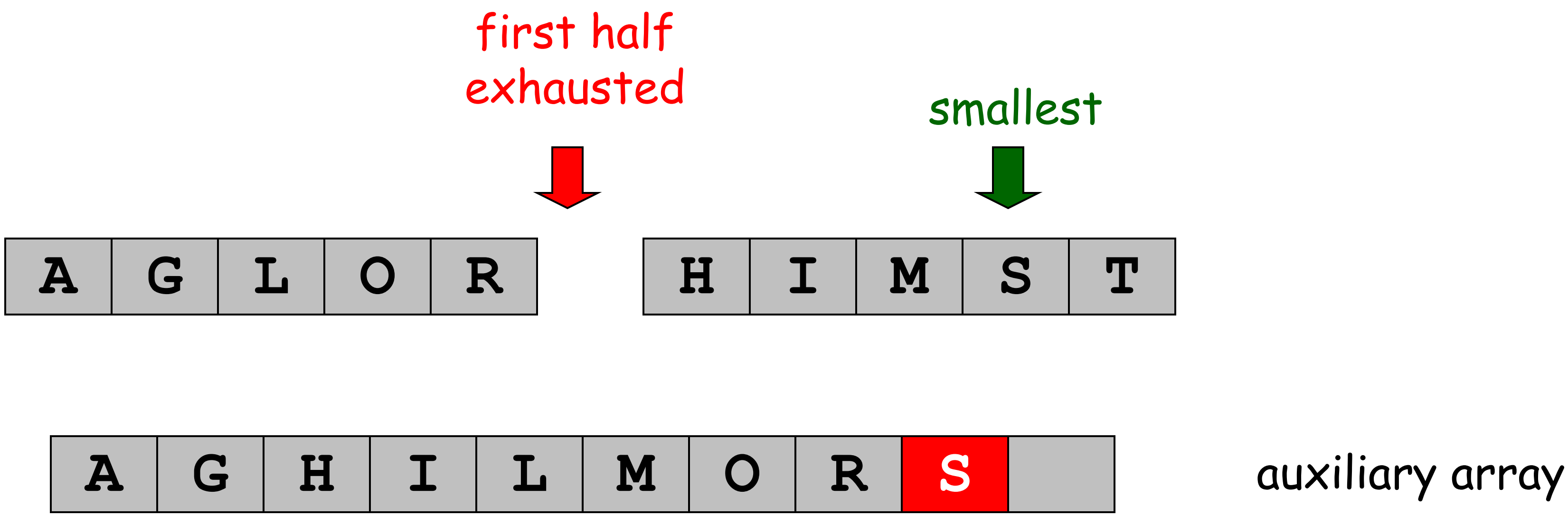
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

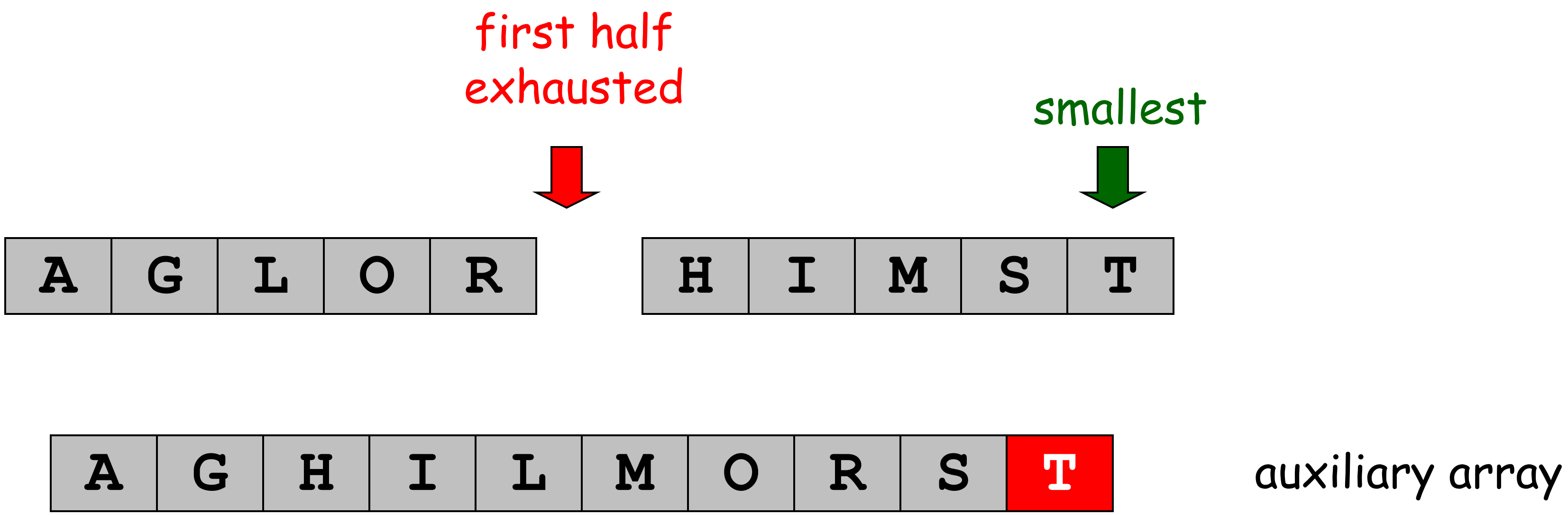
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

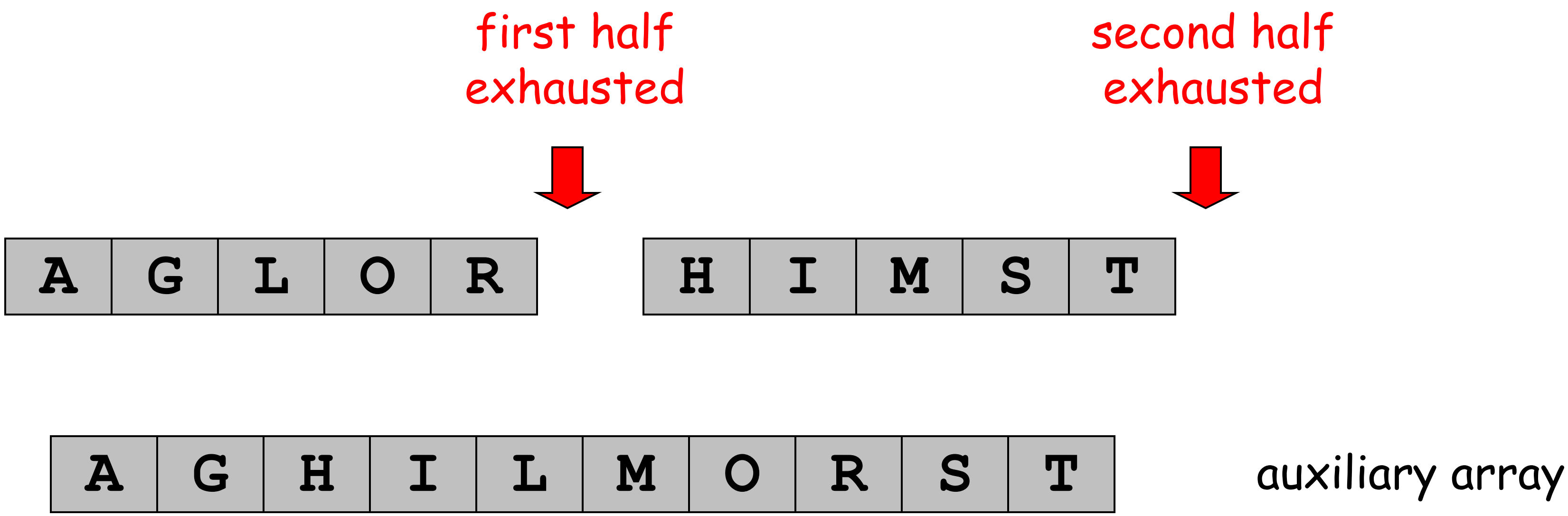
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



A Useful Recurrence Relation

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

$$T(n) \le \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = O(n \log_2 n)$.

Why?
We will discuss the solution of recurrence functions later

Summary

Divide-and-Conquer

- Divide: Divide problem in to subproblems.
  - Subproblem is at most a constant fraction of the original problem.
- Conquer: Recursively solve each subproblem.
- Combine: Merge solutions of subproblems to the solution of the original problem

Mergesort

- Divide array into two halves.
- Merge two halves to make sorted whole.

Counting inversions

Counting Inversions

Music site tries to match your song preferences with others.

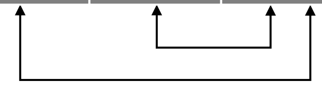
- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: 1, 2, …, n.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$.

Songs

Me	1	2	3	4	5
You	1	3	4	2	5



Inversions

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j.

Applications

Applications

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Genomic distance between two gene sequences.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: $O(1)$.

1	5	4	8	10	2

6	9	12	11	3	7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

6	9	12	11	3	7

Conquer: 2T(n / 2)

5 blue-blue inversions

5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- Combine: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7

Divide: O(1).

1	5	4	8	10	2

5 blue-blue inversions

6	9	12	11	3	7

8 green-green inversions

Conquer: 2T(n / 2)

9 blue-green inversions
5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Combine: ???

Total = 5 + 8 + 9 = 22.

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

2	3	7	10	11	14	16	17	18	19	23	25

Count: $O(n)$

Combine:

- Sort two halves.
- Count inversions where a_i and a_j are in different halves.

$O(n(\log n)^2)$ time

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where a_i and a_j are in different halves.
- Merge two sorted halves into sorted whole.

to maintain sorted invariant

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: 6 + 3 + 2 + 2 + 0 + 0

Count: O(n)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

$$T(n) \le T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + O(n) \quad \Rightarrow \quad \mathrm{T}(n) = O(n \log n)$$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r, L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

Lesson

Sometimes, it is useful to redefine the problem to make the recursion work

In the counting inversions problem

- The merge step becomes easier if two halves are sorted
- So, we redefine the problem (as well as the subproblems) as finding the number of inversions and sorting the input