Chapter 2: Divide and Conquer

Anna Brandenberger, Anton Malakhveitchouk

January 26, 2018

This is the second chapter of the augmented transcript of a lecture given by Luc Devroye on the 11th of January 2018 for the Honours Data Structures and Algorithms class (COMP 252, McGill University). The subject was the Divide-and-Conquer algorithm design method.

Principle

The principle of the divide-and-conquer method is to solve a large problem by breaking it down into several sub-problems, recursively solving them, and finally merging the solutions to these sub-problems to give the final solution.

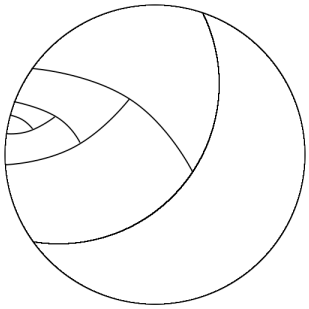

Figure 1: Diagram of the idea behind the divide-and-conquer algorithm principle.

Sample Problems

Example 1. Chip Testing Problem[1]

[1] Cormen et al. [1989] exercise 4-5.

- **Context**: A factory produces chips that are either good (G) or bad (B). To evaluate their state, the chips can be paired in a testing device where the two chips evaluate each other, i.e., each chip gives its opinion on whether the other is good or bad. Good chips tell the truth, while bad chips are unreliable.
- **Goal**: Given n chips, determine their state (G/B), with the premise that the set of good chips $\mathscr{G}$ is larger than the set of bad chips $\mathscr{B}$: $|\mathscr{G}| > |\mathscr{B}|$ [2]. Complexity is measured in terms of the number of uses of the testing device, which acts as an oracle.

[2] This is a necessary condition. One can prove that if it is reversed, the algorithm could fail.

The tester has three possible outputs according to the opinions given by the chips: GG, BB or BG. Note the following:

GG } implies that both chips are from $\mathscr{G}$ or both are from $\mathscr{B}$.

BB
BG } implies that the pair contains at least one chip from $\mathscr{B}$.

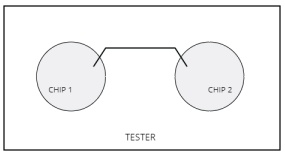

Figure 2: Diagram of the Tester, an oracle (type of device described in the first class on 01/09, a "black box" device capable of producing a solution to a given computational problem in one operation).

Algorithm Proposal:

1. Find one good chip using the Divide-and-Conquer method.
2. Do $n - 1$ additional tests to determine the state of every other chip.

THE DIVIDE-AND CONQUER algorithm for finding one good chip depends on whether n is even or odd. We will deal with the even case first, and then modify it slightly to yield the odd case.

EVEN CASE:

- Pair the chips and test: total $n/2$ pairs (tests).
- Eliminate all pairs that are not GG.
- For each such GG pair, eliminate the second chips. (This is the key divide-and-conquer step.)

This algorithm is applied recursively to the remaining chips, reducing down until there is one chip remaining. The $|\mathscr{G}| > |\mathscr{B}|$ condition is preserved throughout, therefore the last chip remaining must be a chip from $\mathscr{G}$.

There must be at least one GG pair to begin with, since $|\mathscr{G}| > |\mathscr{B}|$ (pidgeonhole principle). In the first elimination, each non-GG pair removed contains at least one chip from $\mathscr{B}$, so there are more (or equal number) bad than good chips eliminated, therefore the $|\mathscr{G}| > |\mathscr{B}|$ condition is preserved for the remaining GG pairs. Furthermore, the two sets of chips in the GG pairs are identical, therefore eliminating one of them still guarantees $|\mathscr{G}| > |\mathscr{B}|$.

ODD CASE:

- Take the n^{th} chip and perform $n-1$ tests against all the other chips. Take a majority vote of the $n-1$ chips. In case of a tie, declare the chip "good".
- If the n^{th} chip is good, halt!
- Otherwise, eliminate the n^{th} chip and apply the even case algorithm.

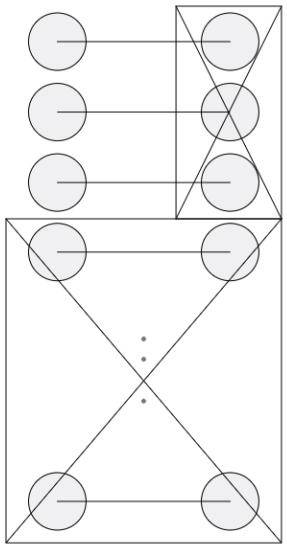

Figure 3: Illustration of the even case divide-and-conquer algorithm (part 1. to find one good chip).

REMARK ABOUT RANDOMIZATION:

It is easy to solve this problem with $O(n)$ expected time complexity, if we randomize. Just repeat the following until a success occurs:

- Choose a chip α at random from the remaining chips.
- Let the remaining chips vote on α. If the majority (50% or more) declare it "good", then it is good, and we can halt: α is in $\mathscr{G}$.

  One such round costs not more than n.

We halt after a round with probability at least 0.5. Therefore, we expect to execute not more than two rounds. The expected complexity is thus $\leq 2n$.

Time Complexity Analysis

Let T_n be the worst-case time taken for a problem of size n.

$$\begin{cases} T_n \leq n-1 + \frac{n-1}{2} + T_{\frac{n-1}{2}} & \text{, when } n \text{ is odd,} \\ T_n \leq 0 + \frac{n}{2} + T_{\frac{n}{2}} & \text{, when } n \text{ is even,} \end{cases} \tag{1}$$

where $T_1 = 0$, $T_2 = 0$.

LET n be a power of 2: $n = 2^k$. By substitution,

$$\begin{aligned} T_n &\leq \frac{n}{2} + T_{\frac{n}{2}} \\ &\leq \frac{n}{2} + \frac{n}{4} + T_{\frac{n}{4}} \\ &\leq \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + ... + 1 \\ &\leq n\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ...\right) + 1 \\ &= n + 1. \end{aligned} \tag{2}$$

It is a good exercise at this point to show that $T_n = O(n)$ in general, without the power-of-two restriction.

Example 2. Fractal/Karatsuba multiplication[3]

[3] Karatsuba and Ofman [1963]

- **Context**: algorithm for multiplying two n-bit numbers a_n and b_n:[4]

$$\begin{aligned} a_n &= ([0100001....][.....0100]) = \alpha_1 + \alpha_2 \times 2^{\frac{n}{2}} \\ b_n &= ([0101101....][.....0110]) = \beta_1 + \beta_2 \times 2^{\frac{n}{2}} \end{aligned} \tag{3}$$

  where α_1 and β_1 are $n/2$ bit numbers.

[4] We assume a_n and b_n are the same size. If not, add zeros in front of the smaller number. By padding with a front 0 if necessary, we can assume that n is even.

NAIVE ALGORITHM:

Compute $\alpha_1\beta_1+\alpha_2\beta_2+(\alpha_1\beta_2+\alpha_2\beta_1)\times 2^{n/2}$ by performing multiplication recursively, noting that α_i and β_i are $n/2$-bit numbers.

Time Complexity Analysis

Let T_n be the cost of multiplication in bits:

$$T_n = 4T_{\frac{n}{2}} + 2n + 6n. \tag{4}$$

$2n$ comes from the shifting (multiplying by 2^n and $2^{\frac{n}{2}}$) and $6n$ comes from the three additions. What costs us are the four recursions.

This yeilds $T_n = \Theta(n^2)$ — This is of the same order as the original simple multiplication algorithm.

KARATSUBA'S INNOVATION was realizing that:

$\alpha_1\beta_2 + \alpha_2\beta_1 = (\alpha_1 - \alpha_2)(\beta_1 - \beta_2) + \alpha_1\beta_1 + \alpha_2\beta_2$.

APPLYING THIS trick, the algorithm now only needs to compute $\alpha_1\beta_1$, $\alpha_2\beta_2$ and $(\alpha_1 - \alpha_2)(\beta_1 - \beta_2)$: three multiplications instead of four. [Note that both $\alpha_1 - \alpha_2$ and $\beta_1 - \beta_2$ are $n/2$-bit numbers]. We have:

$$T_n = 3T_{\frac{n}{2}} + 2n + 10n, \tag{5}$$

$2n$ comes from the shifting and $10n$ from the additions and subtractions.

and, as we will see later, this yields $T_n = \Theta(n^{\log_2 3})$.

FURTHER DEVELOPMENTS, such as the Toom-Cook algorithm[5], are even faster: the Toom-Cook algorithm, also known as Toom3, divides a_n and b_n into three parts instead of two, yielding:

[5] Cook and Aanderaa [1969]

$$T_n = 5T_{n/3} + n, \tag{6}$$

and thus, $T_n = \Theta(n^{\log_3 5})$, which is a slight improvement over Karatsuba's algorithm[6].

[6] As we have:

- $\log_2 3 \approx 1.58$
- $\log_3 5 \approx 1.46$

Example 3. Merge Sort

- **Goal:** sort a given list A of size n.

ALGORITHM PROPOSAL:

1. Sort $A[1, ..., \frac{n}{2}]$
2. Sort $A[\frac{n}{2} + 1, ..., n]$
3. Then merge the two sorted lists.

We measure complexity T_n in terms of the number of element comparisons (using a "comparison oracle"). Note that the merge costs not more than $n - 1$, and therefore,

$$T_n \leq n - 1 + T_{\lfloor \frac{n}{2} \rfloor} + T_{\lceil \frac{n}{2} \rceil} \approx n + 2T_{\frac{n}{2}}. \tag{7}$$

As we will see in the next chapter, this yields $T_n = \Theta(n \log n)$.

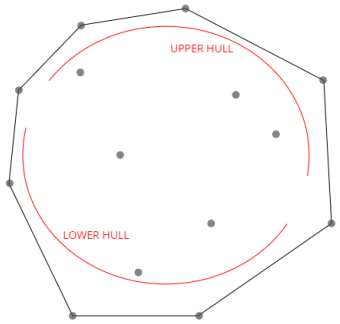

Figure 4: Diagram showing the convex hull, separated into the upper and lower hull, for a set of points in $\mathbb{R}^2$.

To visualize the convex hull, imagine stretching a rubber band around all the points and letting it snap in.

Example 4. Computational Geometry: Finding the Convex Hull

- **Goal**: Given n points in the plane $\mathbb{R}^2$, find the convex hull in order.

A point y is on the convex hull of a set of points if there exists a line through y that has all other points on one side. Let a and b be the points with the smallest and largest x-coordinate in the data. They are part of the convex hull. The line through a and b separates the convex hull into an upper convex hull (the points above the line) and a lower convex hull. We therefore need only develop an algorithm for the upper convex hull.

THE PRINCIPLE OF our algorithm is the same as MergeSort.

ALGORITHM PROPOSAL:

1. Find the upper hull for points $(1, ..., \frac{n}{2})$.
2. Find the upper hull for points $(\frac{n}{2} + 1, ..., n)$.
3. Merge by progressing by x-coordinate from left to right, eliminating points with angle $< 180°$. This step takes linear time as every step either adds one point to the upper convex hull or eliminates one.

   As in the case of MergeSort, $T_n = \Theta(n \log n)$.
   Exercise: Write the merging algorithm out in detail.

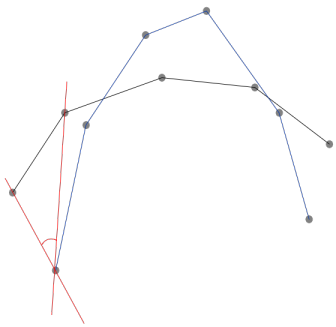

Figure 5: Diagram of the merging step. The blue and black lines are respectively the upper hulls of the $(1, ..., \frac{n}{2})$ and $(\frac{n}{2} + 1, ..., n)$ points from steps 1. and 2.

References

S.A. Cook and S.O. Aanderaa. On the minimum computation time of algorithms. *Transactions of the American Mathematical Society*, 142: 291–314, 1969.

T.H Cormen, C.E. Leiserson, R.L.Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 1989. ISBN 9780262033848.

A. Karatsuba and Yu. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics—Doklady*, 7(7):595–596, 1963. Translation of an article in *Doklady Akademii Nauk SSSR*, 145(2), 1962.