بسم الله الرحمن الرحیم

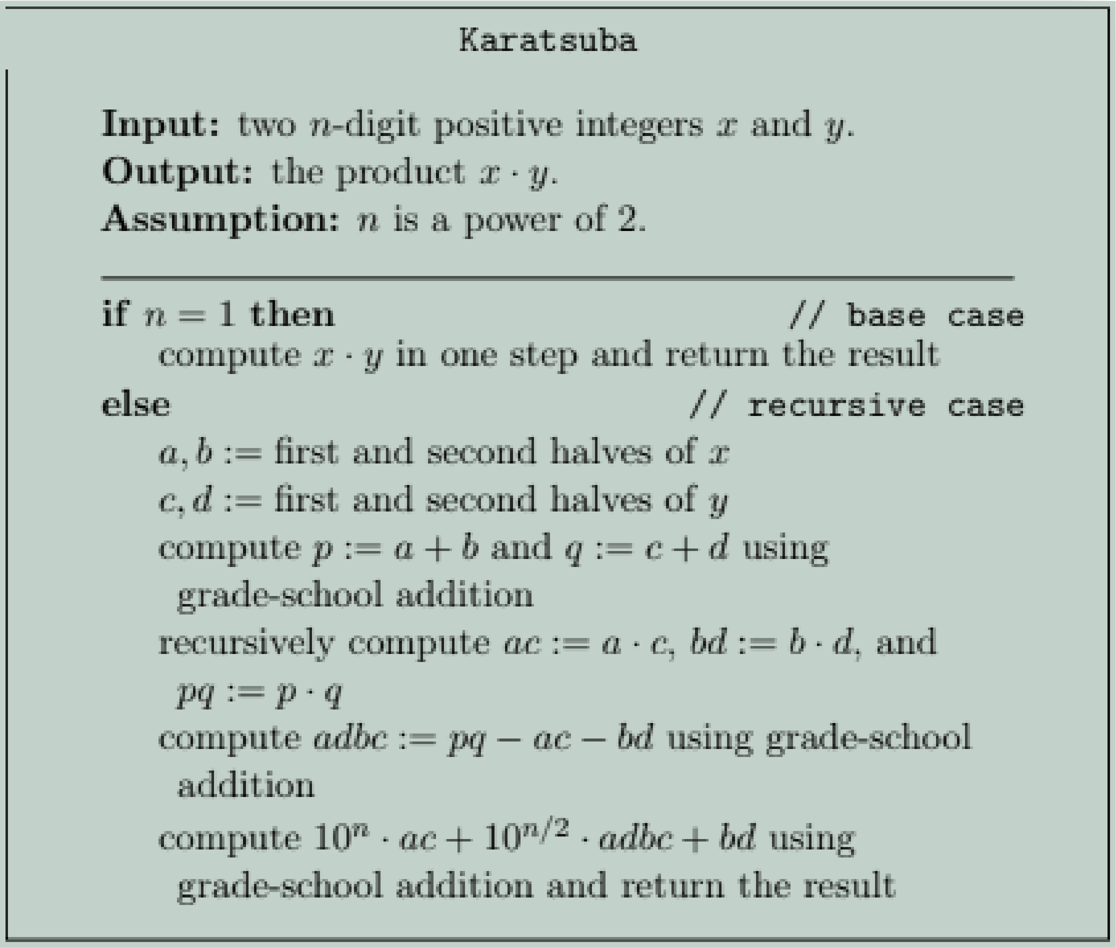
# طراحی الگوریتم

## (مرتب‌سازی ادغامی)

دانشکده مهندسی برق و کامپیوتر، دانشگاه صنعتی اصفهان

بهار ۱۴۰۰

# یادآوری: روش ضرب Karatsub (۱۹۶۰)



Karatsuba

**Input:** two $n$-digit positive integers $x$ and $y$.
**Output:** the product $x \cdot y$.
**Assumption:** $n$ is a power of 2.

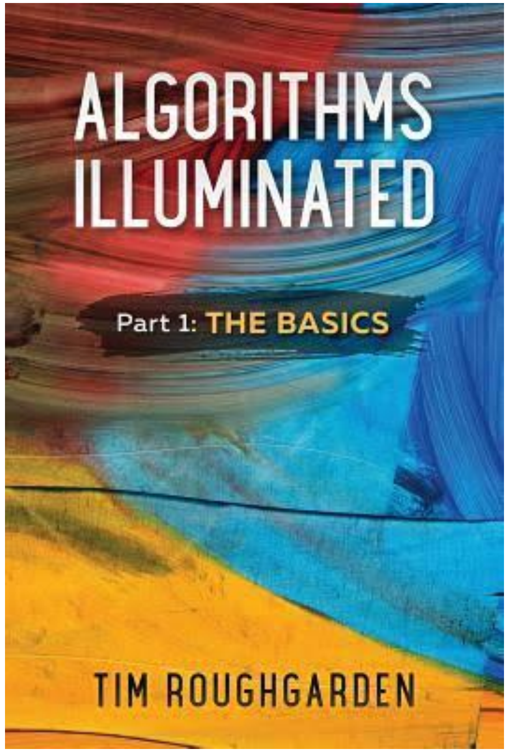
```
if $n = 1$ then                                   // base case
    compute $x \cdot y$ in one step and return the result
else                                              // recursive case
    $a, b :=$ first and second halves of $x$
    $c, d :=$ first and second halves of $y$
    compute $p := a + b$ and $q := c + d$ using
      grade-school addition
    recursively compute $ac := a \cdot c$, $bd := b \cdot d$, and
      $pq := p \cdot q$
    compute $adbc := pq - ac - bd$ using grade-school
      addition
    compute $10^n \cdot ac + 10^{n/2} \cdot adbc + bd$ using
      grade-school addition and return the result
```

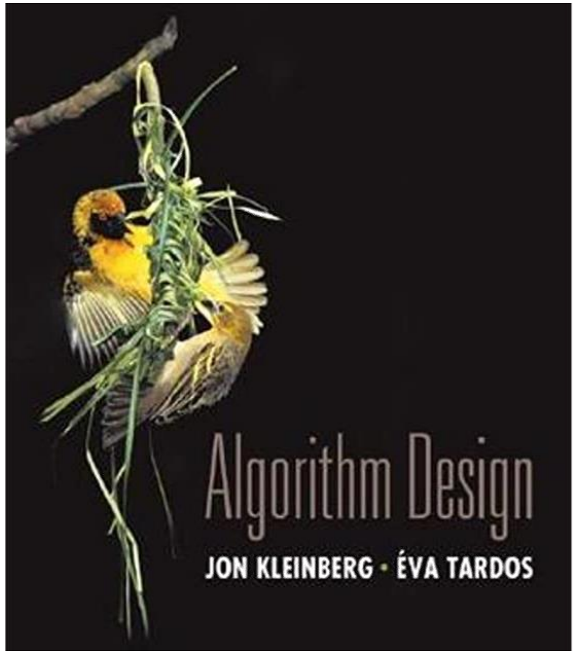
$$\begin{array}{r} 5678 \\ \times\ 1234 \\ \hline \end{array}$$

# مرتب‌سازی ادغامی

**ورودی:** یک دنباله از اعداد، با ترتیب دلخواه

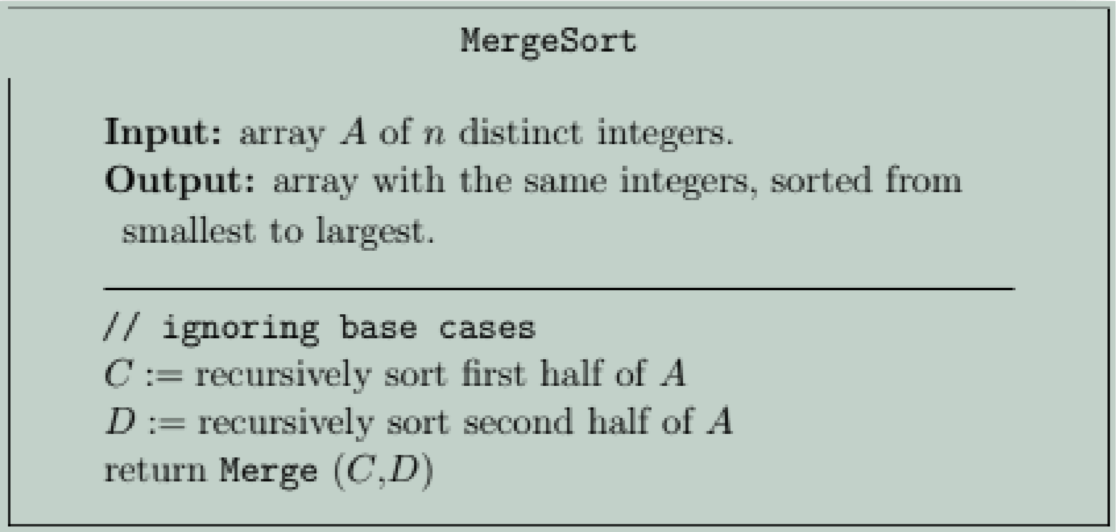
**هدف:** همان دنباله به صورت مرتب‌شده





**فصل اول، صفحه ۱۲**

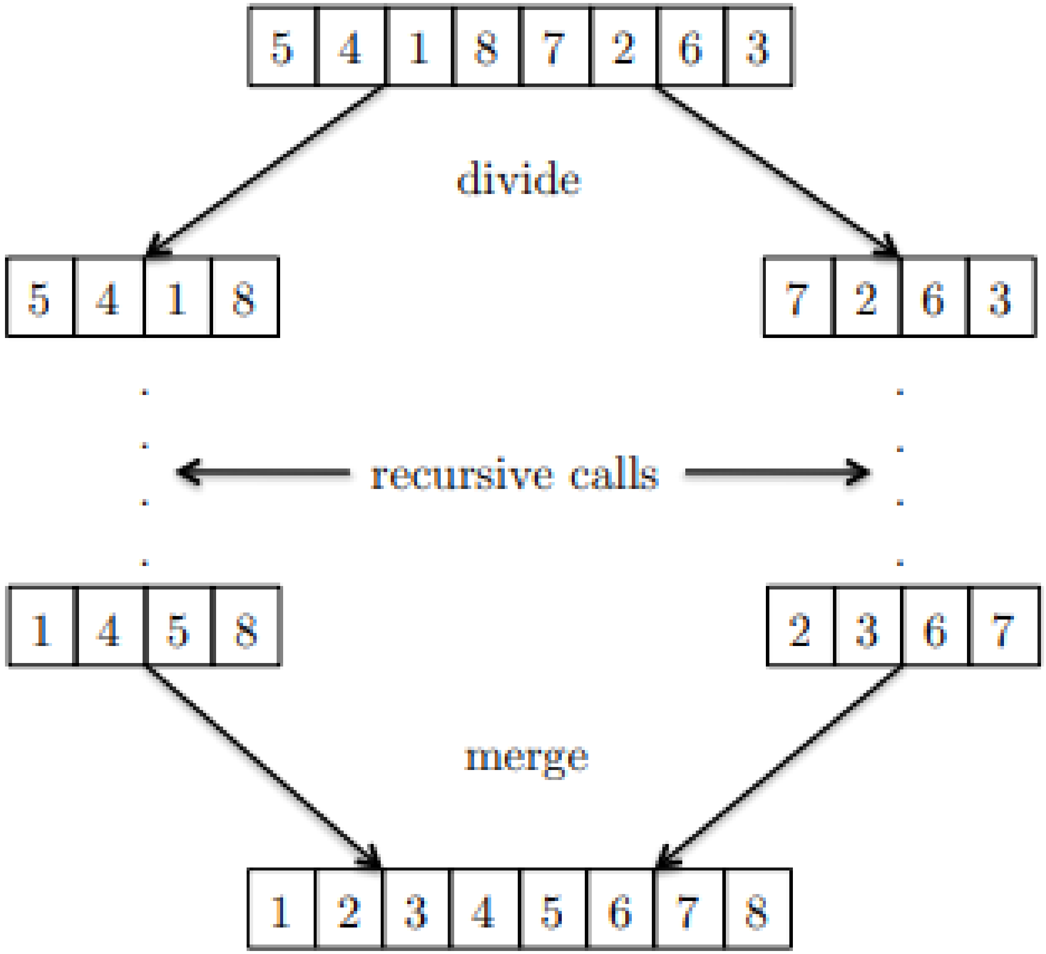
# مرتب‌سازی ادغامی



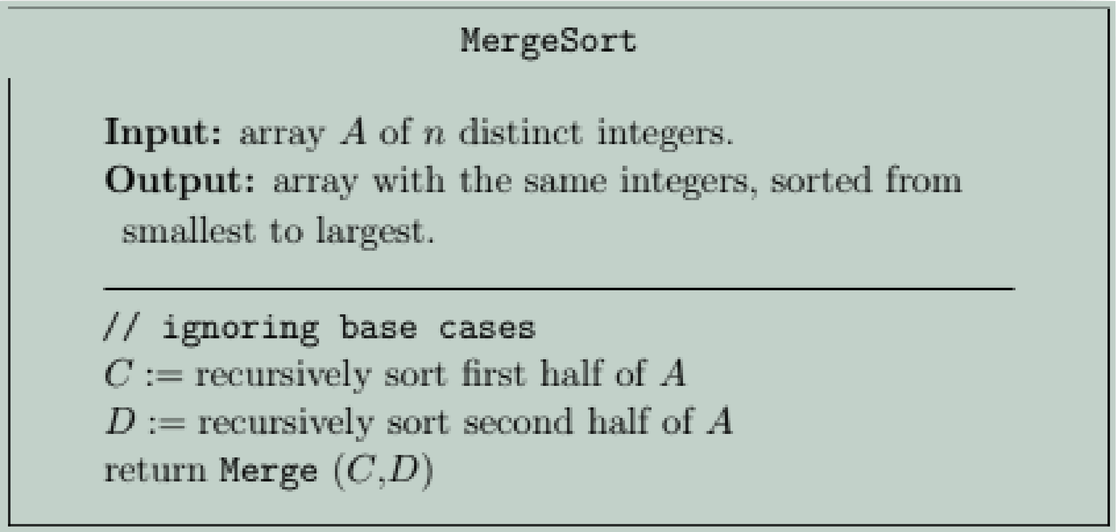
MergeSort

**Input:** array $A$ of $n$ distinct integers.
**Output:** array with the same integers, sorted from smallest to largest.

```
// ignoring base cases
$C$ := recursively sort first half of $A$
$D$ := recursively sort second half of $A$
return Merge ($C$,$D$)
```



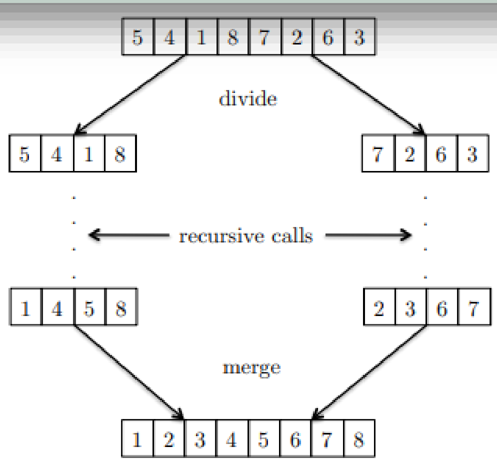
# مرتب‌سازی ادغامی

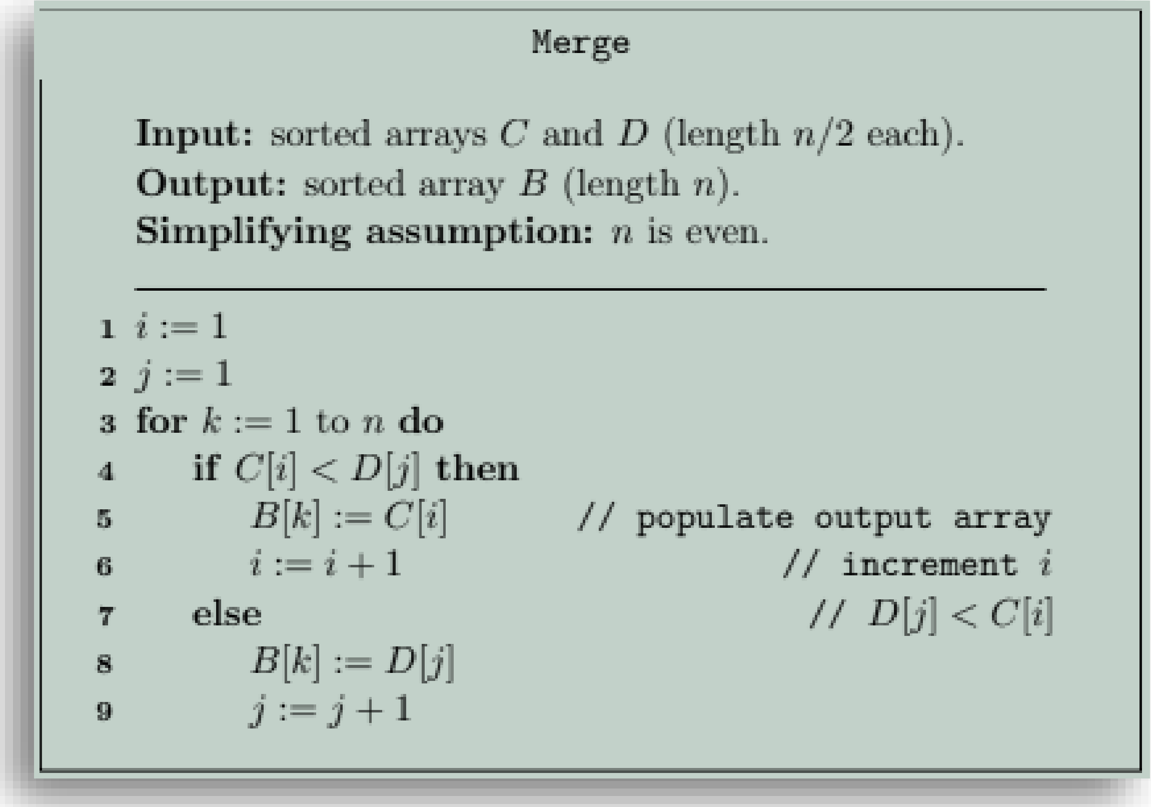


MergeSort

**Input:** array $A$ of $n$ distinct integers.
**Output:** array with the same integers, sorted from smallest to largest.

```
// ignoring base cases
$C$ := recursively sort first half of $A$
$D$ := recursively sort second half of $A$
return Merge ($C$,$D$)
```



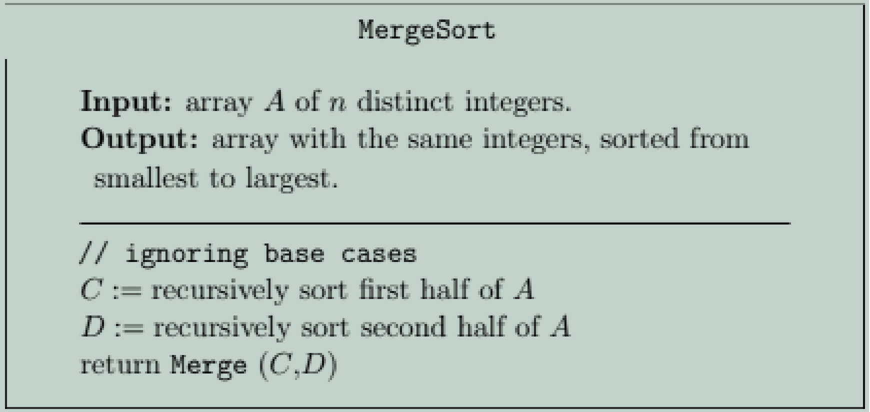


Merge

**Input:** sorted arrays $C$ and $D$ (length $n/2$ each).
**Output:** sorted array $B$ (length $n$).
**Simplifying assumption:** $n$ is even.

```
$i := 1$
$j := 1$
for $k := 1$ to $n$ do
    if $C[i] < D[j]$ then
        $B[k] := C[i]$          // populate output array
        $i := i + 1$                    // increment $i$
    else                                 // $D[j] < C[i]$
        $B[k] := D[j]$
        $j := j + 1$
```

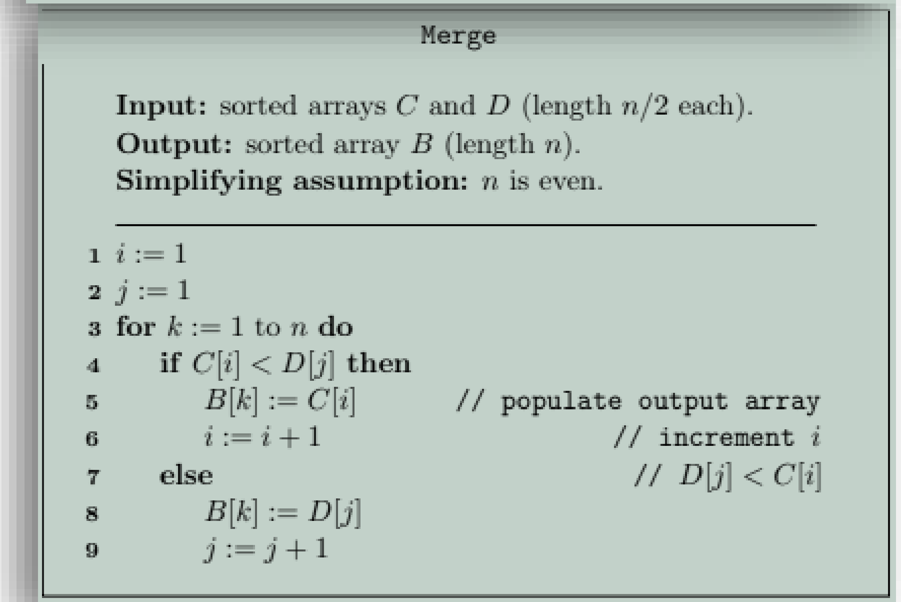
# مرتب‌سازی ادغامی



MergeSort

**Input:** array $A$ of $n$ distinct integers.
**Output:** array with the same integers, sorted from smallest to largest.

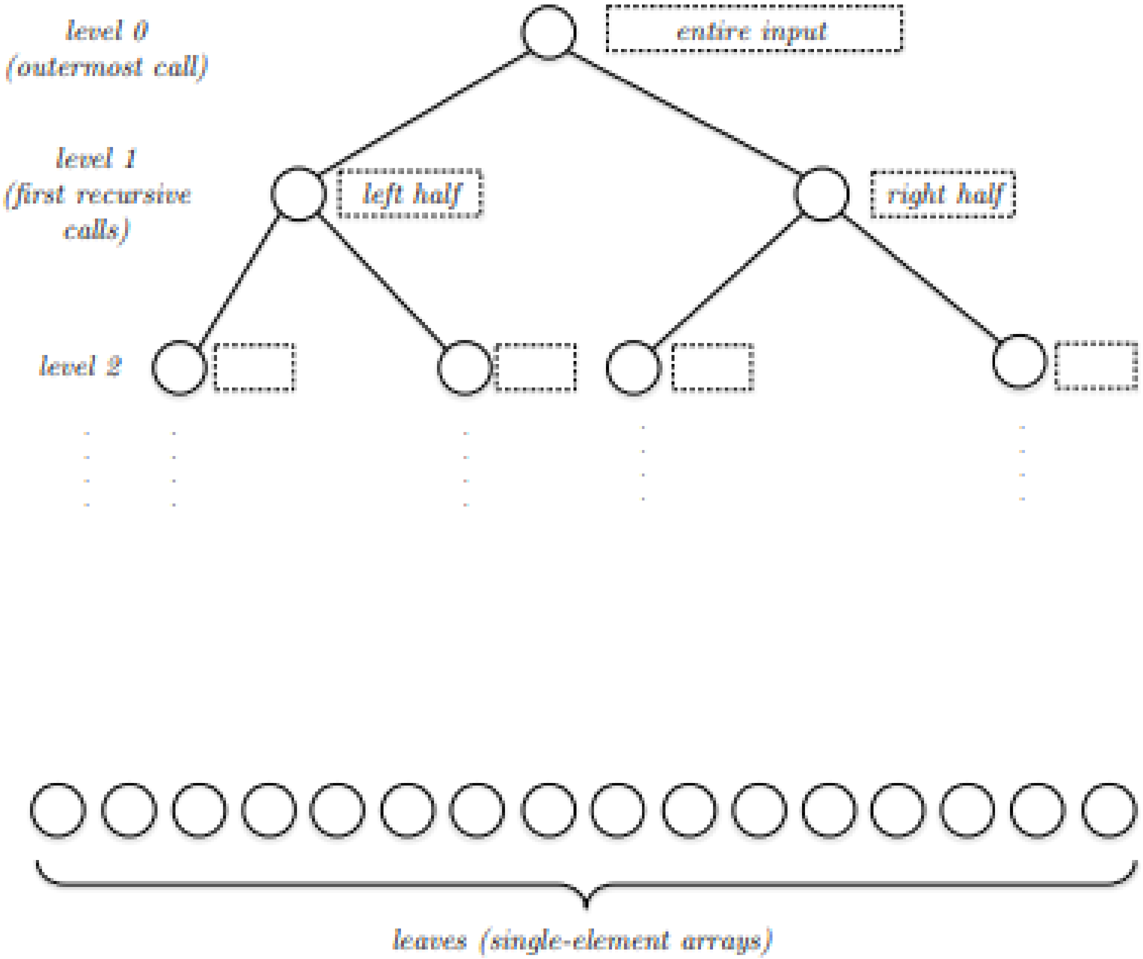
```
// ignoring base cases
$C$ := recursively sort first half of $A$
$D$ := recursively sort second half of $A$
return Merge ($C$,$D$)
```



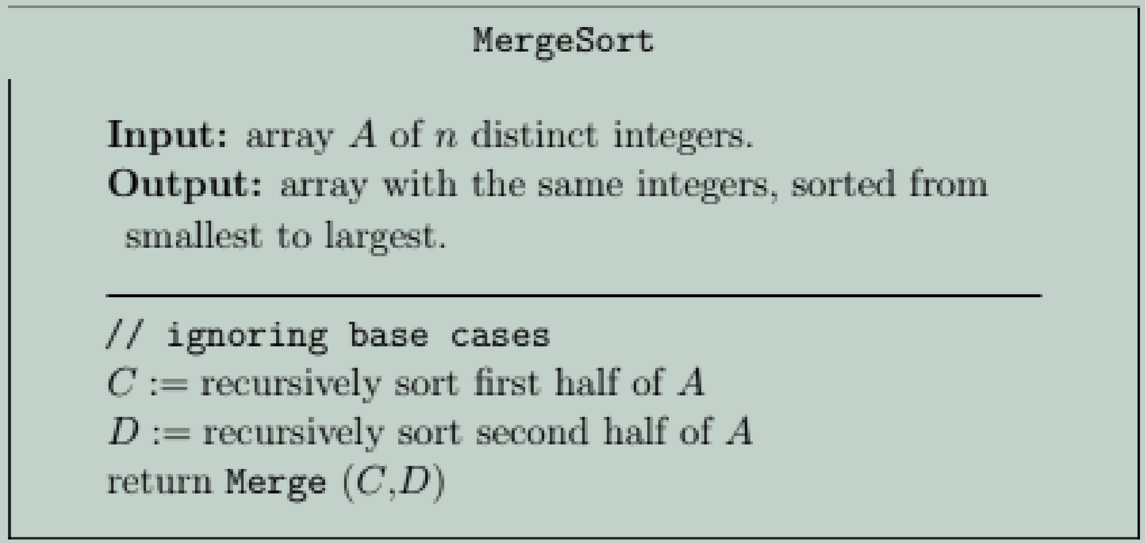
Merge

**Input:** sorted arrays $C$ and $D$ (length $n/2$ each).
**Output:** sorted array $B$ (length $n$).
**Simplifying assumption:** $n$ is even.

```
$i := 1$
$j := 1$
for $k := 1$ to $n$ do
    if $C[i] < D[j]$ then
        $B[k] := C[i]$          // populate output array
        $i := i + 1$                       // increment $i$
    else                                   // $D[j] < C[i]$
        $B[k] := D[j]$
        $j := j + 1$
```



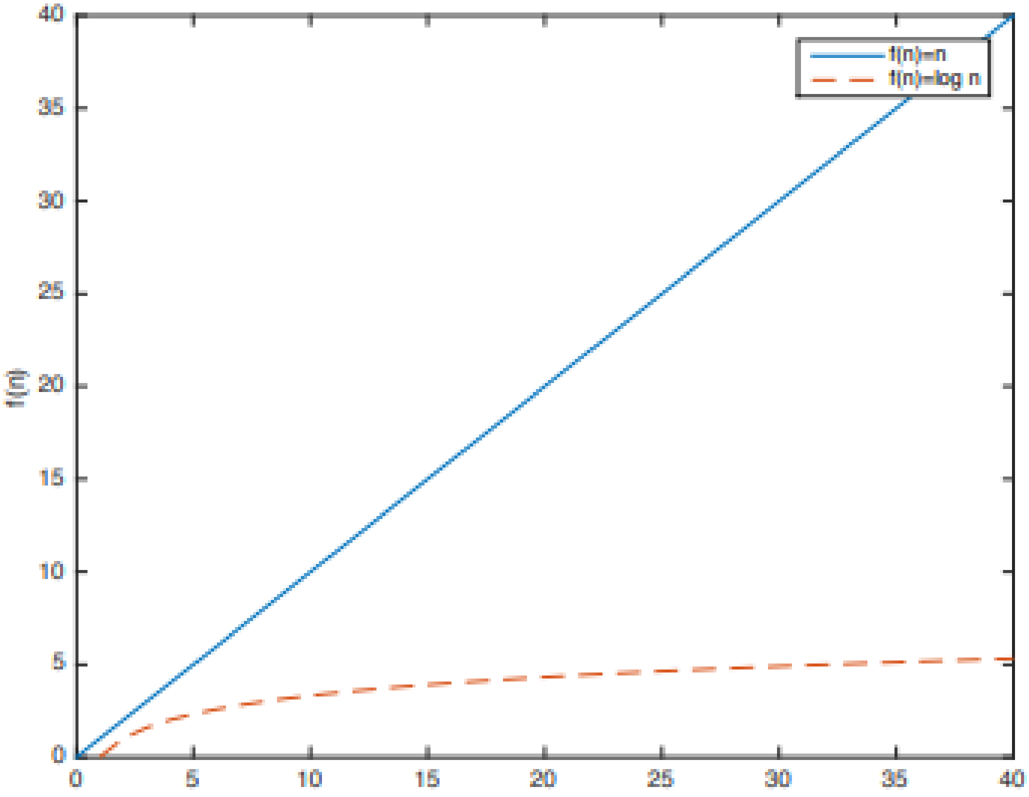
# مرتب‌سازی ادغامی

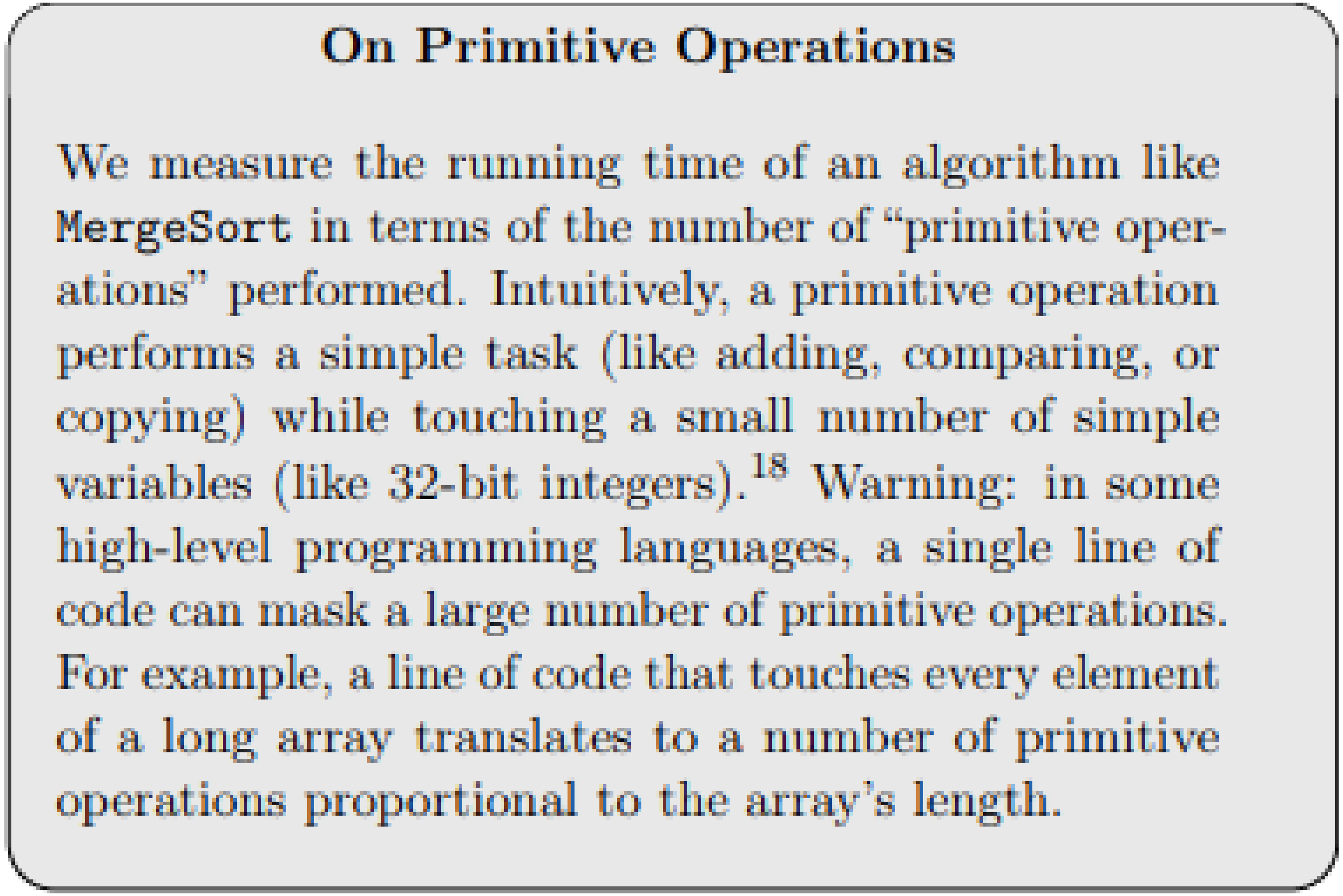


MergeSort

**Input:** array $A$ of $n$ distinct integers.
**Output:** array with the same integers, sorted from smallest to largest.

```
// ignoring base cases
$C$ := recursively sort first half of $A$
$D$ := recursively sort second half of $A$
return Merge ($C$,$D$)
```





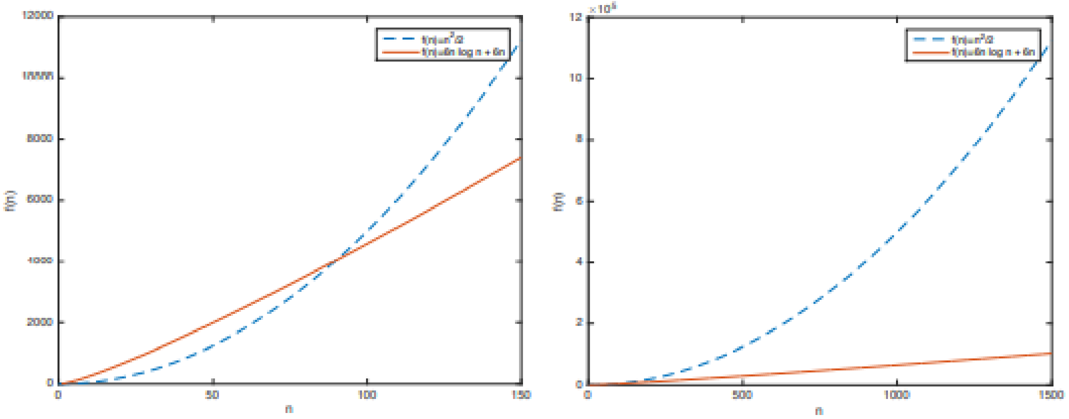
## On Primitive Operations

We measure the running time of an algorithm like `MergeSort` in terms of the number of "primitive operations" performed. Intuitively, a primitive operation performs a simple task (like adding, comparing, or copying) while touching a small number of simple variables (like 32-bit integers).[18] Warning: in some high-level programming languages, a single line of code can mask a large number of primitive operations. For example, a line of code that touches every element of a long array translates to a number of primitive operations proportional to the array's length.

# اصول راهنما برای تحلیل الگوریتم‌ها

**اصل اول:** تحلیل در بدترین حالت

**اصل دوم:** تحلیل مجانبی



(a) Small values of $n$

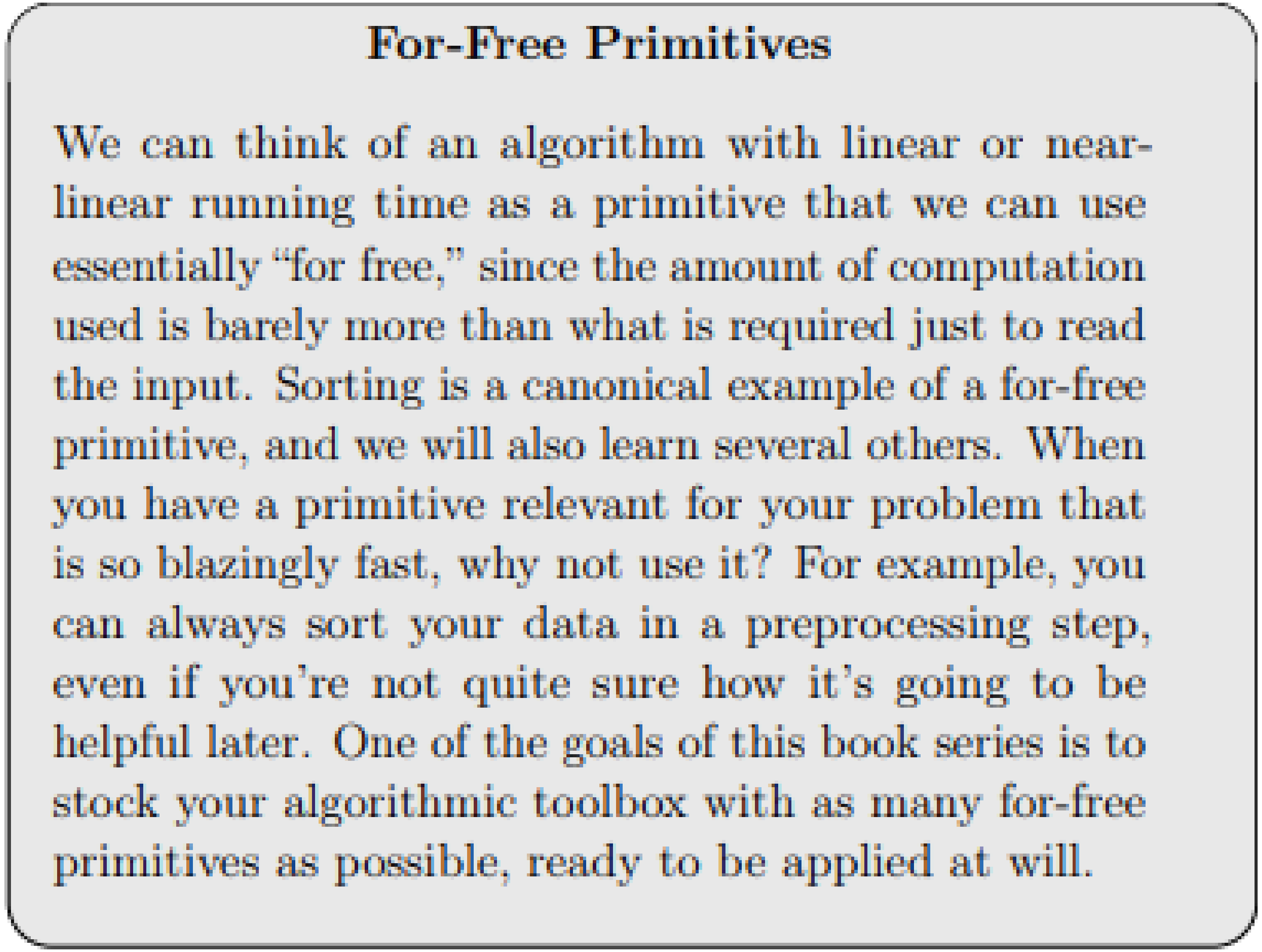
(b) Medium values of $n$

# چه الگوریتمی "سریع" است؟

یک "الگوریتم سریع" الگوریتمی است که زمان اجرای آن در بدترین حالت نسبت به سایز ورودی با سرعت کمی رشد نماید.

# چه الگوریتمی "سریع" است؟

یک "الگوریتم سریع" الگوریتمی است که زمان اجرای آن در بدترین حالت نسبت به سایز ورودی با سرعت کمی رشد نماید.



## For-Free Primitives

We can think of an algorithm with linear or near-linear running time as a primitive that we can use essentially "for free," since the amount of computation used is barely more than what is required just to read the input. Sorting is a canonical example of a for-free primitive, and we will also learn several others. When you have a primitive relevant for your problem that is so blazingly fast, why not use it? For example, you can always sort your data in a preprocessing step, even if you're not quite sure how it's going to be helpful later. One of the goals of this book series is to stock your algorithmic toolbox with as many for-free primitives as possible, ready to be applied at will.