# Winter 2016
# COMP-250: Introduction to Computer Science

Lecture 10, February 11, 2016

# A Survey of Common Running Times

# Linear Time: O(n)

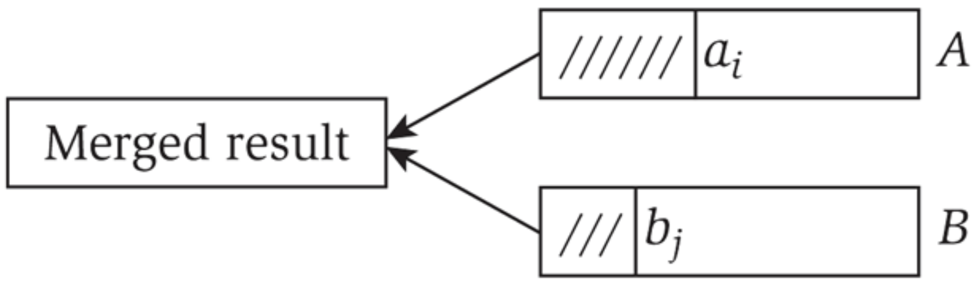
**Linear time.** Running time is proportional to input size.

**Computing the maximum.** Compute maximum of n numbers $a_1, \ldots, a_n$.

```
max ← a1
for i = 2 to n {
   if (ai > max)
      max ← ai
}
```

# Linear Time: O(n)

**Merge.** Combine two sorted lists $A = a_1, a_2, \ldots, a_n$ with $B = b_1, b_2, \ldots, b_n$ into a sorted whole.




```
i = 1, j = 1
while (both lists are nonempty) {
   if (ai ≤ bj) append ai to output list and increment i
   else         append bj to output list and increment j
}
append remainder of nonempty list to output list
```

**Claim.** Merging two lists of size n takes O(n) time.

**Pf.** After each comparison, the length of output list increases by 1.

# O(n log n) Time

also referred to as linearithmic time

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

**Sorting.** **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

**Sorting.** **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

**Sorting.** **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

**Largest empty interval.** Given n time-stamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

**Sorting.** **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

**Largest empty interval.** Given n time-stamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

# O(n log n) Time

**O(n log n) time.** Arises in divide-and-conquer algorithms.

↖ also referred to as linearithmic time

**Sorting.** **Mergesort** and **Heapsort** are sorting algorithms that perform O(n log n) comparisons.

**Largest empty interval.** Given n time-stamps $x_1, \ldots, x_n$ on which copies of a file arrive at a server, what is largest interval of time when no copies of the file arrive?

**O(n log n) solution.** Sort the time-stamps. Scan the sorted list in order, identifying the maximum gap between successive time-stamps.

# Quadratic Time: $O(n^2)$

**Quadratic time.** Enumerate all pairs of elements.

**Closest pair of points.** Given a list of n points in the plane $(x_1, y_1), \ldots, (x_n, y_n)$, find the pair that is closest.

**$O(n^2)$ solution.** Try all pairs of points.

```
min ← (x1 - x2)² + (y1 - y2)²
for i = 1 to n {
   for j = i+1 to n {
      d ← (xi - xj)² + (yi - yj)²
      if (d < min)
         min ← d
   }
}
```

don't need to take square roots

**Remark.** This algorithm is $\Omega(n^2)$ and it seems inevitable in general, but this is just an illusion.

# Cubic Time: $O(n^3)$

**Cubic time.** Enumerate all triples of elements.

**Set disjointness.** Given n sets $S_1, \ldots, S_n$ each of which is a subset of 1, 2, …, n, is there some pair of these which are disjoint?

**$O(n^3)$ solution.** For each pair of sets, determine if they are disjoint.

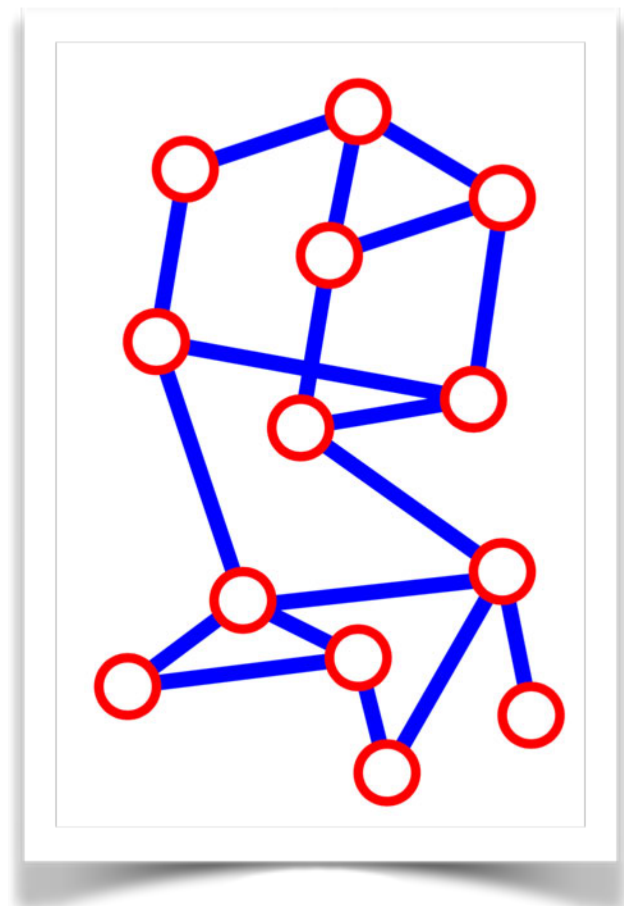
```
foreach set Si {
   foreach other set Sj {
      foreach element p of Si {
         determine whether p also belongs to Sj
      }
      if (no element of Si belongs to Sj)
         report that Si and Sj are disjoint
   }
}
```

# Polynomial Time: $O(n^k)$

**Independent set of size k.** Given a graph, are there k nodes such that no two are joined by an edge?

k is a constant

**$O(n^k)$ solution.** Enumerate all subsets of k nodes.



```
foreach subset S of k nodes {
   check whether S in an independent set
   if (S is an independent set)
      report S is an independent set
   }
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets : $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \leq \frac{n^k}{k!}$
- $O(k^2\, n^k / k!)$ is $O(n^k)$.

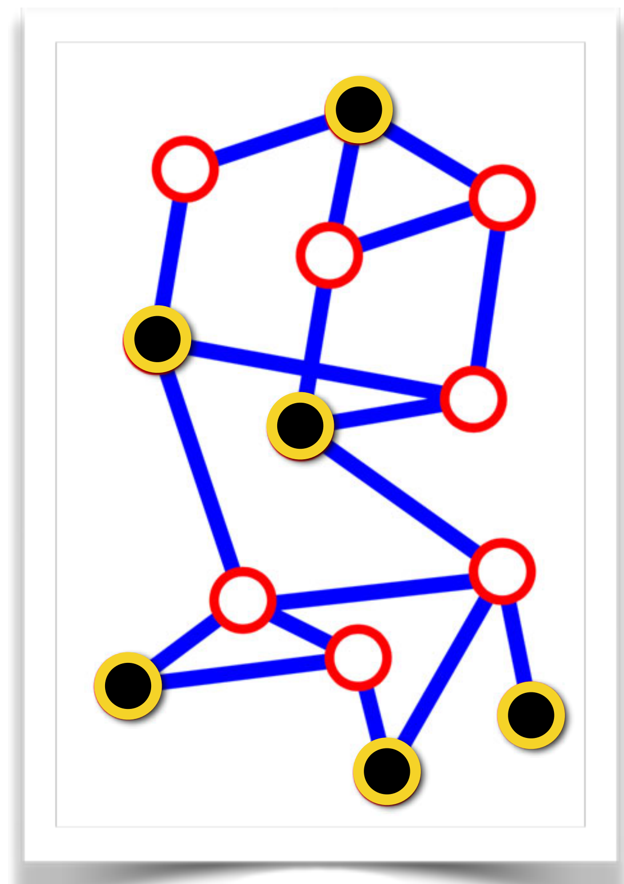
poly-time for k=17,
but not practical

# Polynomial Time: $O(n^k)$

**Independent set of size k.** Given a graph, are there k nodes such that no two are joined by an edge?

k is a constant

**$O(n^k)$ solution.** Enumerate all subsets of k nodes.



```
foreach subset S of k nodes {
   check whether S in an independent set
   if (S is an independent set)
      report S is an independent set
   }
}
```

- Check whether S is an independent set = $O(k^2)$.
- Number of k element subsets : $\binom{n}{k} = \frac{n\,(n-1)\,(n-2)\cdots(n-k+1)}{k\,(k-1)\,(k-2)\cdots(2)\,(1)} \le \frac{n^k}{k!}$
- $O(k^2\ n^k\ /\ k!)$ is $O(n^k)$.

poly-time for k=17,
but not practical

# Exponential Time: $O(c^n)$

**Independent set.** Given a graph, what is the maximum size of an independent set?

**$O(n^2 2^n)$ solution.** Enumerate all subsets.

```
S* ← ∅
foreach subset S of nodes {
   check whether S in an independent set
   if (S is largest independent set seen so far)
      update S* ← S
   }
}
```

# Induction and Recursion

# Induction Proofs

# Induction Proofs

Predicate.

# Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

$\forall n \geq 1, P(n)$ is true.

Proof.

# Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

$\forall n \geq 1$, P(n) is true.

Proof.

- Base case: proof that P(1) is true.

# Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

$\forall n \geq 1$, P(n) is true.

Proof.

- Base case: proof that P(1) is true.

# Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.
$\forall n \geq 1$, P(n) is true.

Proof.

- Base case: proof that P(1) is true.

- Induction step: $\forall n \geq 1$, $P(n) \implies P(n+1)$.

Let $n \geq 1$.
Assume for induction hypothesis that P(n) is true and prove P(n+1) is also true.

# Induction Proof (1)

# Induction Proof (1)

Predicate.

# Induction Proof (1)

Predicate.

- P(n) : $1+2+\ldots+n = n(n+1)/2$

# Induction Proof (1)

Predicate.

- P(n) : 1+2+…+n = n(n+1)/2

# Induction Proof (1)

Predicate.

- P(n) : $1+2+\ldots+n = n(n+1)/2$

- Base case: when n=1 we have

  $1+\ldots+n = 1 = 1(2)/2 = n(n+1)/2.$

  P(1) is true.

# Induction Proof (1)

Predicate.

- P(n) : $1+2+\ldots+n = n(n+1)/2$

- Base case: when n=1 we have

  $1+\ldots+n = 1 = 1(2)/2 = n(n+1)/2.$

  P(1) is true.

# Induction Proof (1)

Predicate.

- $P(n)$ : $1+2+\ldots+n = n(n+1)/2$

- Base case: when $n=1$ we have

  $$1+\ldots+n = 1 = 1(2)/2 = n(n+1)/2.$$

  $P(1)$ is true.

- Induction step: let $n \geq 1$. Assume for induction hypothesis that $P(n)$ is true.
  We show $P(n+1)$ is true as well :

  $$\begin{aligned} 1+2+\ldots+n+(n+1) &= n(n+1)/2 + (n+1) \text{ by I.H.} \\ &= (n+1)(n/2 + 1) \\ &= (n+1)(n+2)/2. \end{aligned}$$

  $n \geq 1, P(n) \Longrightarrow P(n+1)$.

# Induction Proof (II)

# Induction Proof (II)

Predicate.

# Induction Proof (II)

Predicate.

- $P(n) : \sum_{i=1}^{n} i = n(n+1)/2$

# Induction Proof (II)

Predicate.

- P(n) : $\sum_{i=1}^{n} i = n(n+1)/2$

# Induction Proof (II)

Predicate.

- P(n) : $\sum_{i=1}^{n} i = n(n+1)/2$
- Base case: when n=1, $\sum_{i=1}^{1} i = 1 = 1(2)/2 = n(n+1)/2$.

  P(1) is true.

# Induction Proof (II)

Predicate.

- P(n) : $\sum_{i=1}^{n} i = n(n+1)/2$
- Base case: when n=1, $\sum_{i=1}^{1} i = 1 = 1(2)/2 = n(n+1)/2$.

  P(1) is true.

# Induction Proof (II)

Predicate.

- $P(n) : \sum_{i=1}^{n} i = n(n+1)/2$
- Base case: when n=1, $\sum_{i=1}^{1} i = 1 = 1(2)/2 = n(n+1)/2$.

  P(1) is true.

- Induction step: let $n \geq 1$. Assume for induction hypothesis that P(n) is true.
  We show P(n+1) is true as well :

  $$\sum_{i=1}^{n+1} i = (n+1) + \sum_{i=1}^{n} i$$

  $$= (n+1) + n(n+1)/2 \quad \text{by I.H.}$$
  $$= (n+1)(1+n/2)$$
  $$= (n+1)(n+2)/2.$$

  $n \geq 1, P(n) \Longrightarrow P(n+1)$.

# Iteration vs Recursion

```
f(n)
sum ← 0
for i = 2 to n {
   sum ← sum + i
}
return sum
```

{

```
f(n)
if n = 0 { return 0 }
else { return f(n-1)+n }
```

# Iteration vs Recursion

```
f(n)
sum ← 0
for i = 2 to n {
   sum ← sum + i
}
return sum
```

{

```
f(n)
if n = 0 { return 0 }
else { return f(n-1)+n }
```

# Iteration vs Recursion

- $f(n) = 1 + 2 + \ldots + n = \sum_{i=1}^{n} i$

```
f(n)
sum ← 0
for i = 2 to n {
    sum ← sum + i
}
return sum
```

$$\begin{cases} 0 & \text{if } n = 0 \end{cases}$$

```
f(n)
if n = 0 { return 0 }
else { return f(n-1)+n }
```

# Iteration vs Recursion

- $f(n) = 1 + 2 + \ldots + n = \sum_{i=1}^{n} i$

```
f(n)
sum ← 0
for i = 2 to n {
   sum ← sum + i
}
return sum
```

- $f(n) = \begin{cases} 0 & \text{if } n = 0 \\ f(n-1)+n & \text{if } n > 0 \end{cases}$

```
f(n)
if n = 0 { return 0 }
else { return f(n-1)+n }
```

# Induction Proof (III)

# Induction Proof (III)

Predicate.

# Induction Proof (III)

Predicate.

- P(n) : $f(n) = n(n+1)/2$

# Induction Proof (III)

Predicate.

- $P(n)$ : $f(n) = n(n+1)/2$

# Induction Proof (III)

Predicate.

- P(n) :  f(n) = n(n+1)/2

- Base case: when n=1, f(1) = 1 = 1(2)/2 = n(n+1)/2.

  P(1) is true.

# Induction Proof (III)

Predicate.

- $P(n)$ : $f(n) = n(n+1)/2$
- Base case: when $n=1$, $f(1) = 1 = 1(2)/2 = n(n+1)/2$.

  $P(1)$ is true.

# Induction Proof (III)

Predicate.

- $P(n)$ :  $f(n) = n(n+1)/2$

- Base case: when $n=1$, $f(1) = 1 = 1(2)/2 = n(n+1)/2$.

  $P(1)$ is true.

- Induction step: let $n \geq 1$.  Assume for induction hypothesis that $P(n)$ is true.
  We show $P(n+1)$ is true as well :

  $$\begin{aligned} f(n+1) &= f(n) + (n+1) && \text{by definition} \\ &= n(n+1)/2 + (n+1) && \text{by I.H.} \\ &= (n+1)(n/2+1) \\ &= (n+1)(n+2)/2. \end{aligned}$$

  $n \geq 1, P(n) \Longrightarrow P(n+1)$.

# Generalized Induction Proofs

# Generalized Induction Proofs

Predicate.

# Generalized Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

For all $n \geq 1$, P(n) is true.

Proof.

# Generalized Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

For all $n \geq 1$, P(n) is true.

Proof.

- Base case: proof that P(1) is true.

# Generalized Induction Proofs

Predicate.

- $P(n)$ : $f(n)$ = some formula in n

Statement.

For all $n \geq 1$, $P(n)$ is true.

Proof.

- Base case: proof that $P(1)$ is true.

# Generalized Induction Proofs

Predicate.

- P(n) : f(n) = some formula in n

Statement.

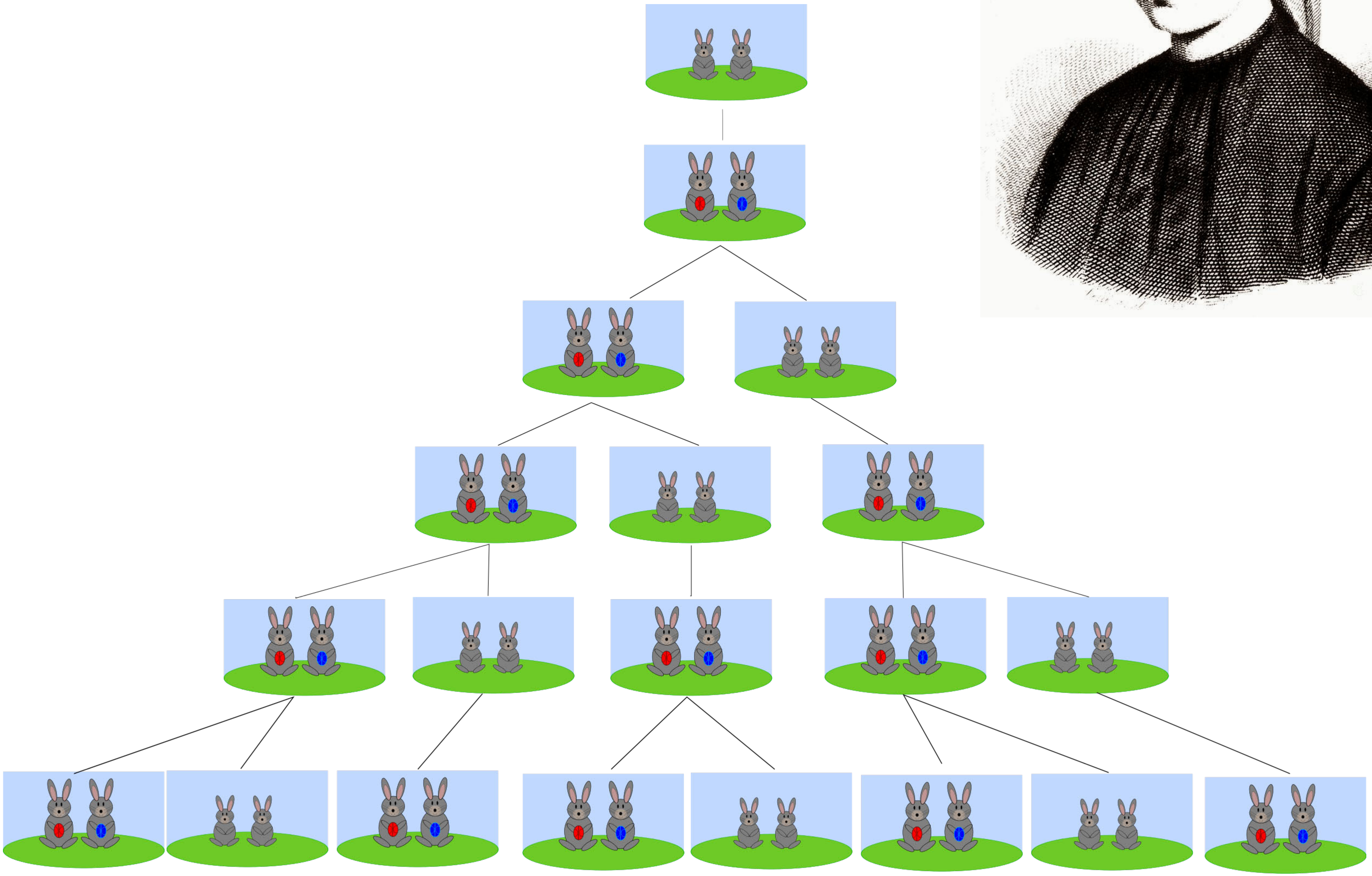
For all $n \geq 1$, P(n) is true.

Proof.

- Base case: proof that P(1) is true.

- Induction step: let $n \geq 1$. Assume for induction hypothesis that P(1)…P(n) are all true. We show P(n+1) is also true.

# Recursion



# Recursion:
## Fibonacci Sequence

{

# Recursion:
## Fibonacci Sequence

$$\begin{cases} n & \text{if } n \leq 1 \\ \end{cases}$$

# Recursion:
## Fibonacci Sequence

- $$\text{fib}(n) = \begin{cases} n & \text{if } n \leq 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

Fibonacci sequence:
0,1,1,2,3,5,8,13,21,34,55,89,144,…

# Recursion:
## Fibonacci Sequence

- $$\text{fib}(n) = \begin{cases} n & \text{if } n \leq 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

  Fibonacci sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

# Recursion:
## Fibonacci Sequence

- $$\mathrm{fib}(n) = \begin{cases} n & \text{if } n \leq 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2) & \text{if } n > 1 \end{cases}$$

Fibonacci sequence:
0,1,1,2,3,5,8,13,21,34,55,89,144,…

# Recursion:
## Fibonacci Sequence

- $$\text{fib}(n) = \begin{cases} n & \text{if } n \leq 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$$

  Fibonacci sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

- NOT so easy to define iteratively…

# Recursion vs Iteration

{

```
fib(n)
if n < 2 { return n }
else { return fib(n-1)+ fib(n-2) }
```

```
fib(n)
a ← 0
b ← 1
for i = 1 to n {
   b ← a + b
   a ← b - a
}
return a
```

# Recursion vs Iteration

$$\begin{cases} n & \text{if } n \leq 1 \\ & \end{cases}$$

```
fib(n)
if n < 2 { return n }
else { return fib(n-1)+ fib(n-2) }
```

```
fib(n)
a ← 0
b ← 1
for i = 1 to n {
   b ← a + b
   a ← b - a
}
return a
```

# Recursion vs Iteration

- $\text{fib}(n) = \begin{cases} n & \text{if } n \leq 1 \\ \text{fib}(n-1) + \text{fib}(n-2) & \text{if } n > 1 \end{cases}$

```
fib(n)
if n < 2 { return n }
else { return fib(n-1)+ fib(n-2) }
```

```
fib(n)
a ← 0
b ← 1
for i = 1 to n {
   b ← a + b
   a ← b - a
}
return a
```

# Generalized Induction Proofs

# Generalized Induction Proofs

**Statement.**
For all $n \geq 0$, $P(n)$ : $fib(n) \leq 2^n$ is true.

**Proof.**

# Generalized Induction Proofs

**Statement.**
For all $n \geq 0$, $P(n)$ : $fib(n) \leq 2^n$ is true.

**Proof.**

- Base case: $P(0)$: $fib(0) = 0 \leq 2^0$ is true.
  $P(1)$: $fib(1) = 1 \leq 2^1$ is true.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 0$, $P(n)$ : $fib(n) \leq 2^n$ is true.

**Proof.**

- Base case: $P(0)$: $fib(0) = 0 \leq 2^0$ is true.
  $P(1)$: $fib(1) = 1 \leq 2^1$ is true.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 0$, $P(n)$ : $fib(n) \leq 2^n$ is true.

**Proof.**

- Base case: $P(0)$: $fib(0) = 0 \leq 2^0$ is true.
  $P(1)$: $fib(1) = 1 \leq 2^1$ is true.

- Induction step: let $n \geq 1$. Assume for induction hypothesis that $P(0) \ldots P(n)$ are all true. We show $P(n+1)$ is also true:

  $fib(n+1) = fib(n) + fib(n-1)$ by definition
  $\leq 2^n + 2^{n-1}$ by gen. I. H.
  $\leq 2^{n-1} \cdot 3 < 2^{n+1}$

# Generalized Induction Proofs

# Generalized Induction Proofs

Statement.

For all $n \geq 1$, $P(n)$ : $fib(n) \leq \varphi^n$ is true.

Proof.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 1$, $P(n)$ : $fib(n) \leq \varphi^n$ is true.

**Proof.**

- Base case: $P(1)$: $fib(1) = 1 \leq \varphi^1$ is true (if $\varphi \geq 1$).
  $P(2)$: $fib(2) = 1 \leq \varphi^2$ is true (if $\varphi \geq 1$).

# Generalized Induction Proofs

**Statement.**
For all $n \geq 1$, $P(n)$ : $fib(n) \leq \varphi^n$ is true.

**Proof.**
- Base case: $P(1)$: $fib(1) = 1 \leq \varphi^1$ is true (if $\varphi \geq 1$).
  $P(2)$: $fib(2) = 1 \leq \varphi^2$ is true (if $\varphi \geq 1$).

# Generalized Induction Proofs

Statement.

For all $n \geq 1$, $P(n)$ : $fib(n) \leq \varphi^n$ is true.

Proof.

- Base case: $P(1)$: $fib(1) = 1 \leq \varphi^1$ is true (if $\varphi \geq 1$).
  $P(2)$: $fib(2) = 1 \leq \varphi^2$ is true (if $\varphi \geq 1$).

- Induction step: let $n \geq 1$. Assume for induction hypothesis that $P(1) \ldots P(n)$ are all true. We show $P(n+1)$ is also true:

  $fib(n+1) = fib(n) + fib(n-1)$ by definition
  $\leq \varphi^n + \varphi^{n-1}$ by gen. I. H.
  $\leq \varphi^{n-1} (\varphi+1) \leq \varphi^{n+1}$
  whenever $(\varphi+1) \leq \varphi^2$
  whenever $0 \leq \varphi^2 - \varphi - 1$.

# Generalized Induction Proofs

# Generalized Induction Proofs

Statement.
For all $n \geq 1$, $P(n)$ : $fib(n) \geq \varphi^{n-2}$ is true.

Proof.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 1$, $P(n)$ : $fib(n) \geq \varphi^{n-2}$ is true.

**Proof.**

- Base case: $P(1)$: $fib(1) = 1 \geq \varphi^{-1}$ is true (if $\varphi \geq 1$).
  $P(2)$: $fib(2) = 1 = \varphi^{0}$ is true.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 1$, $P(n)$ : fib(n) $\geq \varphi^{n-2}$ is true.

**Proof.**

- Base case: $P(1)$: fib(1) = 1 $\geq \varphi^{-1}$ is true (if $\varphi \geq 1$).
  $P(2)$: fib(2) = 1 = $\varphi^{0}$ is true.

# Generalized Induction Proofs

**Statement.**
For all $n \geq 1$, $P(n)$ : $\text{fib}(n) \geq \varphi^{n-2}$ is true.

**Proof.**

- Base case: $P(1)$: $\text{fib}(1) = 1 \geq \varphi^{-1}$ is true (if $\varphi \geq 1$).
  $P(2)$: $\text{fib}(2) = 1 = \varphi^{0}$ is true.

- Induction step: let $n \geq 1$. Assume for induction hypothesis that $P(1) \ldots P(n)$ are all true. We show $P(n+1)$ is also true:

  $\text{fib}(n+1) = \text{fib}(n) + \text{fib}(n-1)$ by definition
  $\geq \varphi^{n-2} + \varphi^{n-3}$ by gen. I. H.
  $\geq \varphi^{n-3}(\varphi+1) \geq \varphi^{n-1}$
  whenever $(\varphi+1) \geq \varphi^{2}$
  whenever $0 \geq \varphi^{2} - \varphi - 1$.

# Weak Binet Formula

# Weak Binet Formula

Statements.

For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.

whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

# Weak Binet Formula

Statements.
For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

# Weak Binet Formula

Statements.

For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.

whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.

whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

# Weak Binet Formula

Statements.

For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.

whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.

whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

# Weak Binet Formula

Statements.
For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.
whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

Therefore:

# Weak Binet Formula

Statements.
For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.
whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

Therefore:
For all $n \geq 1$, $\varphi^n / \varphi^2 \leq fib(n) \leq \varphi^n$ is true.
whenever $0 = \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

# Weak Binet Formula

Statements.
For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.
whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

Therefore:
For all $n \geq 1$, $\varphi^n/\varphi^2 \leq fib(n) \leq \varphi^n$ is true.
whenever $0 = \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.
Only solution $\varphi$ = golden ration = $(1+\sqrt{5})/2$.

# Weak Binet Formula

Statements.
For all $n \geq 1$, $\text{fib}(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $\text{fib}(n) \geq \varphi^{n-2}$ is true.
whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

Therefore:
For all $n \geq 1$, $\varphi^n/\varphi^2 \leq \text{fib}(n) \leq \varphi^n$ is true.
whenever $0 = \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.
Only solution $\varphi$ = golden ration = $(1+\sqrt{5})/2$.

# Weak Binet Formula

Statements.
For all $n \geq 1$, $fib(n) \leq \varphi^n$ is true.
whenever $0 \leq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

For all $n \geq 1$, $fib(n) \geq \varphi^{n-2}$ is true.
whenever $0 \geq \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.

Therefore:
For all $n \geq 1$, $\varphi^n / \varphi^2 \leq fib(n) \leq \varphi^n$ is true.
whenever $0 = \varphi^2 - \varphi - 1$ and $\varphi \geq 1$.
Only solution $\varphi$ = golden ration = $(1+\sqrt{5})/2$.

fib(n) is $\boldsymbol{\theta}(\varphi^n)$.

# Generalized Induction Proofs

{

# Generalized Induction Proofs

$$\begin{cases} n & \text{if } n \leq 1 \\ & \end{cases}$$

# Generalized Induction Proofs

- $f(n)=\begin{cases} n & \text{if } n \leq 1 \\ f^2(^{n+1}/_2) + f^2(^{n-1}/_2) & \text{if odd } n>1 \\ f^2(^n/_2+1) - f^2(^n/_2-1) & \text{if even } n>1 \end{cases}$

  f-sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

# Generalized Induction Proofs

- $$f(n)=\begin{cases} n & \text{if } n \leq 1 \\ f^2(^{n+1}/_2) + f^2(^{n-1}/_2) & \text{if odd } n>1 \\ f^2(^{n}/_2+1) - f^2(^{n}/_2-1) & \text{if even } n>1 \end{cases}$$

  f-sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

# Generalized Induction Proofs

- $$f(n)=\begin{cases} n & \text{if } n \leq 1 \\ f^2(\tfrac{n+1}{2}) + f^2(\tfrac{n-1}{2}) & \text{if odd } n>1 \\ f^2(\tfrac{n}{2}+1) - f^2(\tfrac{n}{2}-1) & \text{if even } n>1 \end{cases}$$

  f-sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

Statement.
For all $n \geq 0$, fib(n) = f(n).

# Generalized Induction Proofs

- $f(n) = \begin{cases} n & \text{if } n \leq 1 \\ f^2(^{n+1}/_2) + f^2(^{n-1}/_2) & \text{if odd } n>1 \\ f^2(^{n}/_2+1) - f^2(^{n}/_2-1) & \text{if even } n>1 \end{cases}$

  f-sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

**Statement.**
For all $n \geq 0$, fib(n) = f(n).

# Generalized Induction Proofs

- $f(n)=\begin{cases} n & \text{if } n \leq 1 \\ f^2(^{n+1}/_2) + f^2(^{n-1}/_2) & \text{if odd } n>1 \\ f^2(^{n}/_2+1) - f^2(^{n}/_2-1) & \text{if even } n>1 \end{cases}$

  f-sequence:
  0,1,1,2,3,5,8,13,21,34,55,89,144,…

Statement.
For all $n\geq 0$, fib(n) = f(n).

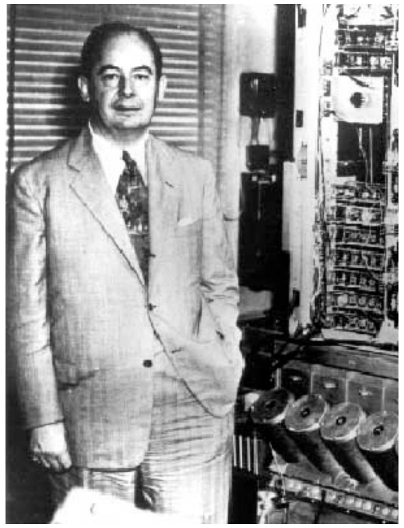
Left as an exercise…

# Recursive Algorithms

# Merge Sort

**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 |

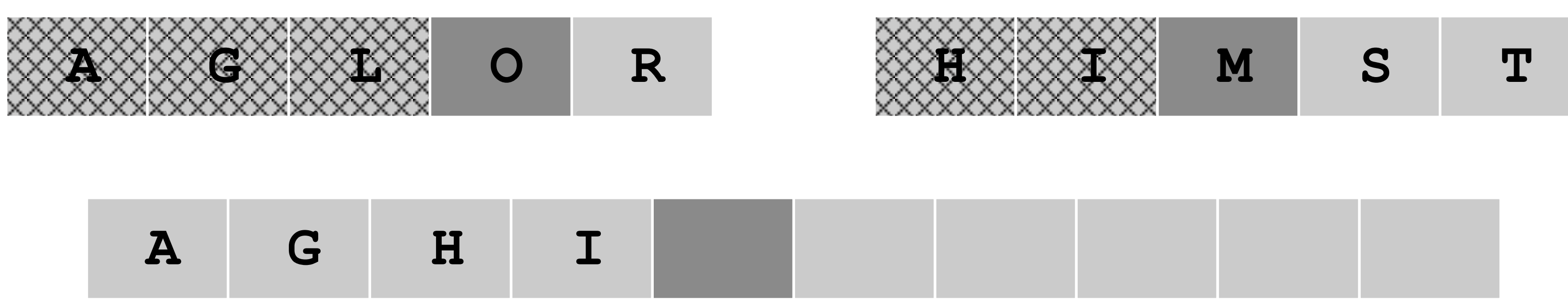
| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| A | L | G | O | R | I | T | H | M | S | divide | O(1) |
| A | G | L | O | R | H | I | M | S | T | sort | 2T(n/2) |
| A | G | H | I | L | M | O | R | S | T | merge | O(n) |

# Merge

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

How to merge efficiently?

- Linear number of comparisons.
- Use temporary array.



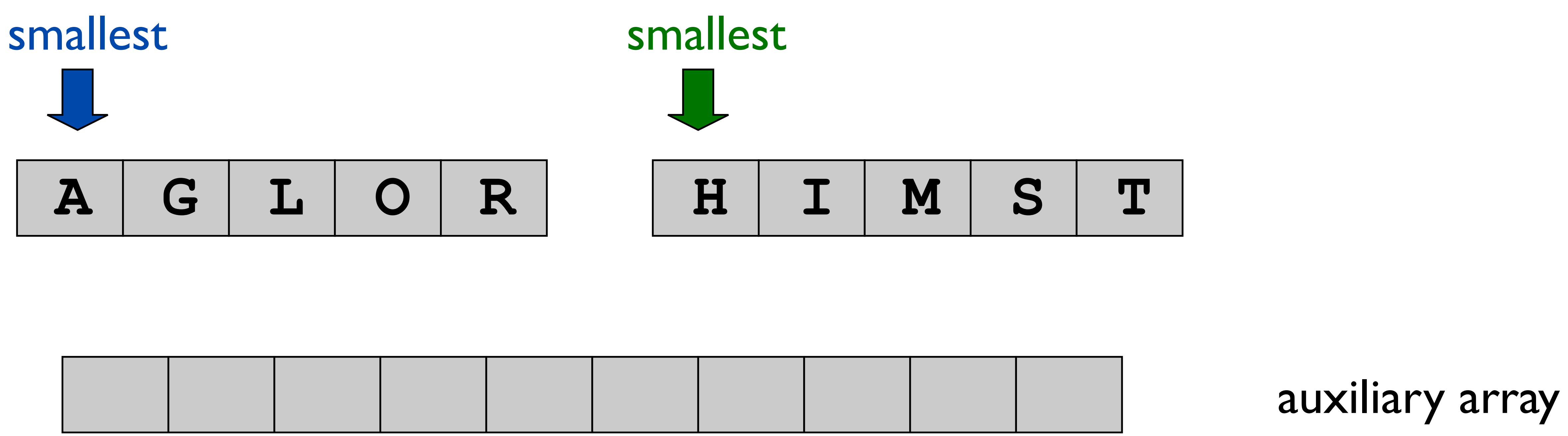

Challenge for the bored. In-place merge. [Kronrod, 1969]

using only a constant amount of extra storage

# Merge

Merging.

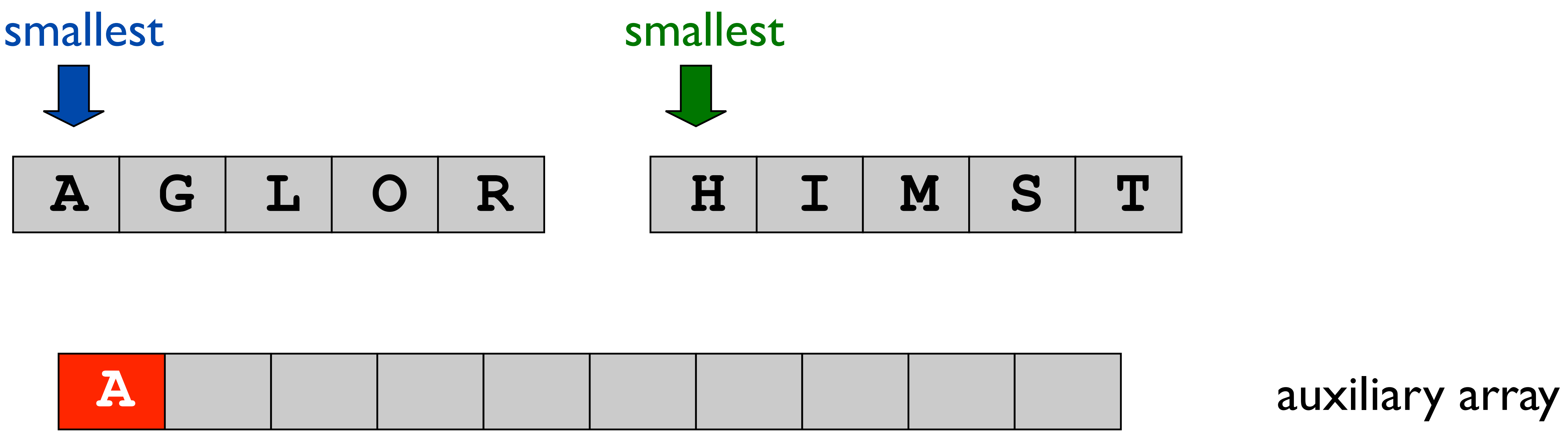
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

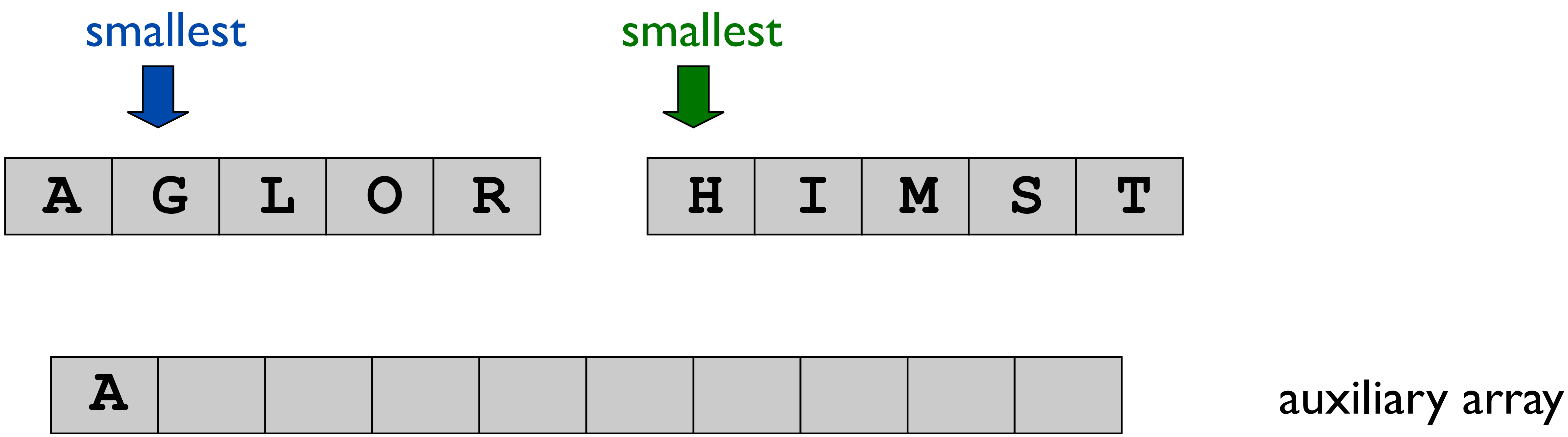
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

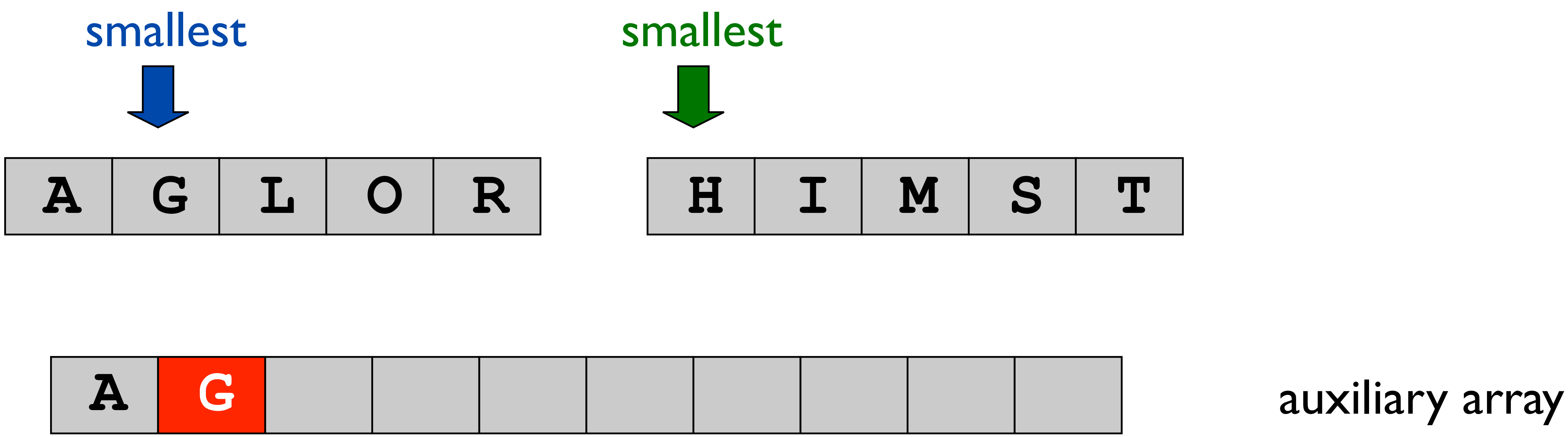
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

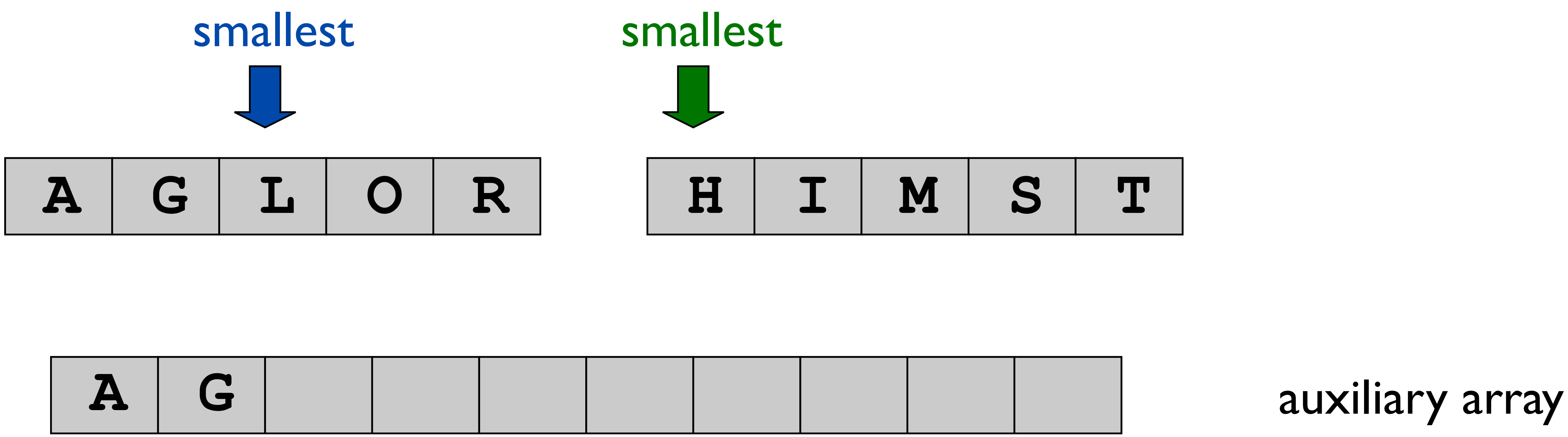
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

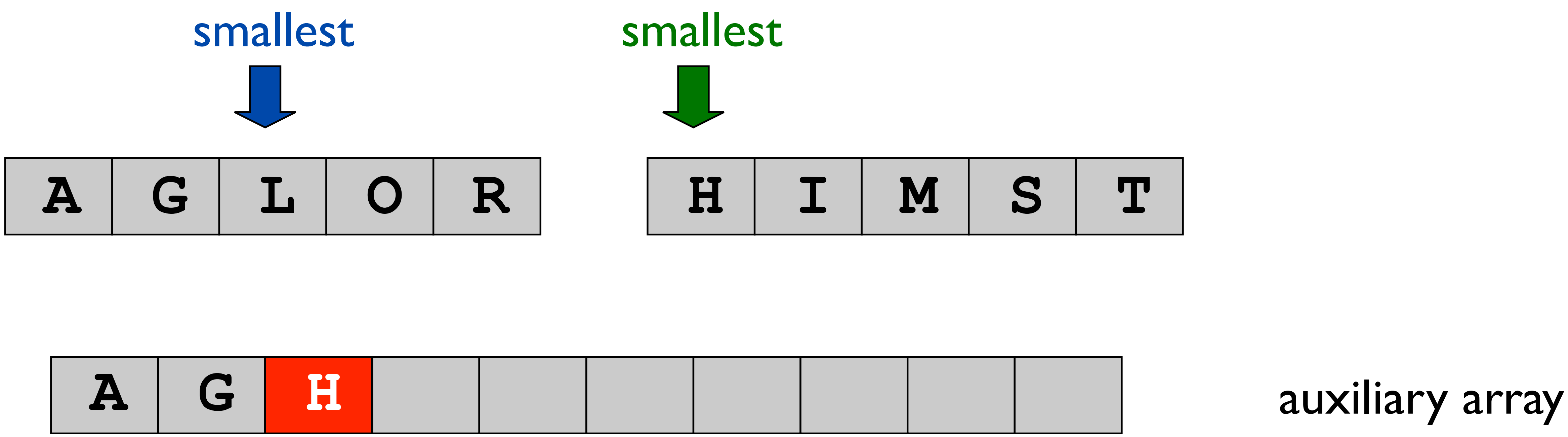
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

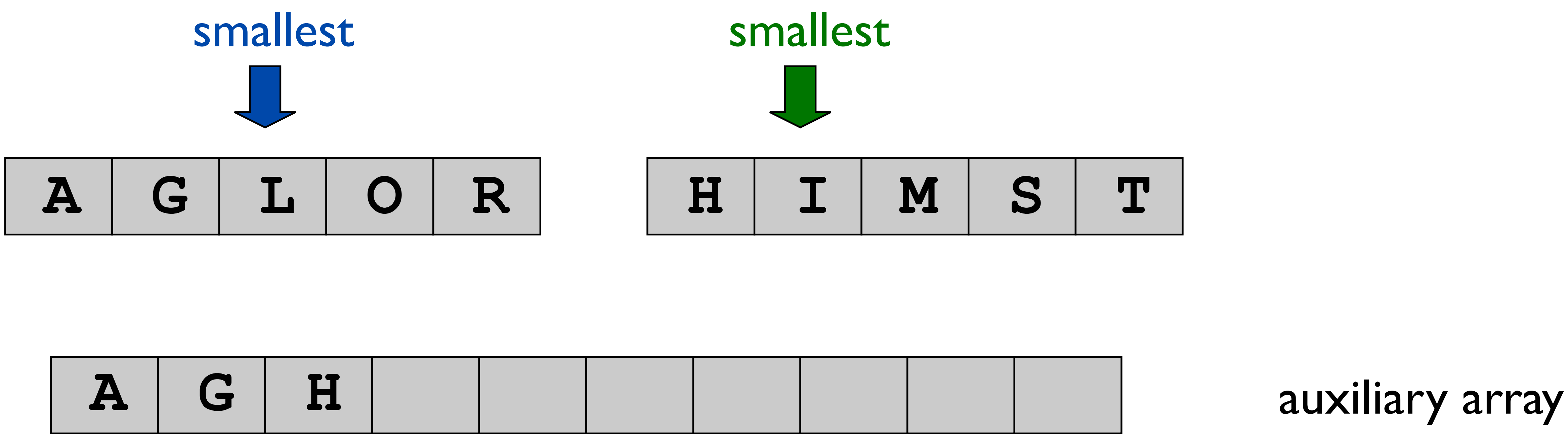
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

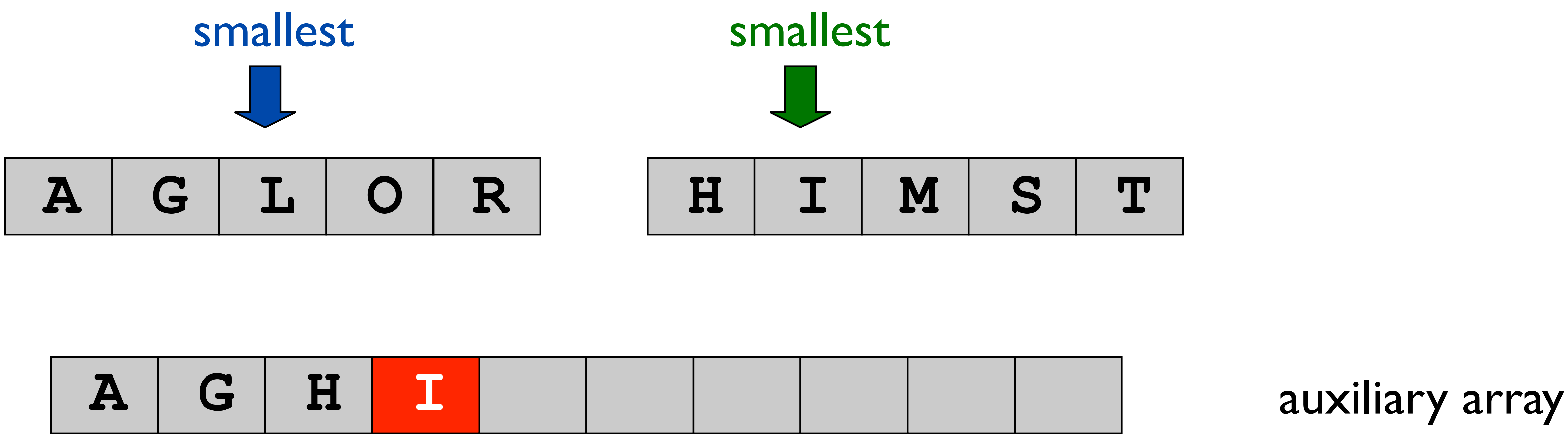
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

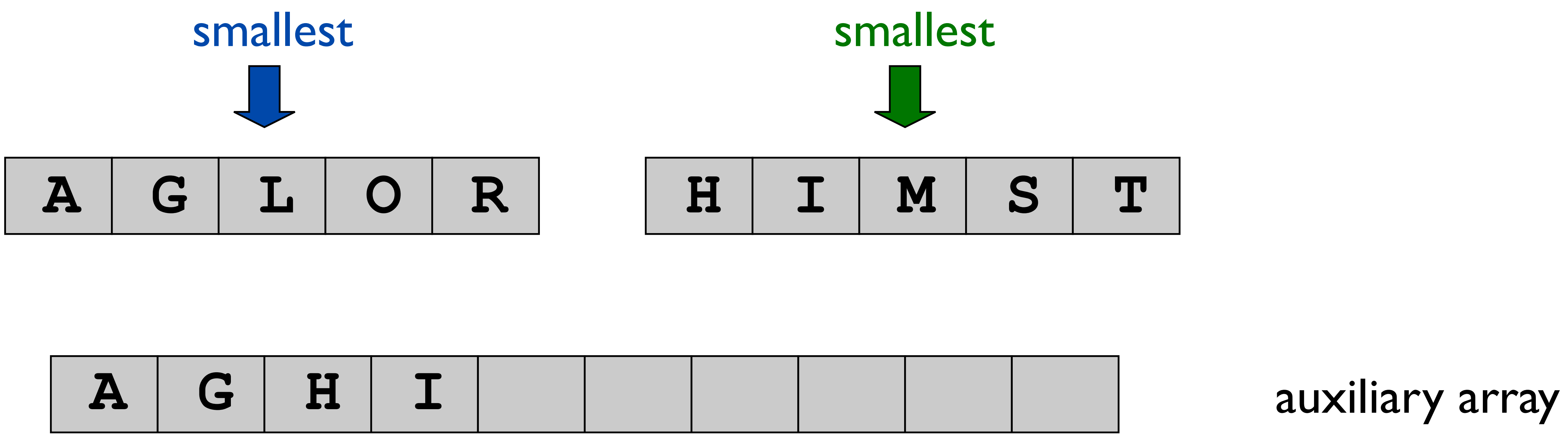
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

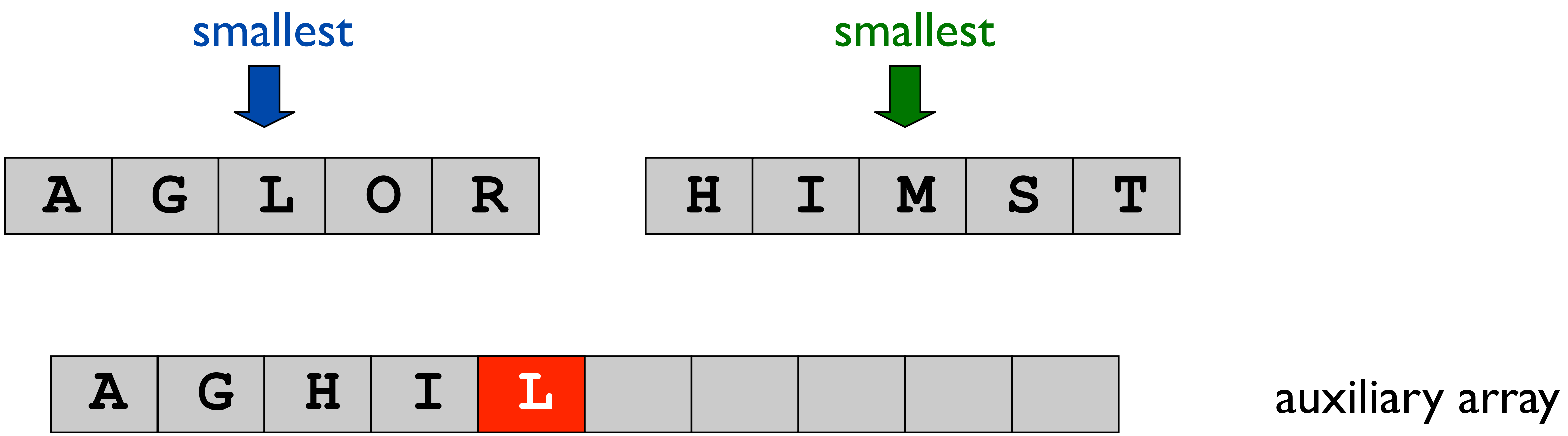
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

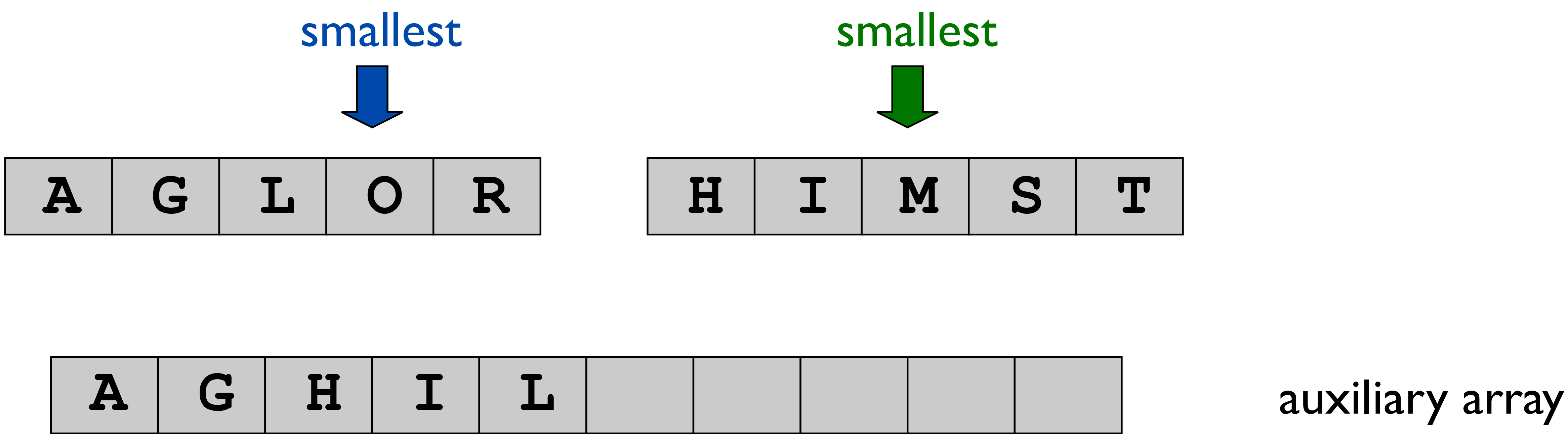
Merging.

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



# Merge

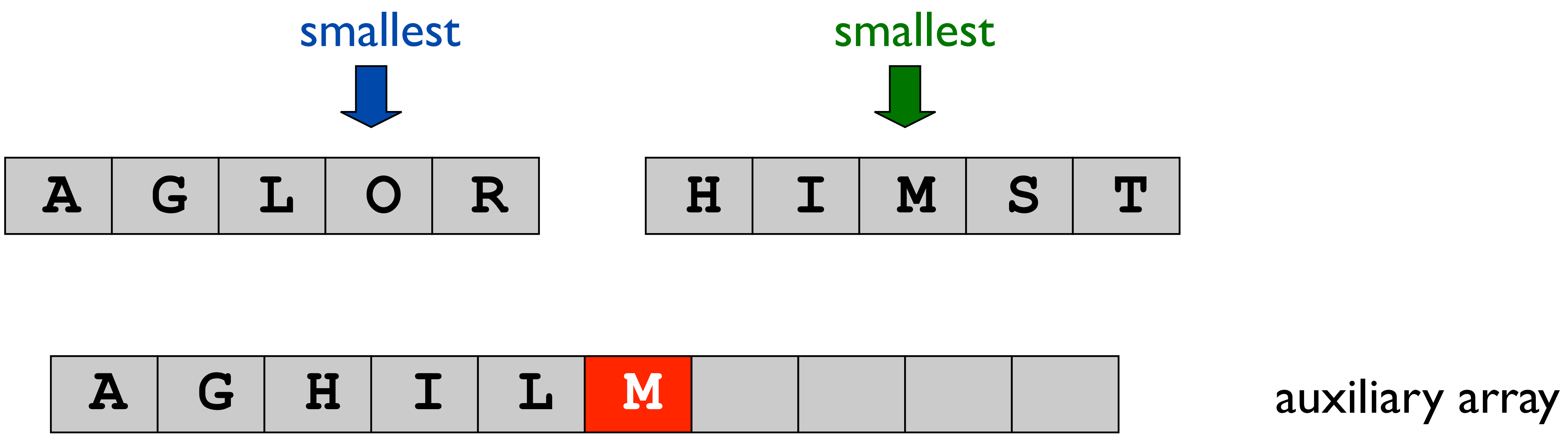
Merging.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

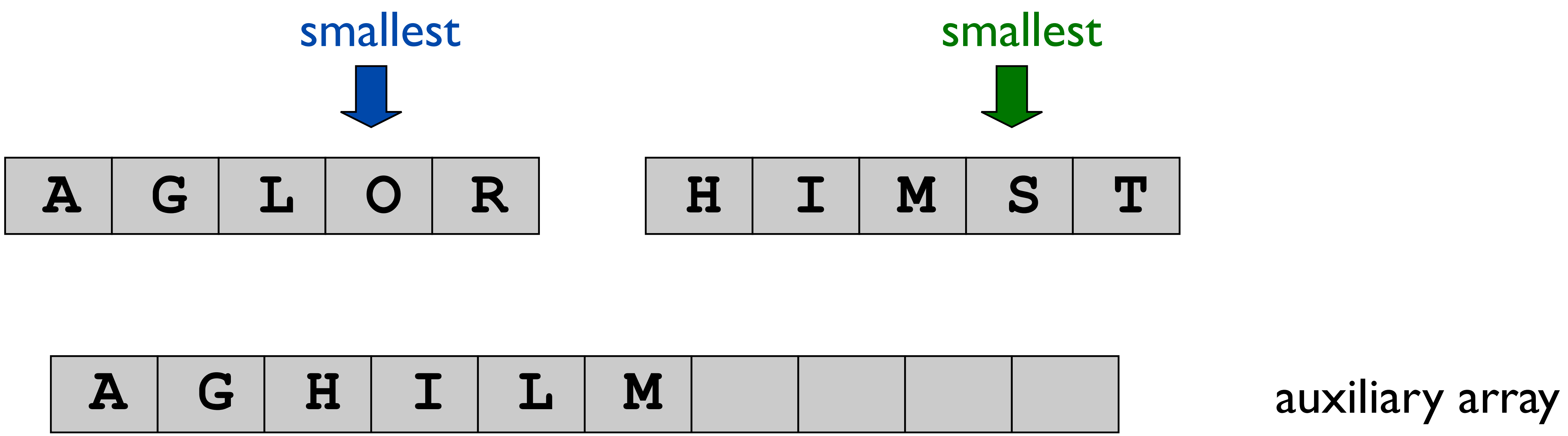
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

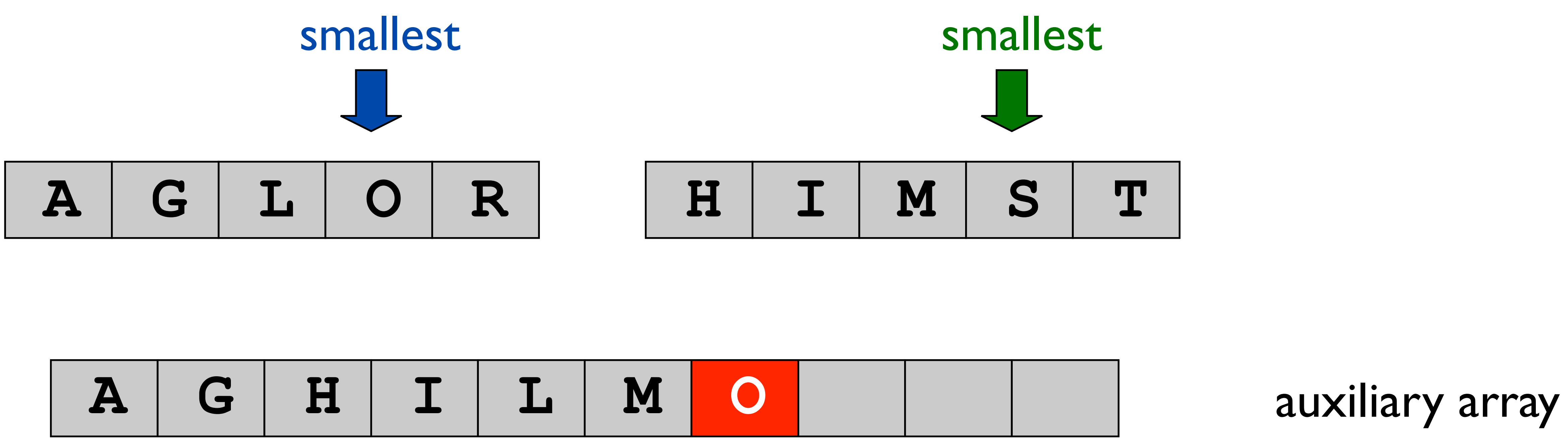
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

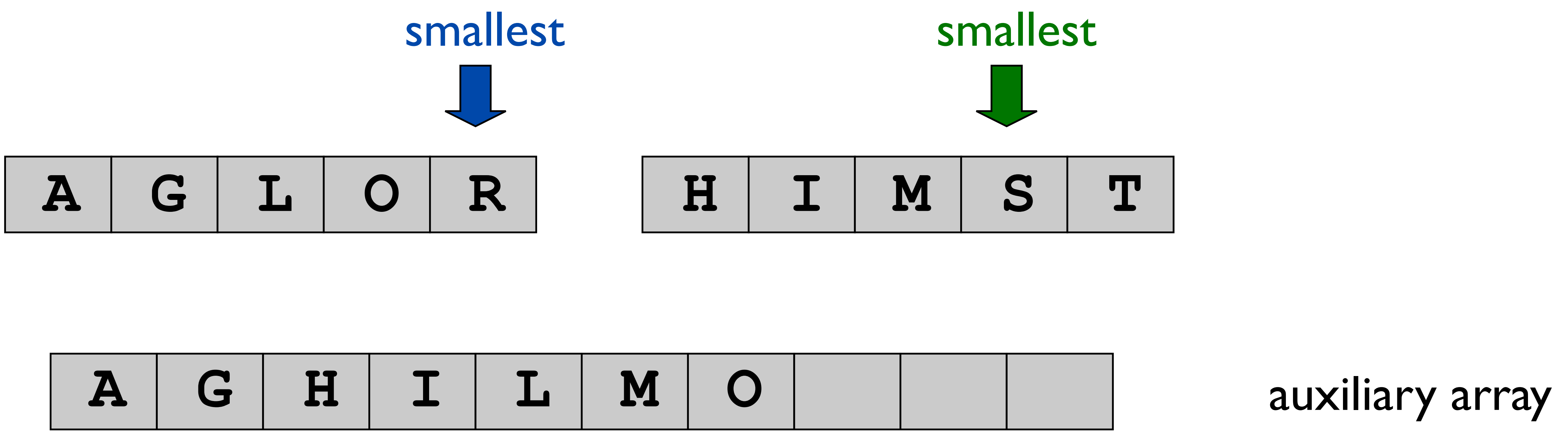
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

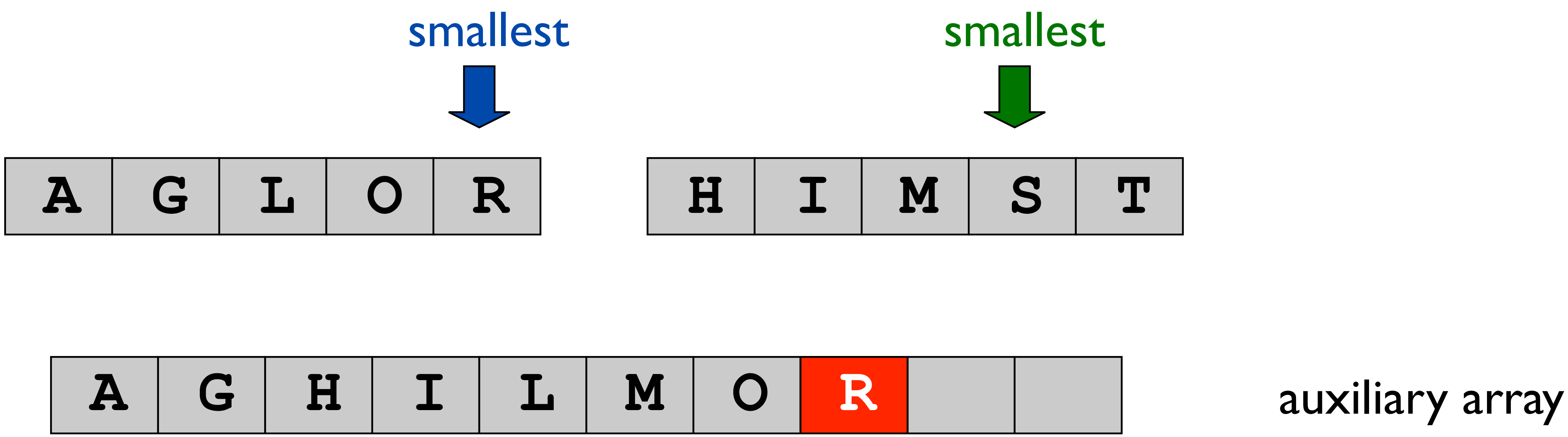
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

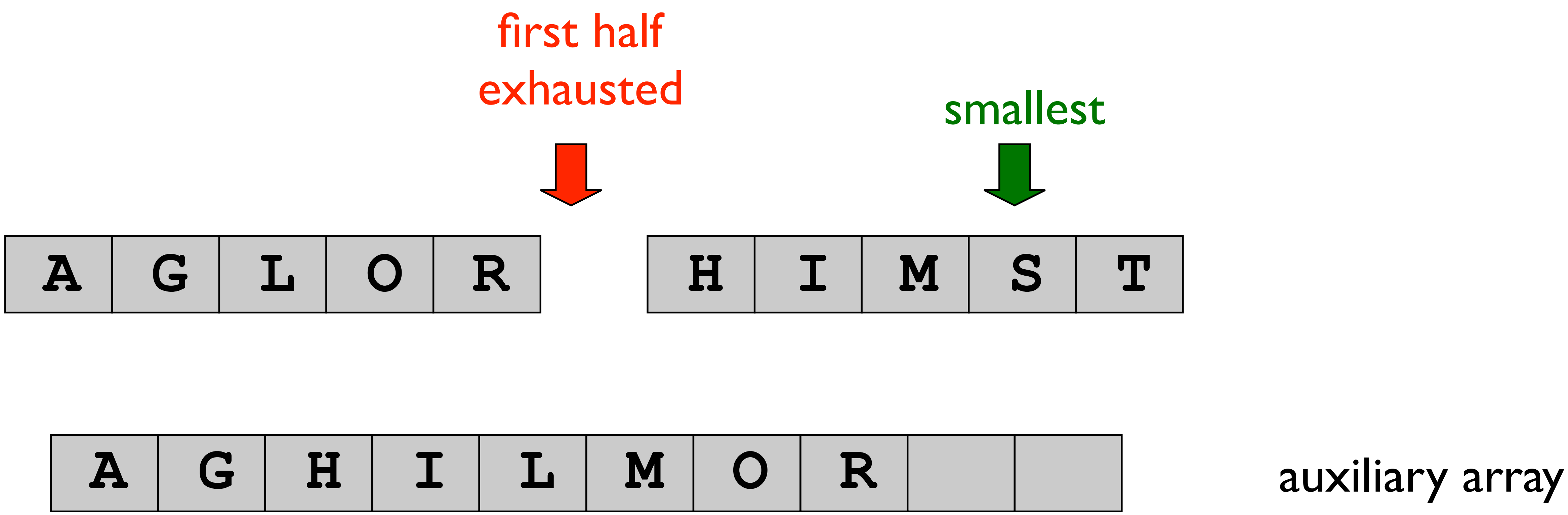
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

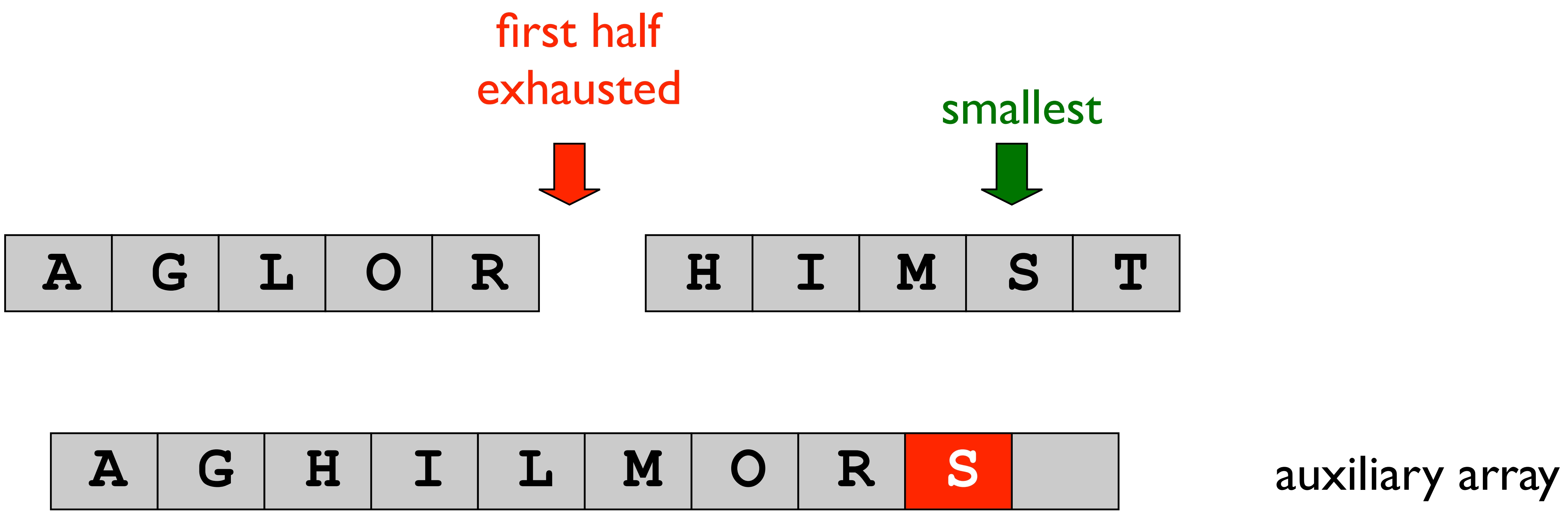
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

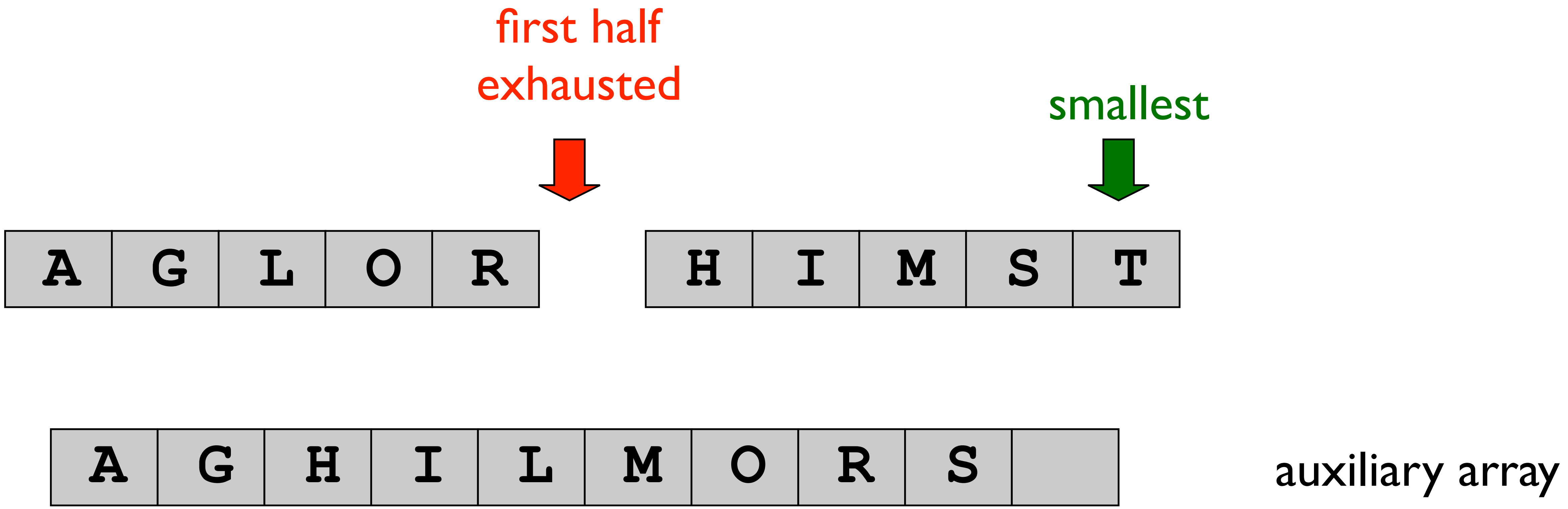
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

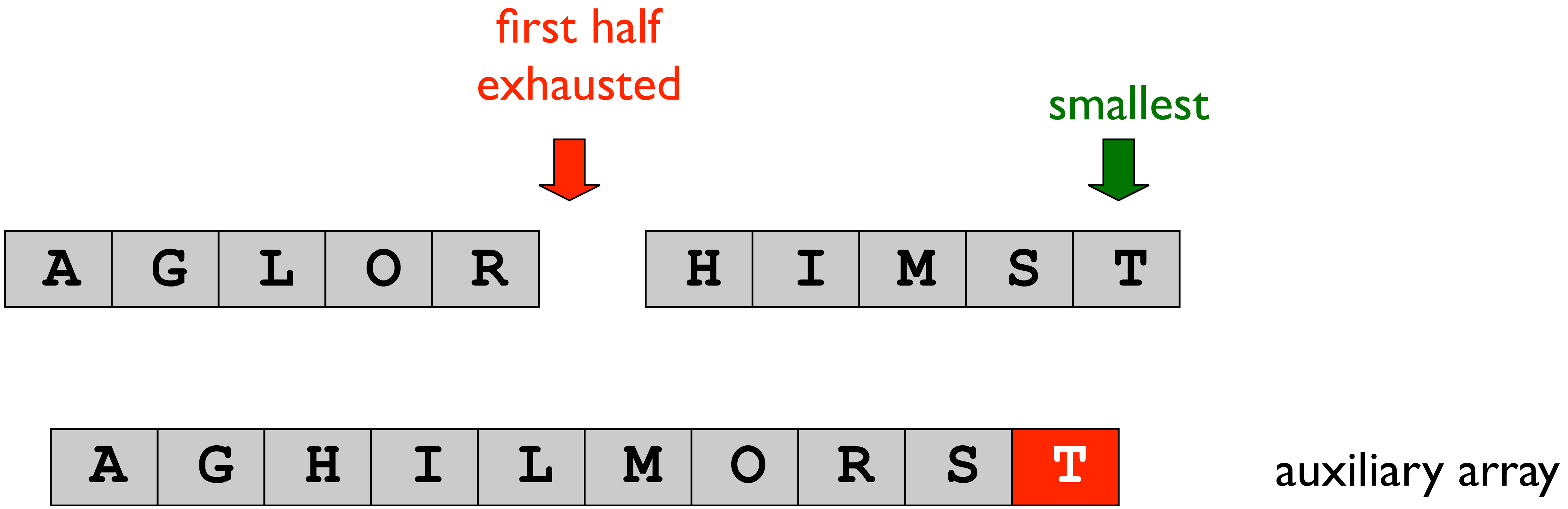
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

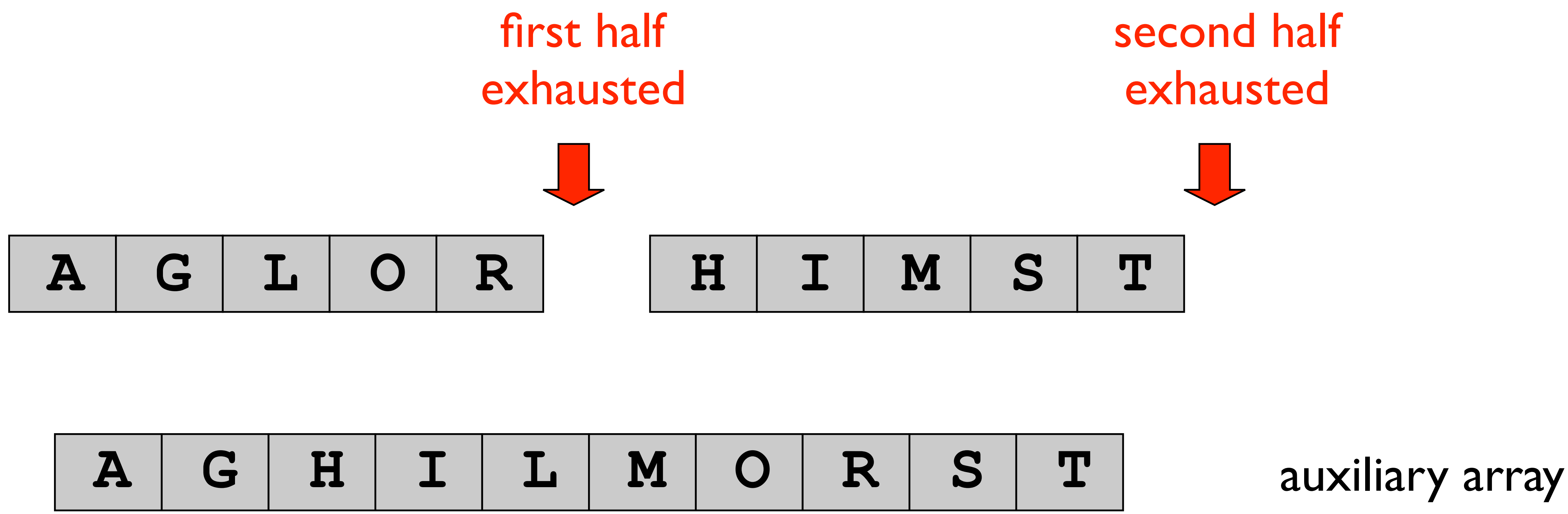
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



# Merge

Merging.

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



# 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) is $O(n \log_2 n)$.

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace ≤ with =.

# Telescoping Proof

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

↑ assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. For n > 1:

$$\begin{aligned} \frac{T(n)}{n} &= \frac{2T(n/2)}{n} + 1 \\ &= \frac{T(n/2)}{n/2} + 1 \\ &= \frac{T(n/4)}{n/4} + 1 + 1 \\ &\cdots \\ &= \frac{T(n/n)}{n/n} + \underbrace{1 + \cdots + 1}_{\log_2 n} \\ &= \log_2 n \end{aligned}$$

# Induction Proof

Claim. If T(n) satisfies this recurrence, then $T(n) = n \log_2 n$.

assumes n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$

Pf. (by induction on k such that $n=2^k$)

- Base case: $n = 2^0 = 1$.
- Inductive hypothesis: $T(n) = T(2^k) = n \log_2 n$.
- Goal: show that $T(2n) = T(2^{k+1}) = 2n \log_2 (2n)$.

$$\begin{aligned} T(2n) &= 2T(n) + 2n \\ &= 2n \log_2 n + 2n \\ &= 2n\left(\log_2(2n) - 1\right) + 2n \\ &= 2n \log_2(2n) \end{aligned}$$

# Generalized Induction Proof

**Claim.** If T(n) satisfies the following recurrence, then $T(n) \leq n \lceil \lg n \rceil$.

($\lg n$ = $\log_2 n$)

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

**Pf.** (by induction on n)

- Base case: $n = 1$. $T(1) = 0 = 1 \lceil \lg 1 \rceil$.
- Define $n_1 = \lfloor n / 2 \rfloor$, $n_2 = \lceil n / 2 \rceil$. (note $1 \leq n_1 < n$, $1 \leq n_2 < n$)
- Induction step: Let $n \geq 2$, assume true for 1, 2, ... , n–1.

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + n \\ &\leq n_1 \lceil \lg n_1 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &\leq n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\ &= n \lceil \lg n_2 \rceil + n \\ &\leq n( \lceil \lg n \rceil - 1 ) + n \\ &= n \lceil \lg n \rceil \end{aligned}$$

$$\begin{aligned} n_2 &= \lceil n/2 \rceil \\ &\leq \left\lceil 2^{\lceil \lg n \rceil} / 2 \right\rceil \\ &= 2^{\lceil \lg n \rceil} / 2 \\ \Rightarrow \lg n_2 &\leq \lceil \lg n \rceil - 1 \end{aligned}$$

# Winter 2016
# COMP-250: Introduction to Computer Science

Lecture 10, February 11, 2016