Divide-and-conquer

Solves a problem by:
1. Breaking it into subproblems
2. Recursively solving these subproblems
3. Appropriately combining their answers

Divide-and-Conquer Multiplication, 1

\[(x_L \cdot 2^{n/2} + x_R) \times (y_L \cdot 2^{n/2} + y_R) = (x_L \times y_L) \cdot 2^n + (x_L \times y_R + x_R \times y_L) \cdot 2^{n/2} + (x_R \times y_R)\]

\[= (x_L \times y_L) \cdot 2^n + (Q - x_L \times y_R - x_R \times y_R) \cdot 2^{n/2} + (x_R \times y_R)\]

where \(Q = (x_L + x_R) \times (y_L + y_R)\)

Multiplying two \(n\)-bit numbers can be broken down into:
- multiplying two pairs of \(\frac{n}{2}\)-bit numbers, \(x_L \times y_L\) and \(x_R \times y_R\), and
- multiplying a pair of \(\frac{2n}{2} + 1\)-bit numbers, \((x_L + x_R) \times (y_L + y_R)\)

Divide-and-Conquer Multiplication, 2

\[\text{split}(z, n) = (z_L, z_R)\]

where
\[z_L = \text{the } \lceil \frac{n}{2} \rceil\text{-leftmost bits of } z\]
\[z_R = \text{the } \lfloor \frac{n}{2} \rfloor\text{-leftmost bits of } z\]

function multiply\((x, y)\)
// Input: \(x, y \in \mathbb{N}^+\) in binary  Output: Their product
\[n = \max(\text{size of } x, \text{size of } y)\]
if \(n = 1\) then return \(x \times y\)
\((x_L, x_R) \leftarrow \text{split}(x, n); \quad (y_L, y_R) \leftarrow \text{split}(y, n)\)
\(P_1 \leftarrow \text{multiply}(x_L, y_L); \quad P_2 \leftarrow \text{multiply}(x_R, y_R); \quad P_3 \leftarrow \text{multiply}(x_L + x_R, y_L + y_R)\)
return \(P_1 \times 2^n + (P_3 - P_1 - P_2) \times 2^{n/2} + P_2\)

- If \(T(n)\) is the time required for multiplying two \(n\)-bit numbers, then \(T(n) \leq 2 \cdot T(n/2) + T(n/2 + 1) + c \cdot n\) for \(n > 1\) and \(T(1) = c'\).

a recurrence inequality
The Master Theorem, 1

Theorem

Suppose \( T(n) = a \cdot T(n/b) + O(n^d) \) where \( a, b \geq 1 \) and \( d \geq 0 \). Then:

- i) \( T(n) \in O(n^d) \), if \( d > \log_b a \).
- ii) \( T(n) \in O(n^{d \log_b a}) \), if \( d = \log_b a \).
- iii) \( T(n) \in O(n^{\log_b a}) \), if \( d < \log_b a \).

Case i. The work at the top dominates.

Case ii. Each level is roughly the same.

Case iii. The work at the leaves dominates.

The Master Theorem does not apply to:

- \( T(n) = n^a T(n/b) + n^n \).
- \( T(n) = 2^n T(n/2) + n^2 \).

Theorem

The Master Theorem, 2

Suppose \( T(n) = a \cdot T(n/b) + O(n^d) \) where \( a, b \geq 1 \) and \( d \geq 0 \). Then:

- i) \( T(n) \in O(n^d) \), if \( d > \log_b a \).
- ii) \( T(n) \in O(n^{d \log_b a}) \), if \( d = \log_b a \).
- iii) \( T(n) \in O(n^{\log_b a}) \), if \( d < \log_b a \).

- The theorem still holds when \( n/b \) is replaced by \( \left[ \frac{n}{b} \right] \pm c, \left[ \frac{n}{b} \right] \pm c, \ldots \)
- So \( T(n) = 2T(n/2) + (n/2 + 1) + \Theta(n) \Rightarrow T(n) = 3T(n/2) + \Theta(n) \).

Theorem

Binary Search

function binarySearch(A[1..n], v, low, high)

// Input: A is sorted; 1 \leq low; high \leq n; and A[low] \leq v \leq A[high].
// Output: If v is in A, then returns an i with A[i] = v.
if low = high then return low
mid \leftarrow\left\lfloor (low + high)/2 \right\rfloor
if v \leq A[mid] then return binarySearch(A[1..n], v, low, mid)
else return binarySearch(A[1..n], v, mid + 1, high)

- \( T(n) = T(n/2) + O(1) \).
- Then \( a = 1, b = 2, \) and \( d = 0 = \log_2 1 \).
- So, case ii applies and \( T(n) \in O(n^0 \log n) = O(\log n) \).
Problem: Sort a list of numbers.

Idea: Split the list in two roughly equal sized lists, sort each half recursively, and merge the results.

```plaintext
function merge([x1, ..., xk], [y1, ..., yℓ])
    // Input: two input lists are sorted Output: the merge of two lists
    if k = 0 then return [y1, ..., yℓ]
    if ℓ = 0 then return [x1, ..., xk]
    if x1 < y1 then x1 ⊙ merge([x2, ..., xk], [y1, ..., yℓ])
    else y1 ⊙ merge([x1, ..., xk], [y2, ..., yℓ])
```

Claim: merge is \(O(k + ℓ)\) time.

DPV’s Iterative Mergesort

```plaintext
function iterative-mergesort([a1, ..., an])
    // Input: a list of integers to be sorted.
    Q ← an empty queue
    for i ← 1, ..., n do enqueue(Q, ai)
    while Q has more than 1 element do
        enqueue(Q, merge(dequeue(Q), dequeue(Q)))
    return dequeue(Q)
```

Medians

Definition

Suppose we have a set: \(\{ x_1 \leq x_2 \leq \cdots \leq x_n \}\).

The median of a finite multiset \(S\) is an element \(v\) splits the multiset into two equal-sized (±1) halves:

- one with elements \(\leq\) the median, and
- the other with elements \(\geq\) the median.

- The median is often a better estimator than the average. Bill Gates walks into a bar and raises the average salary by several billion dollars, but the median is not likely to move that much.
- How to compute: Sort the input, take the middle element. This takes \(\Theta(n \log n)\) time.
- Can we do better?
The Selection Problem

**Input:** A list of $n$-many numbers, $S$, and $k \in \{1, \ldots, n\}$.

**Output:** The $k$th smallest element of $S$.

- For $k = 1$, this gives the minimum.
- For $k = n$, this gives the maximum.
- For $k = \lfloor n/2 \rfloor$, this gives the median.

```plaintext
function selection(S, k) // returns $k$-th smallest elm of $S$
  Pick $v$ randomly from $S$.
  Split $S$ into three lists:
  $S_L \leftarrow$ the elements of $S$ that are $< v$
  $S_v \leftarrow$ the elements of $S$ that are $= v$
  $S_R \leftarrow$ the elements of $S$ that are $> v$
  if $k \leq |S_L|$ then return selection($S_L, k$)
  else if $|S_L| < k \leq |S_L| + |S_v|$ then return $v$
  else return selection($S_R, k - |S_L| - |S_v|$)
```

If we are lucky $|S_L| \approx |S_R| \approx |S|/2$ and we have the recurrence $T(n) = T(n/2) + O(n)$.

If we are unlucky, $S$ gets smaller by 1 each time. (How?)

What happens on average? We are close to lucky ... next time.