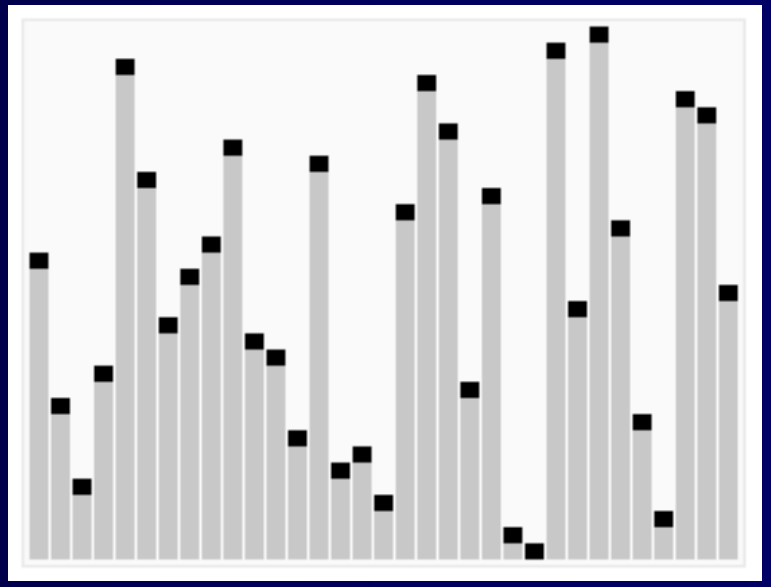
Simple Sorting Algorithms

Review of Quick Sort

Pick a pivot, arrange other elements based on if they're greater or less than the pivot

Repeat for each smaller group until the whole thing is sorted

Usually O(nlogn), absolute worst case is $O(n^2)$



Bubble Sort

Compare each element (except the last one) with its neighbor to the right

- If they are out of order, swap them
- This puts the largest element at the very end
- The last element is now in the correct and final place

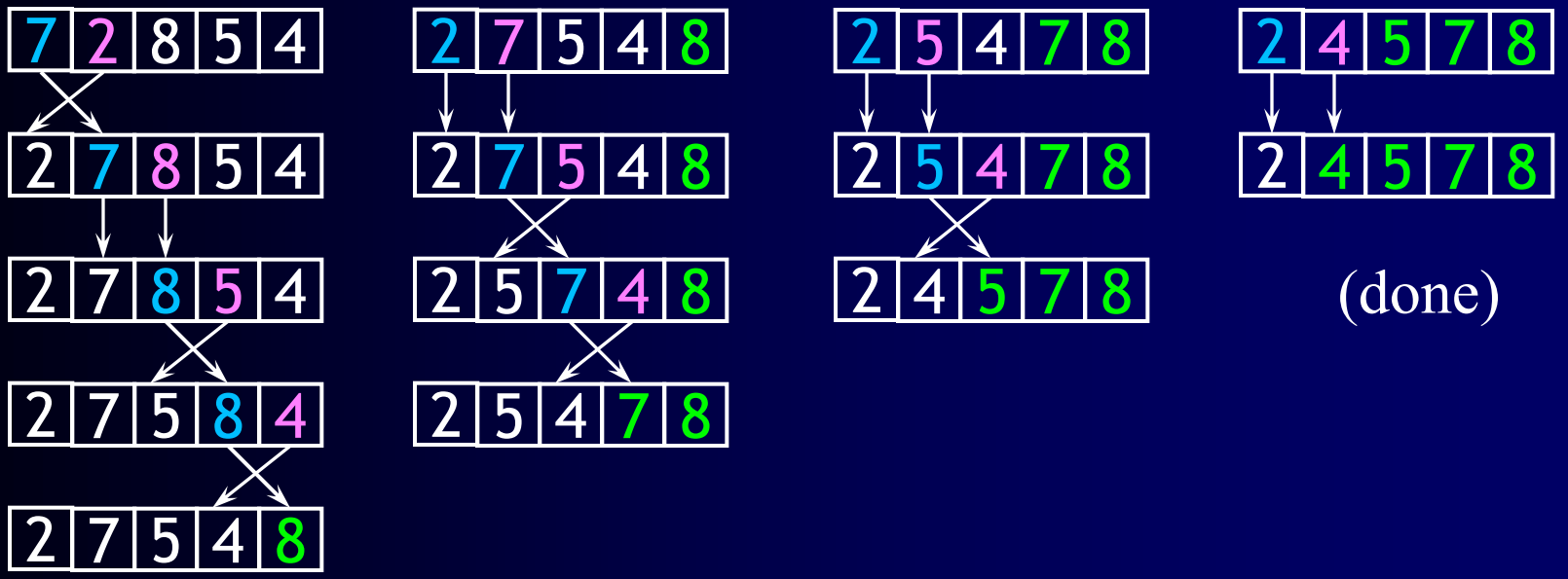
Compare each element (except the last *two*) with its neighbor to the right

- If they are out of order, swap them
- This puts the second largest element next to last
- The last *two* elements are now in their correct and final places

Compare each element (except the last *three*) with its neighbor to the right

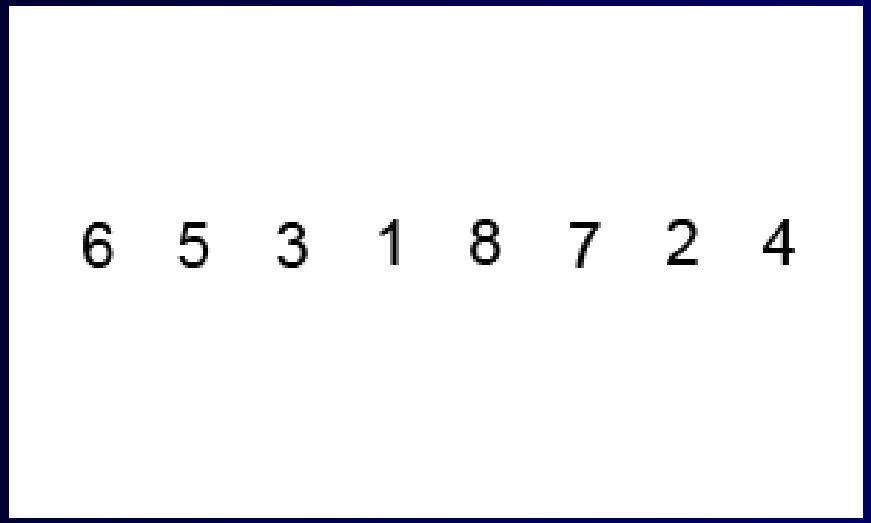
- Continue as above until you have no unsorted elements on the left

Example of Bubble Sort



Can you guess its Big-O notation?

$O(n^2)$

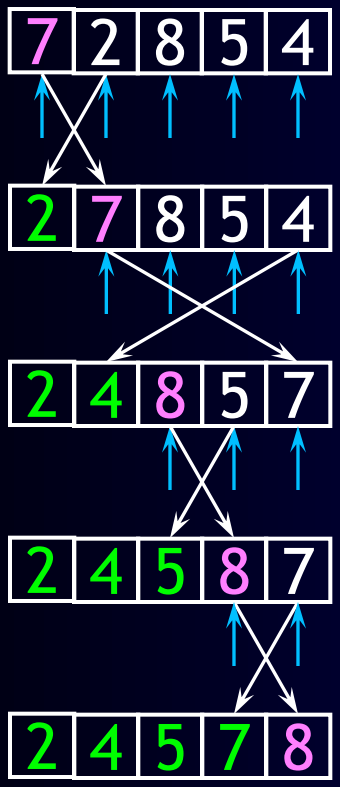


Selection sort

Given a list of length n,

- Search elements 0 through n-1 and select the smallest
  - Swap it with the element in location 0
- Search elements 1 through n-1 and select the smallest
  - Swap it with the element in location 1
- Search elements 2 through n-1 and select the smallest
  - Swap it with the element in location 2
- Search elements 3 through n-1 and select the smallest
  - Swap it with the element in location 3
- Continue in this fashion until there's nothing left to search

Analysis of Selection Sort



Analysis:

The outer loop executes $n-1$ times

The inner loop executes about $n/2$ times on average (from n to 2 times)

Work done in the inner loop is constant (swap two array elements)

Time required is roughly $(n-1)*(n/2)$
You should recognize this as $O(n^2)$

Invariants for Selection Sort

For the inner loop:

- This loop searches through the array, incrementing inner from its initial value of outer+1 up to a.length-1
- As the loop proceeds, min is set to the index of the smallest number found so far
- Our invariant is:
  for all i such that outer <= i <= inner, a[min] <= a[i]

For the outer (enclosing) loop:

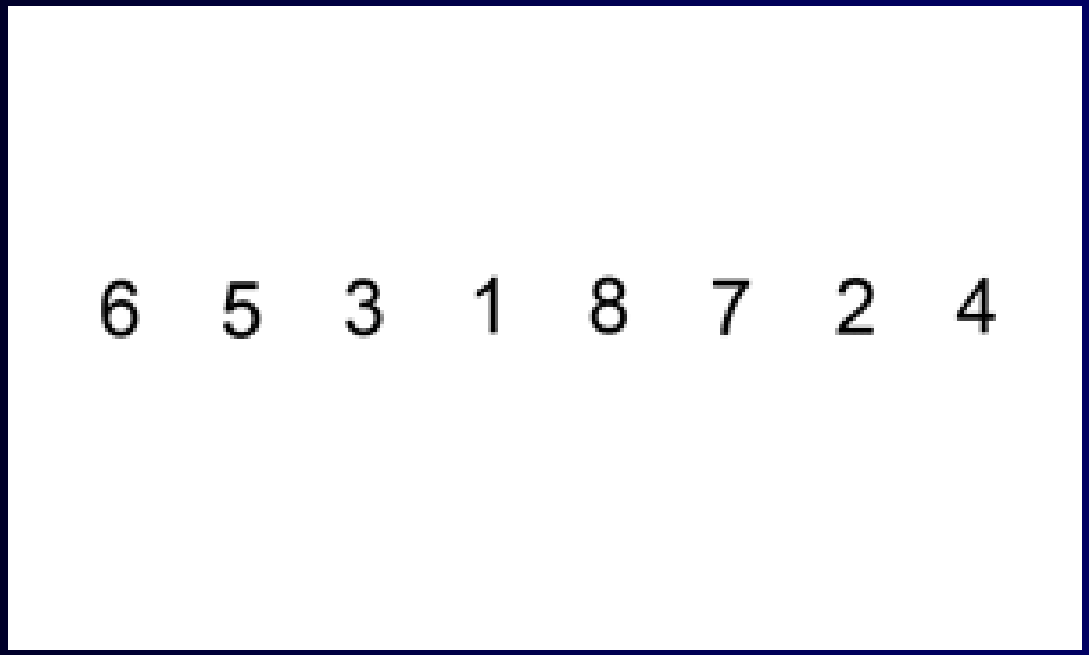
- The loop counts up from outer = 0
- Each time through the loop, the minimum remaining value is put in a[outer]
- Our invariant is:
  for all i <= outer, if i < j then a[i] <= a[j]

Insertion sort

From left to right, go through each element in the list.

If it is smaller than the element to its left, check all the elements you've already done to see where it belongs.

When you find the right place, insert it between the element that is bigger than it and the element that is smaller.


6 5 3 1 8 7 2 4

Analysis of insertion sort

We have to check each of the n elements

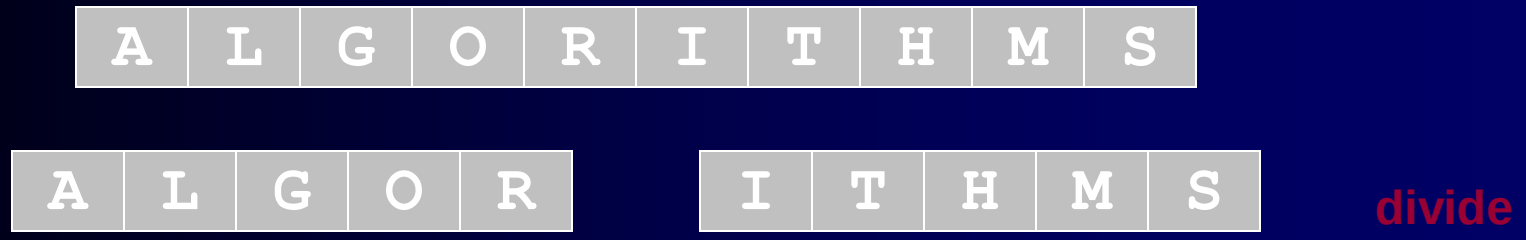
On average, there are $n/2$ elements already sorted
- The inner loop looks at (and moves) half of these
- This gives a second factor of $n/4$

Hence, the time required for an insertion sort of an array of n elements is proportional to $n^2/4$

Discarding constants, we find that insertion sort is $O(n^2)$

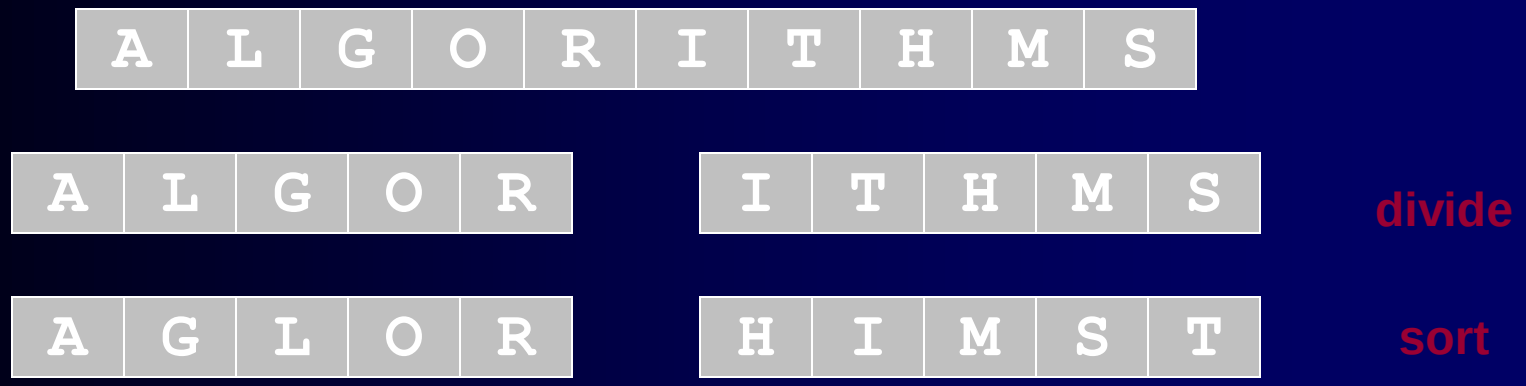
Mergesort

Divide array into two halves.



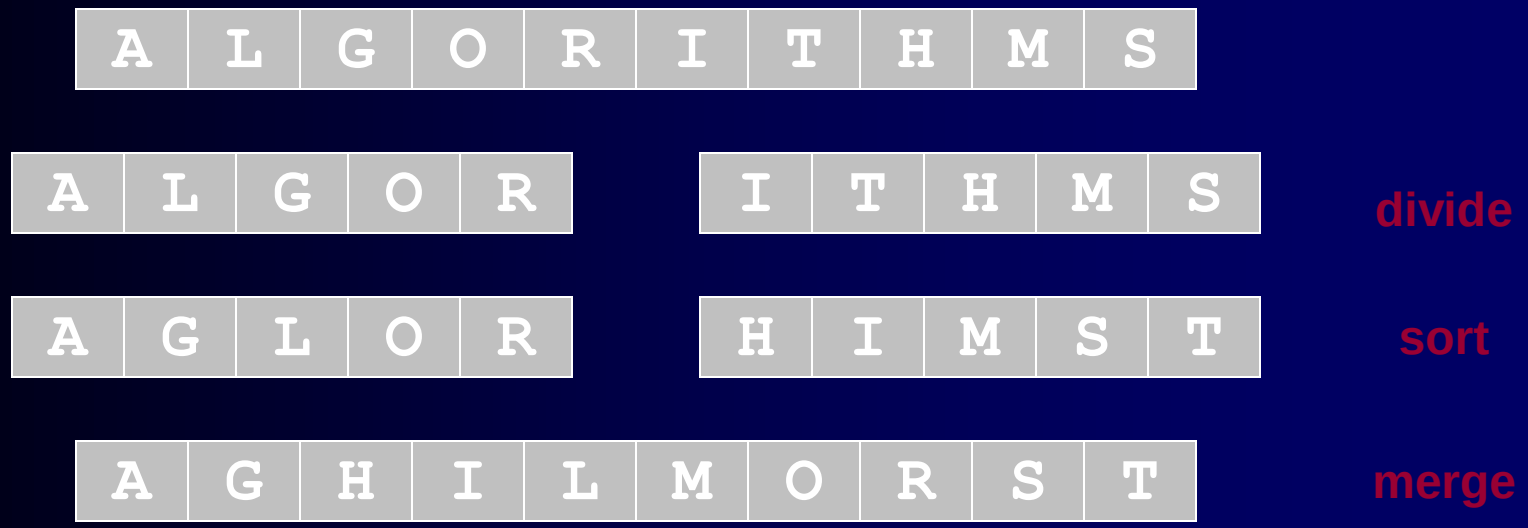
Mergesort

Divide array into two halves.
Recursively sort each half.



Mergesort

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

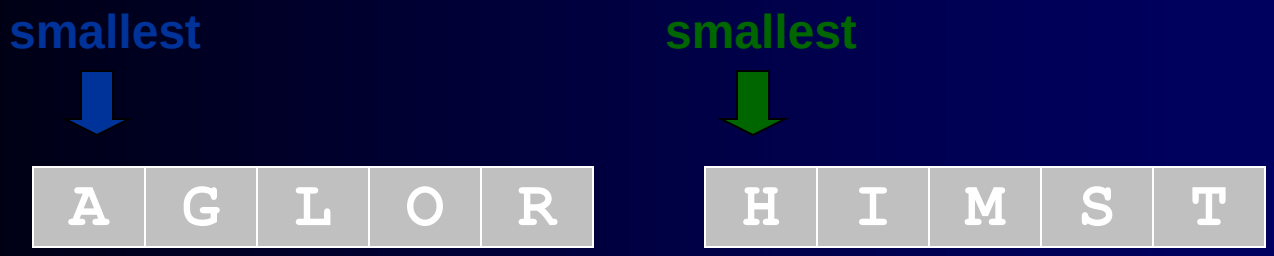


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.



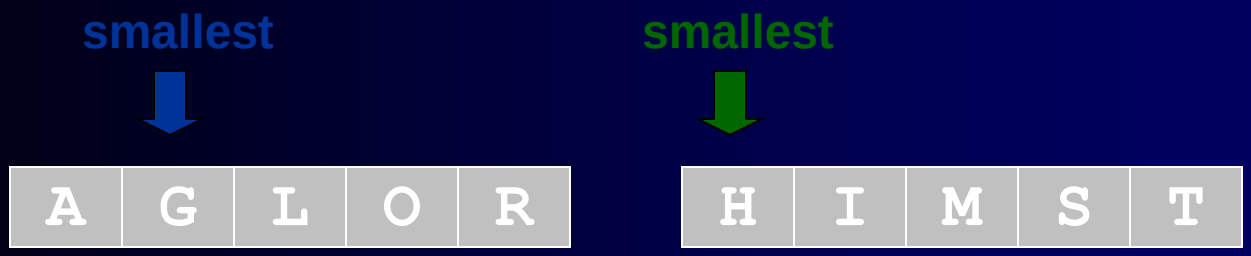
auxiliary array

Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.



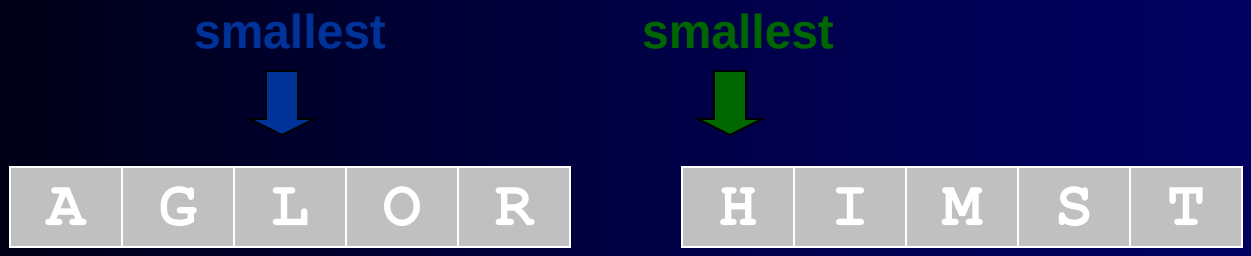
auxiliary array

Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

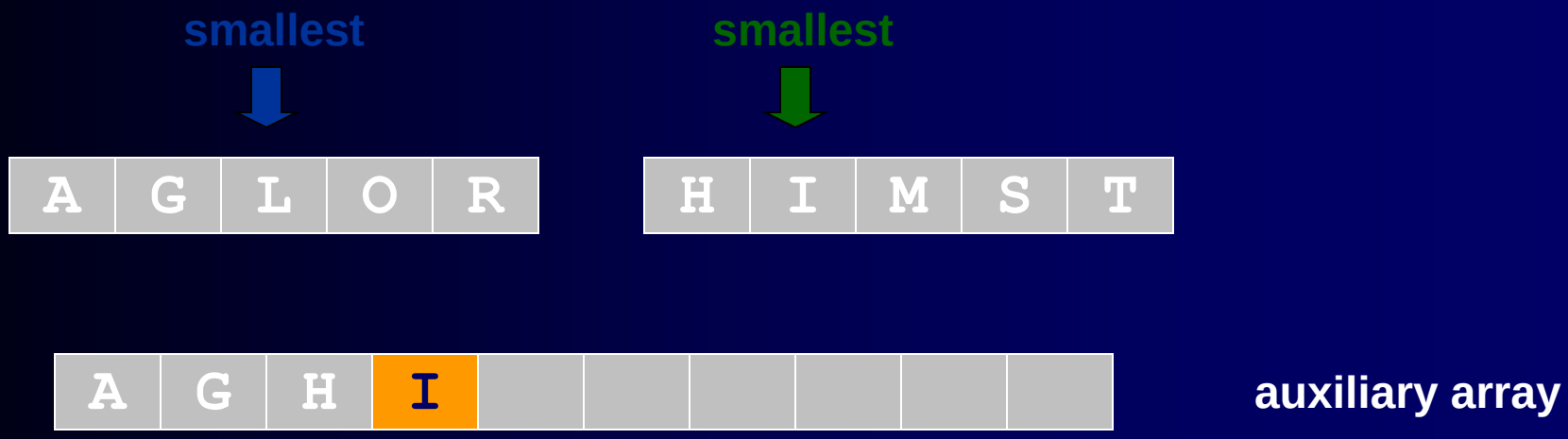


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

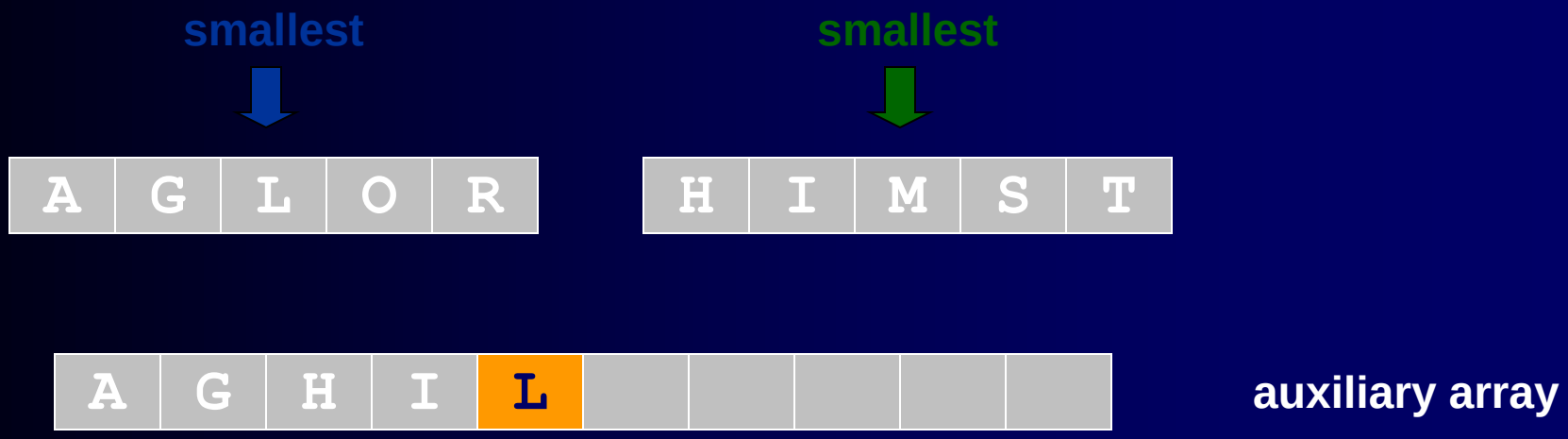


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

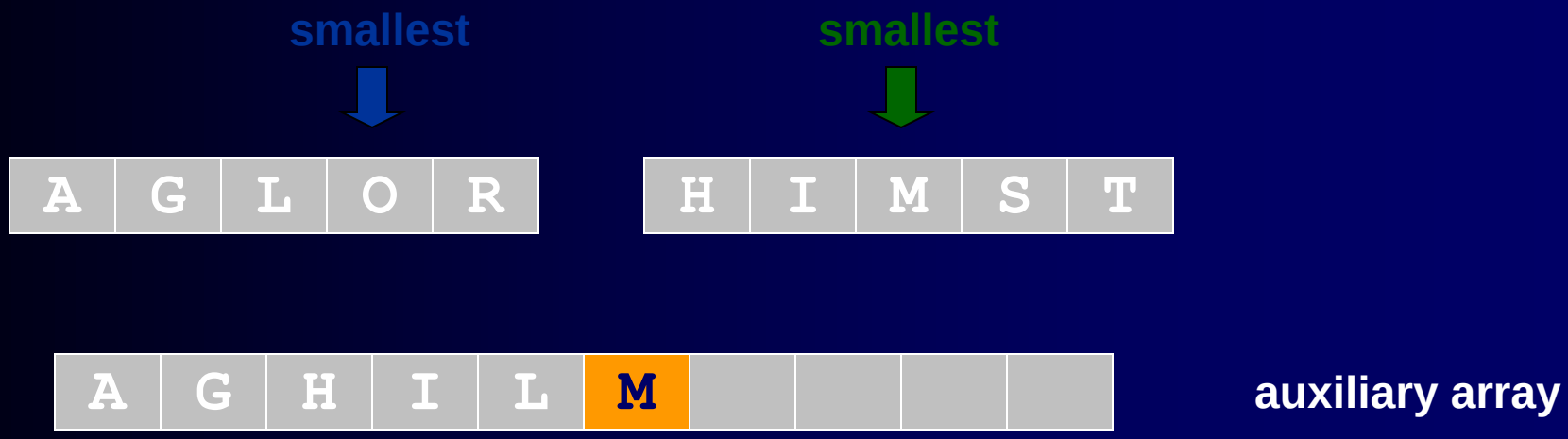


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

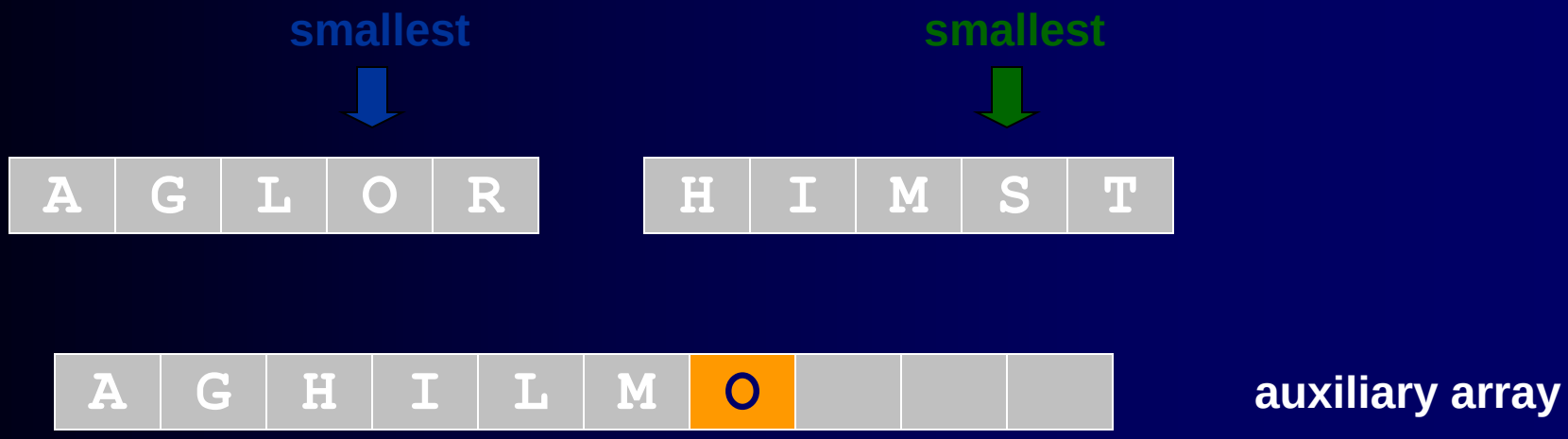


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

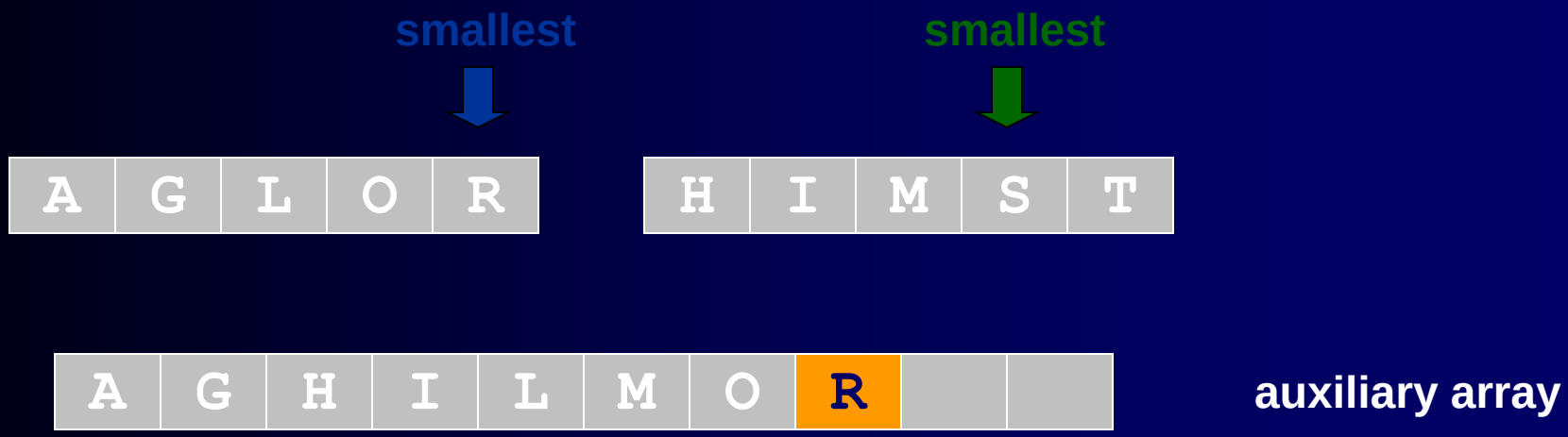


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.

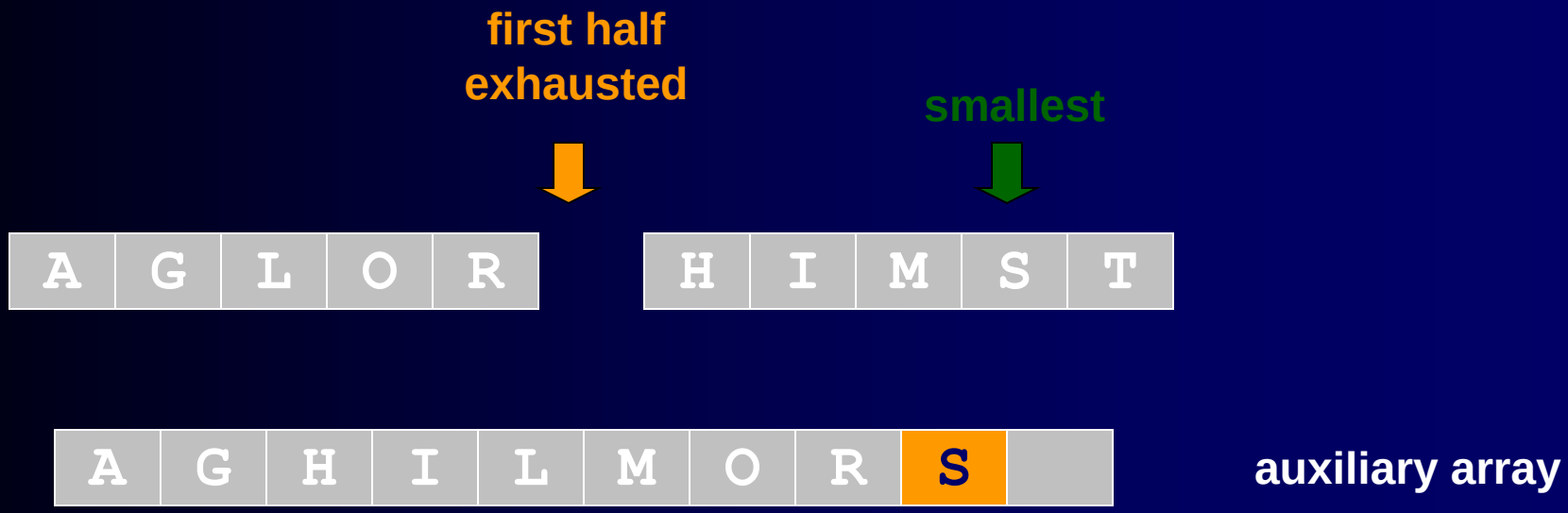


Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.



Merging

Keep track of smallest element in each sorted half.

Insert smallest of two elements into auxiliary array.

Repeat until done.