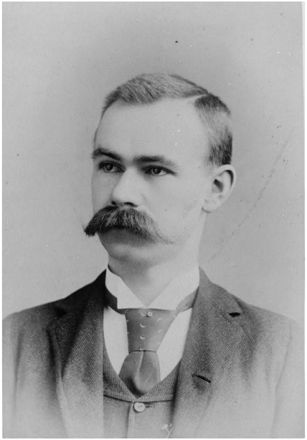
Intro to Analysis of Algorithms
Divide & Conquer
Chapter 3

Michael Soltys

CSU Channel Islands

[Ed: 4th, last updated: September 16, 2025]



Herman Hollerith, 1860–1929

Suppose that we have two lists of numbers that are already sorted.

That is, we have a list $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_m$.

We want to combine those two lists into one long sorted list $c_1 \leq c_2 \leq \cdots \leq c_{n+m}$.

The mergesort algorithm sorts a given list of numbers by first dividing them into two lists of length $\lceil n/2 \rceil$ and $\lfloor n/2 \rfloor$, respectively, then sorting each list recursively, and finally combining the results.

Pre-condition: $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_m$

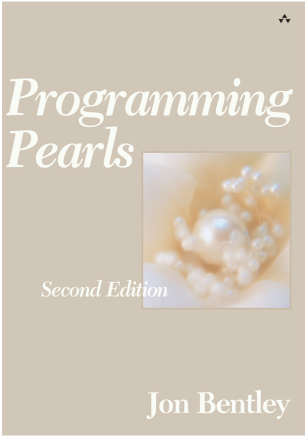
$p_1 \longleftarrow 1$; $p_2 \longleftarrow 1$; $i \longleftarrow 1$
while $i \leq n + m$ **do**
    if $a_{p_1} \leq b_{p_2}$ **then**
        $c_i \longleftarrow a_{p_1}$
        $p_1 \longleftarrow p_1 + 1$
    else
        $c_i \longleftarrow b_{p_1}$
        $p_2 \longleftarrow p_2 + 1$
    end if
    $i \longleftarrow i + 1$
end while

Post-condition: $c_1 \leq c_2 \leq \cdots \leq c_{n+m}$

Pre-condition: A list of integers $a_1, a_2, \ldots, a_n$

```
L ⟵ a_1, a_2, ..., a_n
if |L| ≤ 1 then
        return L
else
        L_1 ⟵ first ⌈n/2⌉ elements of L
        L_2 ⟵ last ⌊n/2⌋ elements of L
        return Merge(Mergesort(L_1), Mergesort(L_2))
end if
```

Post-condition: $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_n}$


Programming
Pearls
Second Edition
Jon Bentley

Multiplication

	1	2	3	4	5	6	7	8
x					1	1	1	0
y					1	1	0	1
s_1					1	1	1	0
s_2				0	0	0	0	
s_3			1	1	1	0		
s_4		1	1	1	0			
$x \times y$	1	0	1	1	0	1	1	0

Multiply 1110 times 1101, i.e., 14 times 13. Takes $O(n^2)$ steps.

Clever multiplication

Let x and y be two n-bit integers. We break them up into two smaller $n/2$-bit integers as follows:

$$x = (x_1 \cdot 2^{n/2} + x_0),$$
$$y = (y_1 \cdot 2^{n/2} + y_0).$$

x_1 and y_1 correspond to the high-order bits of x and y, respectively, and x_0 and y_0 to the low-order bits of x and y, respectively.

The product of x and y appears as follows in terms of those parts:

$$\begin{aligned} xy &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0. \end{aligned} \tag{1}$$

A divide and conquer procedure appears surreptitiously. To compute the product of x and y we compute the four products $x_1y_1, x_1y_0, x_0y_1, x_0y_0$, *recursively*, and then we combine them to obtain xy.

Let $T(n)$ be the number of operations that are required to compute the product of two n-bit integers using the divide and conquer procedure:

$$T(n) \leq 4T(n/2) + cn, \tag{2}$$

since we have to compute the four products $x_1y_1, x_1y_0, x_0y_1, x_0y_0$ (this is where the $4T(n/2)$ factor comes from), and then we have to perform three additions of n-bit integers (that is where the factor cn, where c is some constant, comes from).

Notice that we do not take into account the product by 2^n and $2^{n/2}$ as they simply consist in shifting the binary string by an appropriate number of bits to the left (n for 2^n and $n/2$ for $2^{n/2}$). These shift operations are inexpensive, and can be ignored in the complexity analysis.

It appears that we have to make four recursive calls; that is, we need to compute the four multiplications $x_1y_1, x_1y_0, x_0y_1, x_0y_0$.

But we can get away with only three multiplications, and hence three recursive calls: x_1y_1, x_0y_0 and $(x_1+x_0)(y_1+y_0)$; the reason being that

$$(x_1y_0 + x_0y_1) = (x_1+x_0)(y_1+y_0) - (x_1y_1 + x_0y_0). \tag{3}$$

	multiplications	additions	shifts
Method 1	4	3	2
Method 2	3	4	2

Algorithm takes $T(n) \leq 3T(n/2) + dn$ operations.

Thus, the running time is $O(n^{\log 3}) \approx O(n^{1.59})$.

Recursive Binary Mult A3.3

Pre-condition: Two n-bit integers x and y

if $n = 1$ **then**
  if $x = 1 \wedge y = 1$ **then**
    return 1
  else
    return 0
  end if
end if
$(x_1, x_0) \longleftarrow$ (first $\lfloor n/2 \rfloor$ bits, last $\lceil n/2 \rceil$ bits) of x
$(y_1, y_0) \longleftarrow$ (first $\lfloor n/2 \rfloor$ bits, last $\lceil n/2 \rceil$ bits) of y
$z_1 \longleftarrow \mathrm{Multiply}(x_1 + x_0, y_1 + y_0)$
$z_2 \longleftarrow \mathrm{Multiply}(x_1, y_1)$
$z_3 \longleftarrow \mathrm{Multiply}(x_0, y_0)$
return $z_2 \cdot 2^n + (z_1 - z_2 - z_3) \cdot 2^{\lceil n/2 \rceil} + z_3$

Savitch's Algorithm

We have a directed graph, and we want to establish whether we have a path from s to t.

Savitch's algorithm solves the problem in *space* $O(\log^2 m)$.

$$\mathrm{R}(G, u, v, i) \iff (\exists w)[\mathrm{R}(G, u, w, i-1) \wedge \mathrm{R}(G, w, v, i-1)]. \quad (4)$$

```
if i = 0 then
    if u = v then
        return T
    else if (u, v) is an edge then
        return T
    end if
else
    for every vertex w do
        if R(G, u, w, i − 1) and R(G, w, v, i − 1) then
            return T
        end if
    end for
end if
return F
```

Example run

$\bullet^1 \text{ ——— } \bullet^2 \text{ ——— } \bullet^3 \text{ ——— } \bullet^4$

Then the recursion stack would look as follows for the first 6 steps:

		$R(1,4,0)$	F	$R(2,4,0)$	F
		$R(1,1,0)$	T	$R(1,2,0)$	T
	$R(1,4,1)$	$R(1,4,1)$	$R(1,4,1)$	$R(1,4,1)$	$R(1,4,1)$
	$R(1,1,1)$	$R(1,1,1)$	$R(1,1,1)$	$R(1,1,1)$	$R(1,1,1)$
$R(1,4,2)$	$R(1,4,2)$	$R(1,4,2)$	$R(1,4,2)$	$R(1,4,2)$	$R(1,4,2)$
Step 1	Step 2	Step 3	Step 4	Step 5	Step 6

Quicksort & git bisect

```
qsort [] = []
qsort (x:xs) = qsort smaller ++ [x] ++ qsort larger
  where
    smaller = [a | a <- xs, a <= x]
    larger = [b | b <- xs, b > x]
```