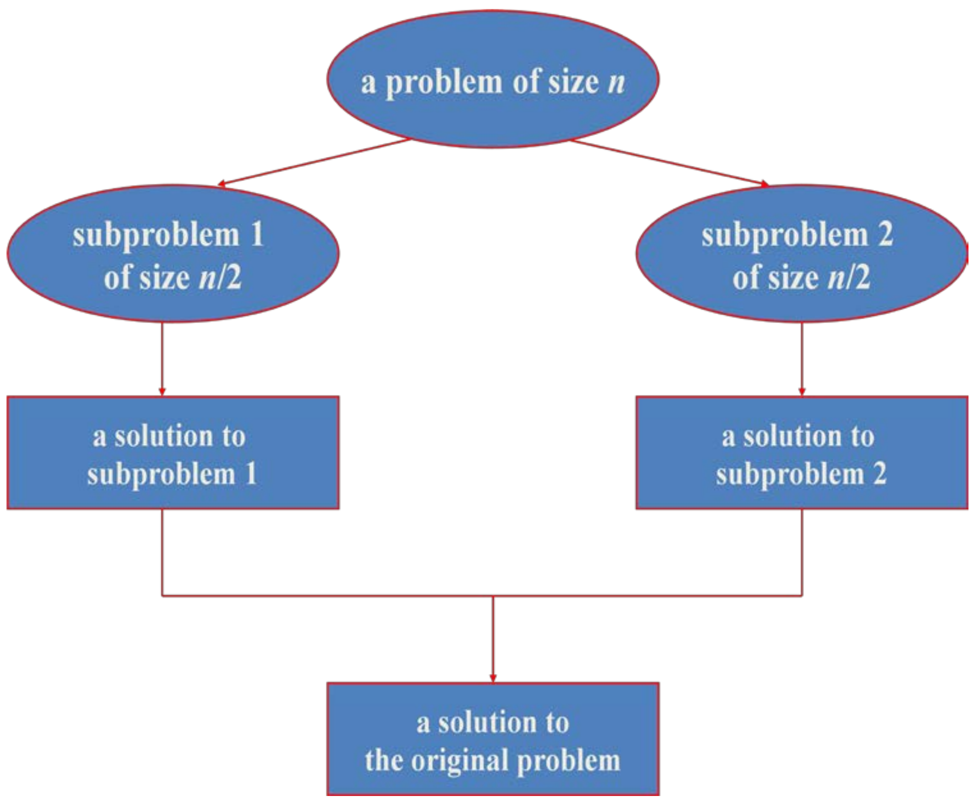
DIVIDE-AND-CONQUER

Approach

1. Divide instance of problem into two or more smaller instances
2. Solve smaller instances recursively
3. Obtain solution to original (larger) instance by combining these solutions

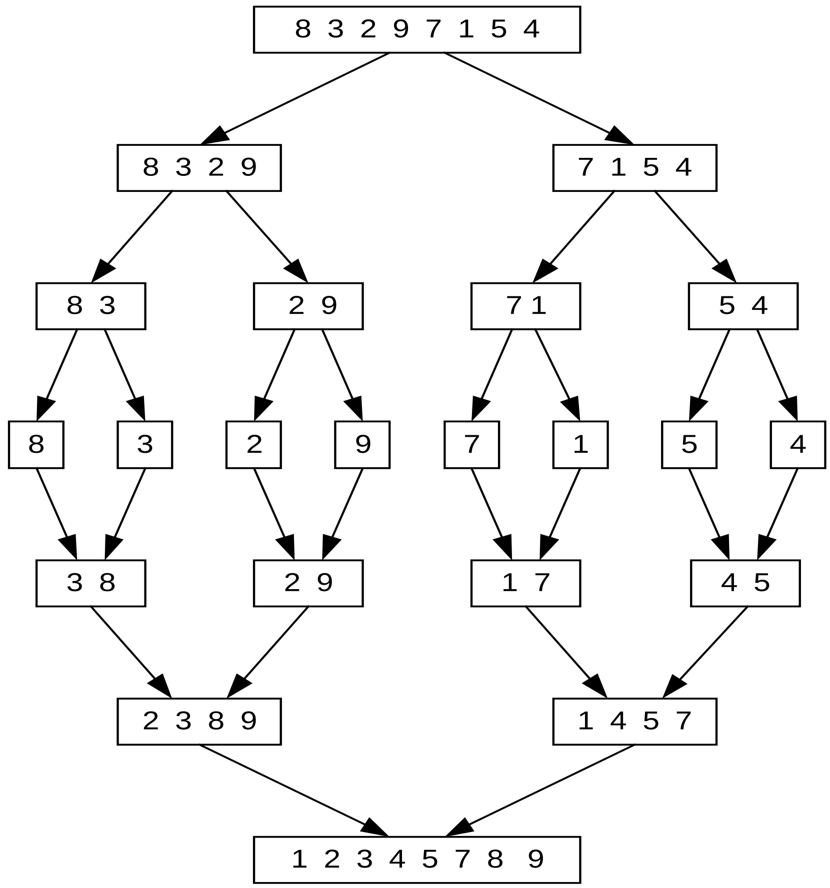


EXAMPLES

- Sorting: mergesort and quicksort
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen's algorithm
- Closest-pair and convex-hull algorithms

MERGESORT

Example



Algorithm

- Split array A[0..n-1] in two about equal halves and make copies of each half in arrays B and C
- Sort arrays B and C recursively
- Merge sorted arrays B and C into array A as follows:
  - Repeat the following until no elements remain in one of the arrays:
    - compare the first elements in the remaining unprocessed portions of the arrays
    - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
  - Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

Mergesort complexity

- Let size $n = 2^k$, basic operation = comparison
- $C(n)$ = cost of sorting n elements
- Recurrence:

  k=0: $C(1) = 0$ k=1: $C(2) = 1$

  $C(n) = 2C(n/2) + CostMerge(n)$

  - $CostMerge_{best}(n) = n/2$
  - $CostMerge_{worst}(n) = n-1$

  $C_{best}(n) = 2\,C_{best}(n/2) + n/2$

  $C_{worst}(n) = 2\,C_{worst}(n/2) + n - 1$

Best Case

$$\begin{aligned} C(n) = C(2^k) &= 2C(2^{k-1}) + 2^{k-1} \\ &= 2\,[2C(2^{k-2}) + 2^{k-2}] + 2^{k-1} \\ &= 2^2C(2^{k-2}) + 2^{k-1} + 2^{k-1} \\ &= 2^2\,[2C(2^{k-3}) + 2^{k-3}] + 2^{k-1} + 2^{k-1} \\ &= 2^3C(2^{k-3}) + 2^2.\,2^{k-3} + 2^{k-1} + 2^{k-1} \\ &= 2^3C(2^{k-3}) + 3.\,2^{k-1} \\ = \ldots &= 2^kC(2^{k-k}) + k.\,2^{k-1} \\ &= k.\,2^{k-1} = (n/2)\log_2 n \in \Theta(n \log_2 n) \end{aligned}$$

Worst Case

$$\begin{aligned} C(n) = C(2^k) &= 2C(2^{k-1}) + 2^k - 1 \\ &= 2\,[2C(2^{k-2}) + 2^{k-1} - 1] + 2^k - 1 \\ &= 2^2C(2^{k-2}) + 2.\,2^{k-1} - 2 + 2^k - 1 \\ &= 2^2C(2^{k-2}) + 2^k + 2^k - 2 - 1 \\ &= 2^2\,[2C(2^{k-3}) + 2^{k-2} - 1] + 2^k + 2^k - 2^1 - 2^0 \\ &= 2^3C(2^{k-3}) + 2^2.\,2^{k-2} - 2^2 + 2^k + 2^k - 2^1 - 2^0 \\ &= 2^3C(2^{k-3}) + 2^k + 2^k + 2^k - 2^2 - 2^1 - 2^0 \\ &= 2^3C(2^{k-3}) + 3.\,2^k - \textstyle\sum_{i=0}^{3-1} 2^i \\ = \ldots &= 2^k\,C(2^{k-k}) + k.\,2^k - \textstyle\sum_{i=0}^{k-1} 2^i \\ &= k.\,2^k - (2^k - 1) = (k-1).\,2^k + 1 = n\log_2 n - n + 1 \in \Theta(n \log_2 n) \end{aligned}$$

Generally

- Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting: $\lceil \log 2\ n! \rceil \approx n \log 2\ n - 1.44n$
- Space requirement: $\Theta(n)$ (not in-place)
- Can be implemented without recursion (bottom-up) (i.e. 2 by 2, then 4 by 4, then 8 by 8, etc.)

GENERAL DIVIDE AND CONQUER RECURRENCE

General Recurrence

Divide n into b equal parts and solve a of them

$T(n) = aT(n/b) + f(n)$ where $f(n) \in \Theta(n^d)$, $d \geq 0$

f(n) = cost of dividing n into b instances of size n/b and combining their solutions

Master Theorem

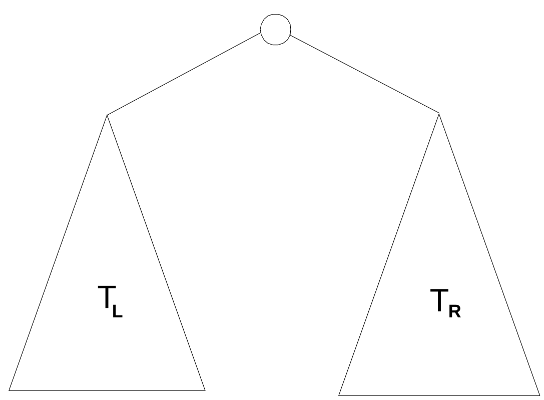
If $a < b^d$, $T(n) \in \Theta(n^d)$
If $a = b^d$, $T(n) \in \Theta(n^d \log n)$
If $a > b^d$, $T(n) \in \Theta(n^{\log_b a})$

Applying Master Theorem to Mergesort

- $C_{best}(n) = 2\, C_{best}(n/2) + n/2$
- $C_{worst}(n) = 2\, C_{worst}(n/2) + n - 1$
- a=2, b=2, d=1, $a = b^d$, $C(n) \in \Theta(n^d \log n) = \Theta(n \log n)$

BINARY TREE ALGORITHMS

Traversal:
Algorithm Inorder(T)
if $T \neq \varnothing$
  Inorder(T_{left})
  print(root of T)
  Inorder(T_{right})



Height h(T):
$h(\varnothing) = -1$
$h(T) = \max\{h(T_L), h(T_R)\} + 1$ if $T \neq \varnothing$

Applying Master Theorem to Binary Tree Algorithms

$T(n) = 2\, T(n/2) + 1$
a=2, b=2, d=0, $a > b^d$, $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 2}) = \Theta(n)$

MULTIPLICATION OF LARGE INTEGERS

Brute Force

Consider the problem of multiplying two (large) *n*-digit integers represented by arrays of their digits such as:

A = 12345678901357986429   B = 87654321284820912836

The grade-school (brute-force) algorithm:

$$
\begin{array}{r}
a_1\ a_2 \ldots\ a_n \\
b_1\ b_2 \ldots\ b_n \\
(d_{10})\ d_{11} d_{12} \ldots\ d_{1n} \\
(d_{20})\ d_{21} d_{22} \ldots\ d_{2n} \qquad \\
\ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots\ \ldots \\
(d_{n0})\ d_{n1} d_{n2} \ldots\ d_{nn} \qquad\qquad
\end{array}
$$

Efficiency: n^2 one-digit multiplications

Divide and Conquer

A small example:
$A * B$ where A = 2135 and B = 4014
$A = (21 \cdot 10^2 + 35)$, $B = (40 \cdot 10^2 + 14)$

So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$
$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$

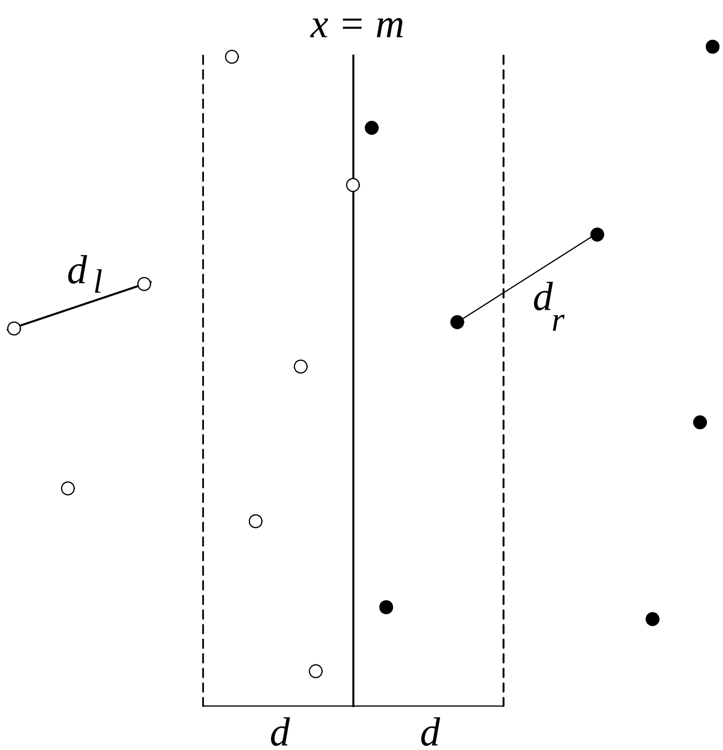
In general, if $A = A_1A_2$ and $B = B_1B_2$ (where A and B are *n*-digit, A_1, A_2, B_1, B_2 are $n/2$-digit numbers),

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

Master Theorem
Recurrence for the number of one-digit multiplications $M(n)$:
$M(n) = 4M(n/2)$, $M(1) = 1$
a=4, b=2, d=0, $a > b^d$, $T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

CLOSEST PAIR

- Step 1 Divide the points given into two subsets P_l and P_r by a vertical line $x = m$ so that half the points lie to the left or on the line and half the points lie to the right or on the line. (m= median of all the x coordinates)



- Step 2 Find recursively the closest pairs d_l, d_r for the left and right subsets.
- Step 3 Set $d = \min\{d_l, d_r\}$

We can now limit our attention to the points in the symmetric vertical strip S of width $2d$ as possible closest pair. (The points are stored and processed in increasing order of their y coordinates.)

- Step 4 Scan the points in the vertical strip S from the lowest up.

  For every point $p(x,y)$ in the strip, inspect points in the strip that may be closer to p than d. It has been proven that

  There can be no more than 5 such points following p on the strip list!

Master Theorem
$T(n) = 2T(n/2) + M(n)$, where $M(n) \in O(n)$
$a = 2$, $b = 2$, $d = 1$, $a = b^d$, $T(n) \in O(n \log n)$