Chapter 2 Divide-and-conquer algorithms

Joe Song

September 10, 2019

Further reading: *The notes are based on Chapter 2 of Dasgupta, Papadimitriou and Vazirani. Algorithms. 2008. McGraw-Hill. New York.*

The divide-and-conquer strategy:

1. Divide: break one given problem to many small subproblems
2. Conquer: recursively solve each subproblem
3. Combine: merge solutions to subproblems to a final solution to the given problem

Dramatically reducing runtime—somehow not intuitive.

1 Multiplication

1.1 Multiplication of two complex numbers:

By definition

$$(a+bi)(c+di) = ac - bd + (ad+bc)i \quad \text{(Four multiplications)}$$

Carl Friedrich Gauss (1777-1855) used the following:

$$\begin{aligned}
&(a+bi)(c+di) && (1)\\
=&ac - bd + (ad+bc)i && (2)\\
=&ac - bd + [(a+b)(c+d) - ac - bd]i && (3)
\end{aligned}$$

which contains three *unique* multiplications:

- ac
- bd
- $(a+b)(c+d)$

Why interesting?

If numbers are n digits,

- multiplication takes $\Theta(n^2)$ operations
- addition takes only $\Theta(n)$ operations
- If $n = 1000$, definition needs $4 \times 1000^2 + 3 \times 1000 = 4,003,000$ operations
- Gauss' algorithm only needs $3 \times 1000^2 + 7 \times 1000 = 3,007,000$ operations

1.2 Multiplication of two binary numbers x and y:

We assume x and y are n bits binary numbers:

$$x = [x_L][x_R] = 2^{n/2}x_L + x_R$$

$$y = [y_L][y_R] = 2^{n/2}y_L + y_R$$

$$\begin{aligned} & xy && (4) \\ & = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) && (5) \\ & = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R && (6) \end{aligned}$$

Then it takes $O(n)$ to add the numbers together. Let $T(n)$ be the time to do xy.

There are four products which can be computed recursively. Thus:

$$T(n) = 4T(n/2) + O(n)$$

which is $O(n^2)$.

However, if we do

$$x_L y_R + x_R y_L = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$$

We will have three products to compute.

function multiply (x, y)

Input: n-bit positive integers x and y

Output: the product xy

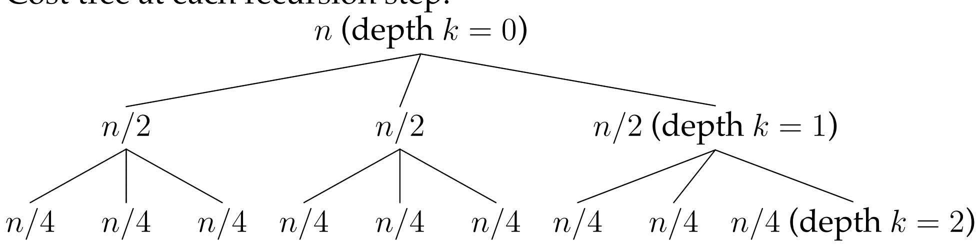
if $n == 1$: return xy
x_L, x_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of x
y_L, y_R = leftmost $\lceil n/2 \rceil$, rightmost $\lfloor n/2 \rfloor$ bits of y
P_1 = multiply(x_L, y_L)
P_2 = multiply(x_R, y_R)
P_3 = multiply$(x_L + x_R, y_L + y_R)$
return $P_1 \times 2^{2\lfloor n/2 \rfloor} + (P_3 - P_1 - P_2) \times 2^{\lfloor n/2 \rfloor} + P_2$

Then the running time becomes

$$T(n) = 3T(n/2) + O(n)$$

which leads to a running time of $O(n^{\log_2 3}) = O(n^{1.58})$.

Cost tree at each recursion step:



At depth k, the total time for all nodes in that layer is

$$3^k O(n/2^k)$$

Review – sum of geometric sequence at ratio r:

$$r^0 + r^1 + \ldots + r^k = \frac{1 - r^{k+1}}{1 - r} \quad (r \neq 1)$$

or

$$r^0 + r^1 + \ldots + r^k = k + 1 \quad (r = 1)$$

Total runtime:

$$\begin{aligned}
&\sum_{k=0}^{\log_2 n} 3^k O(n/2^k) && (7)\\
&= \frac{1 - 3^{1+\log_2 n} O(n/2^{1+\log_2 n})}{1-(3/2)} && (8)\\
&= \frac{(3/2)n^{\log_2 3} - 1}{1/2} && (9)\\
&= 3n^{\log_2 3} - 2 && (10)\\
&= O(n^{\log_2 3}) = O(n^{1.58}) && (11)
\end{aligned}$$

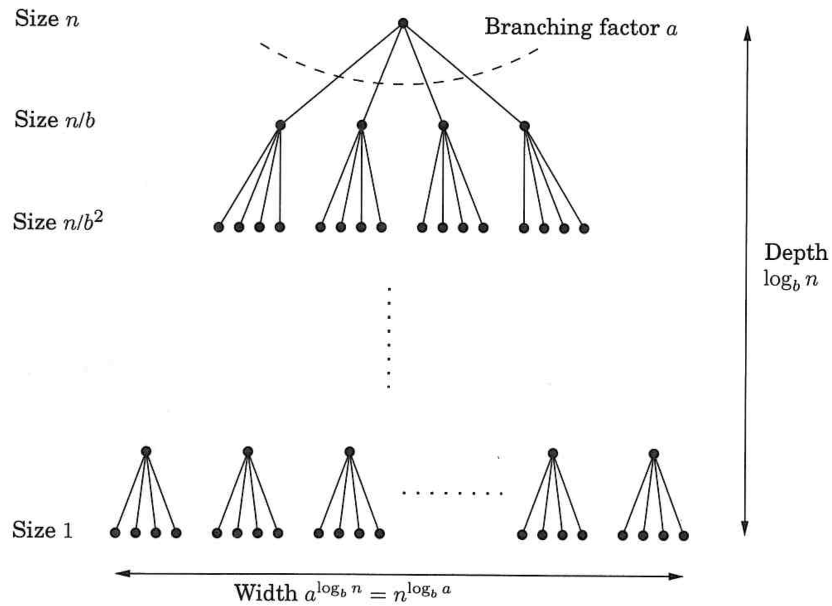
2 Recurrence relations

Theorem 2.1 (Master Theorem)**.** *If*

$$T(n) = aT(\lceil n/b \rceil) + O(n^d)$$

for some constant $a > 0$*,* $b > 1$*, and* $d \geq 0$*, then*

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = \log_b a \\ O(n^{\log_b a}) & \text{if } d < \log_b a \end{cases}$$



Proof.

Total of operations at level k (from 0 to $\log_b n$):

$$a^k \times O\left(\frac{n}{b^k}\right)^d = O(n^d) \times \left(\frac{a}{b^d}\right)^k$$

Thus the total $T(n)$ is

$$T(n) = \sum_{k=0}^{\log_b n} O(n^d) \times \left(\frac{a}{b^d}\right)^k \tag{12}$$

$$= O(n^d)\left(\frac{a}{b^d}\right)^0 + \ldots + O(n^d)\left(\frac{a}{b^d}\right)^{\log_b n} \tag{13}$$

$$= O(n^d)\frac{1-(\frac{a}{b^d})^{1+\log_b n}}{1-\frac{a}{b^d}} \quad \text{if } a \neq b^d \tag{14}$$

$$= O\left(\frac{n^d - \frac{a}{b^d}n^{\log_b a}}{1-\frac{a}{b^d}}\right) \quad \text{if } a \neq b^d \tag{15}$$

Can you show

$$n^d \left(\frac{a}{b^d}\right)^{\log_b n} = n^{\log_b a}$$

Three cases:

1. if ratio $\frac{a}{b^d} < 1$, $T(n)$ is dominated by the first term $O(n^d)$:

$$T(n) = O(n^d)$$

   The condition is equivalent to

$$\log_b a < d$$

2. if ratio $\frac{a}{b^d} = 1$, $T(n)$ contains $1 + \log_b n$ ($= O(\log n)$) terms all equal to $O(n^d)$:

$$T(n) = O(n^d \log n)$$

   The condition is equivalent to

$$\log_b a = d$$

   When using a tree approach, the height of the tree is $\log_b n$. The total cost for each level in the tree is exactly n^d. Therefore $T(n) = O(n^d \log n)$.

   The typical formula for geometric sequence sum is no longer valid when $r = 1$.

3. if ratio $\frac{a}{b^d} > 1$, $T(n)$ is dominated by the last term $O(n^{\log_b a})$:

$$T(n) = O(n^{\log_b a})$$

   The condition is equivalent to

$$\log_b a > d$$

$\square$

Key: decide which of $aT(\lceil n/b \rceil)$ and $O(n^d)$ dominates.

Examples.

$$\begin{aligned} T_1(n) &= 2T_1(n/2) + n^2 + n \\ T_2(n) &= T_2(n/3) \\ T_3(n) &= 10T_3(n/5) + \sqrt{n} \\ T_4(n) &= T_4(n/4) + 1 \end{aligned}$$

Here we assume $T_1(1) = T_2(1) = T_3(1) = T_4(1) = 1$. When possible, apply the Master theorem.

Solution:

$$\begin{aligned} \text{Case 1.} \quad & T_1(n) = O(n^2) \\ \text{Master theorem not applicable.} \quad & T_2(n) = O(1) \\ \text{Case 3.} \quad & T_3(n) = O(n^{\log_5 10}) \\ \text{Case 2.} \quad & T_4(n) = O(\log_4 n) = O(\log_2 n) \end{aligned}$$

3 Merge sort

Example:

$$10, 2, 5, 3, 7, 13, 1, 6$$

---

function mergesort($a[1 \dots n]$)
Input: An array of numbers $a[1 \dots n]$
Output: A sorted version of this array

if $n > 1$:
  return merge(mergesort($a[1 \dots \lfloor n/2 \rfloor]$), mergesort($a[\lfloor n/2 \rfloor + 1 \dots n]$))
else:
  return a

---

function merge($x[1 \dots k], y[1 \dots l]$)
if $k == 0$: return $y[1 \dots l]$
if $l == 0$: return $x[1 \dots k]$
if $x[1] < y[1]$:
  return $x[1] \circ$ merge($x[2 \dots k], y[1 \dots l]$)
else:
  return $y[1] \circ$ merge($x[1 \dots k], y[2 \dots l]$)

Running time for merge:

$$S(k + l) = S(k + l - 1) + 1 = S(k + l - 2) + 2 = \cdots = S(1) + k + l - 1 = O(k + l)$$

Running time for mergesort:

$$T(n) = 2T(n/2) + O(n)$$

By Master theorem case 2,

$$T(n) = O(n \log n)$$

With the help of a queue, merge-sort can be done iteratively:

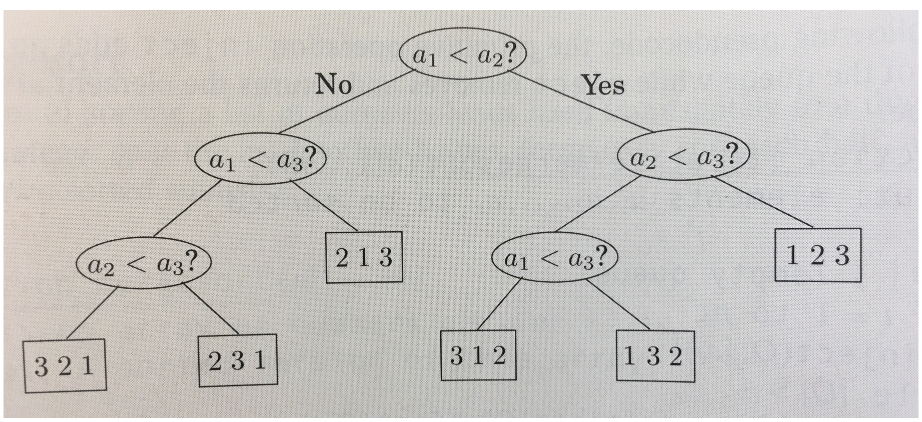
- `inject` adds an element (which can be an array) to the end of the queue;
- `eject` removes and returns the front element of the queue.

---

function iterative-mergesort($a[1 \ldots n]$)
Input: elements $a_1, a_2, \ldots, a_n$ to be sorted

$Q = []$ (empty queue)
for $i = 1$ to n:
    inject(Q, $[a_i]$)
while $|Q| > 1$:
    inject(Q, merge(eject(Q), eject(Q))) (inject the merged array into Q as a single element)
return eject(Q)

3.1 An $n \log n$ lower bound for sorting



Arguments:

1. The depth of comparison tree is the number of comparisons
2. The tree must have $n!$ leaves to accommodate all possible comparisons
3. A binary tree of depth d will have at most 2^d leaves
4. Therefore $2^d \geq n!$, which gives rise to $d \geq \log n!$
5. $\log n! \geq c \cdot n \log n$. Let n be an even number.

$$\begin{aligned}
n! &= n(n-1)\cdots(n/2+1)(n/2)\cdots 5\cdot 4\cdot 3\cdot 2\cdot 1 && (16)\\
&= [n(n-1)\cdots(n/2+1)]\,[(n/2)\cdots 5\cdot 2\cdot 3\cdot 2\cdot 2] \quad (n\geq 8) && (17)\\
&> \left[\underbrace{\frac{n}{2}\cdot\frac{n}{2}\cdots\frac{n}{2}}_{n/2 \text{ terms}}\right]\left[\underbrace{2\cdots 2\cdot 2\cdot 2\cdot 2\cdot 2}_{n/2 \text{ terms}}\right] && (18)\\
&= n^{\frac{n}{2}} && (19)
\end{aligned}$$

Thus, we have when n is even and $n \geq 8$

$$\log n! > \frac{1}{2} n \log n$$

Homework: argue that $\log n! > \frac{1}{2} n \log n$ is mathematically true when n is a large enough odd number.

4 Median

Median: the 50-th percentile of a list of n numbers—an equal number of numbers are bigger and smaller than the median.

E.g., 45, 1, 10, 30, 25. The median is 25.

When n is odd, median is the middle number after input is sorted;

When n is even, two numbers are in the middle of the sorted input. We take the smaller one as median, also known as lower median.

More robust than the mean

4.1 Selection

Selection(S, k):

- Input: A list of numbers S; an integer k
- Output: The kth smallest element of S

Solve by divide-and-conquer:

1. Randomly select a number v from S
2. S_L is a set with all numbers less than v
3. S_v is a set with all numbers equal to v
4. S_R is a set with all numbers greater than v
5. Recursively perform

$$\text{selection}(S, k) = \begin{cases} \text{selection}(S_L, k) & \text{if } k \leq |S_L| \\ \text{v} & \text{if } |S_L| < k \leq |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v| \end{cases}$$

Example:

$$S = 2, 36, 5, 21, 8, 13, 11, 20, 5, 4, 1$$

$$S_L = 2, 4, 1 \qquad S_v = 5, 5 \qquad S_R = 36, 21, 8, 13, 11, 20$$

4.2 Efficiency

4.2.1 Good case

Assume that we pick a pivot randomly between 25th and 75th percentile

```
|---------- |<-----------*----------->|-----------|
0          25th          50th          75th      100th
```

On average

$$T(n) \leq T(3n/4) + O(n)$$

This will lead to $O(n)$ running time by the Master's Theorem.

4.2.2 Worst case

The worst case happens when either S_L or S_R is empty and the pivot is not the k-th smallest element.

$$T(n) \leq T(n-1) + O(n)$$

This will lead to $O(n^2)$ running time.