Algorithms

DATE: 21-04-2022

# DIVIDE-ET-IMPERA / DIVIDE-AND-CONQUER ALGORITHMS

$R_q$ $(q \geq 2)$ (RECURRENCES)

$$T_q(n) \leq \begin{cases} q \cdot T\left(\frac{n}{2}\right) + c \cdot n & \forall n \geq 3 \\ c & n \in \{0,1,2\} \end{cases}$$

(q: parameter)

THM: $T_2(n) \leq O(n \log n)$

THM: $T_q(n) \leq O(n^{\log_2 q})$

$\forall$ INTEGER $q \geq 3$ ← different behaviour than when $q < 3$

Try to prove this as an exercise: $(T_1(n) \leq O(n))$

Thus,

$T_4(n) \leq O(n^{\log_2 4}) = O(n^2)$

$T_3(n) \leq O(n^{\log_2 3}) = O(n^{1.5849\ldots})$ ← strange because we have been looking at combinatorial runtimes

$\log_2 3 = 1.5849$

---

## INTEGER ADDITION

$n = 4$

```
 1100 +
 0111 =
------
10011
```

Summing up two binary (or decimal) numbers of "n" digits each takes $O(n)$ time.

## INTEGER MULTIPLICATION

```
  1100 ·
  0111
------
  1100
 1100
1100
0000
-------
1010100
```

Multiplying two binary (or decimal) numbers of n digits each (using the primary school algo) takes $O(n^2)$ time.

# HOW TO IMPROVE MULTIPLICATION?

Let us assume that our goal is to multiply two n digits numbers, x and y.

We would like to compute $M = x \cdot y$

To implement a divide-et-impera approach we will split x and y into two halves each.

| | | |
|---|---|---|
| $X = 100110$ | $X_1 = 100$ | $X_0 = 110$ |
| $Y = 011010$ | $Y_1 = 011$ | $Y_0 = 010$ |

$x_1 \cdot 2^{n/2} = 100 \cdot 1000 = 100000$

$x_1 2^{n/2} + x_0 = 100000 + 110 = 100110$

Then, $x = x_1 \cdot 2^{n/2} + x_0$ and $Y = Y_1 \cdot 2^{n/2} + y_0$

$(a+b)(c+d)$

$ac + ad + bc + bd$

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

$$= x_1 y_1 2^{n/2} 2^{n/2} + x_1 y_0 2^{n/2} + x_0 y_1 2^{n/2} + x_0 y_0$$

$$= x_1 y_1 2^{n} + (x_1 y_0 + x_0 y_1) 2^{n/2} + x_0 y_0$$

# Multiplications by powers of 2 take linear time (n).

Thus, we have split the problem of multiplying two binary numbers of n-digits each into the problems of

1. Computing 4 products of binary numbers of $n/2$ digits each;
2. Performing 3 sums of n-digits binary numbers;
3. Performing 2 multiplications with powers-of-2

Now, 2 and 3 can be easily performed in $O(n)$ time. What about 1 ?

$$T_4(n) \leq 4T\left(\frac{n}{2}\right) + cn$$

If $q = 4$, we have that $T_4(n) \leq O(n^{\log_2 4}) = O(n^2)$

We didn't gain anything so far! =

If we want to do better than $n^2$, we need to reduce the number of subproblems to $q \leq 3$.

We aim to compute:

$$M = X_1 Y_1 \, 2^n + (X_1 Y_0 + X_0 Y_1) 2^{n/2} + X_0 Y_0$$

3 subproblems of size $\frac{n}{2}$ digits each

$$M_2 = (X_1 + X_0)(Y_1 + Y_0) = X_1 Y_1 + X_1 Y_0 + X_0 Y_1 + X_0 Y_0$$

$$M_1 = X_1 Y_1$$

$$M_0 = X_0 Y_0$$

we are performing ONE SINGLE multiplication. We can see it gives 4 mult. later but it's still a single one.

$$M_2 - M_1 - M_0 = (X_1 + X_0)(Y_1 + Y_0) - X_1 Y_1 - X_0 Y_0 =$$

$$= \cancel{X_1 Y_1} + X_1 Y_0 + X_0 Y_1 + \cancel{X_0 Y_0} - \cancel{X_1 Y_1} - \cancel{X_0 Y_0} =$$

$$= X_1 Y_0 + X_0 Y_1$$

$$M = X_1 Y_1 \, 2^n + (X_1 Y_0 + X_0 Y_1) 2^{n/2} + X_0 Y_0 = M_1 \, 2^n + (M_2 - M_1 - M_0) 2^{n/2} + M_0$$

$X_1 Y_1 \cdot 2^n$: 1 product here; $(M_2 - M_1 - M_0) 2^{n/2}$: 1 product here; $X_0 Y_0$: 1 product here

Thus, we can get the product of two n-digits numbers by taking 3 products of two $\frac{n}{2}$-digits numbers.

With $q = 3$, I get:

$$T_3(n) \leq O(n^{\log_2 3}) = O(n^{1.5849\ldots})$$

This improves significantly the primary school algo.

This algo makes sense only if the numbers that we've to multiply are very large.

RECURSIVE-MULTIPLY $(x, y)$:

| | |
|---|---|
| LET $x = x_1 2^{n/2} + x_0$ and $Y = Y_1 2^{n/2} + Y_0$ | $O(n) \to \frac{n}{2} + \frac{n}{2}$ |
| COMPUTE $S_x = X_1 + X_0$ and $S_Y = Y_1 + Y_0$ | $O(n)$ |
| $M_2$ = RECURSIVE-MULTIPLY $(S_x, S_Y)$ | $T_3(\frac{n}{2})$ |
| $M_1$ = RECURSIVE-MULTIPLY $(X_1, Y_1)$ | $T_3(\frac{n}{2})$ |
| $M_0$ = RECURSIVE-MULTIPLY $(X_0, Y_0)$ | $T_3(\frac{n}{2})$ |
| RETURN $M_1 \, 2^n + (M_2 - M_1 - M_0) 2^{\frac{n}{2}} + M_0$ | $O(n)$ |

$$T_3(n) \leq 3T_3(\tfrac{n}{2}) + O(n) \implies T_3(n) \leq O(n^{\log_2 3})$$

As we did in MergeSort, we assume that the numbers that we multiply are a power of 2

000 11011

The best known algorithm for multiplying two n-digits numbers takes time $O(n \log n \cdot 2^{O(\log^* n)})$

$\log^* n$ = # of logs to take to make result $< 2$

$\log^* n \le 10$ IF $n \le 2^{2^{2^{2^{2^{2^{2^{2^{2^{2}}}}}}}}}$ (10 times)

$2^{50}$

$((2^2)^2)^2 = 2^{2^3} = 2^8 = 256$

$< 2^{273}$ atoms in the universe

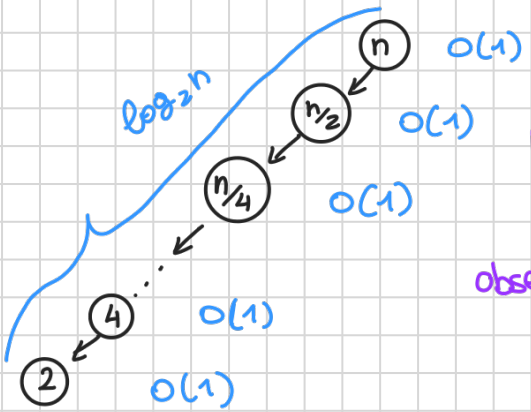
$\log^* 2^{273} < 5$

$\log^* 2^{50} < 6$

---

↙ Constant time for a single recursive call

$$S(n) \le \begin{cases} S(n/2) + c & \forall n \ge 3 \\ c & n \in \{0,1,2\} \end{cases}$$




Look at level by level and observe the runtime approach

TOTAL RUNTIME

$S(n) \le O(\log n)$

Guess the runtime and by induction prove that it holds approach

**THM:** $S(n) \le c \log_2 n, \quad \forall n \ge 2$

**P:** $S(0) = S(1) = S(2) = c$ (BASE CASE HOLDS)

$$S(n) \le S\left(\frac{n}{2}\right) + c \le c \log_2 \left(\frac{n}{2}\right) + c$$

$$= c(\log_2 n - \log_2 2) + c$$

$$= c(\log_2 n - 1) + c$$

More precision

$$= c \log_2 n - \cancel{c} + \cancel{c} = \boxed{c \log_2 n} \quad \blacksquare$$

We can use this recurrence on binary search

---

↙ Mountain – goes up and then down

**EX:** Let $x$ be an unimodal array of size $n$, that is, $\exists\, i \in \{0, 1, \dots, n-1\}$, such that $x[:i+1]$ is sorted increasingly, and $x[i:]$ is sorted decreasingly

$x = [2, 3, 10, 100, 94, 20, 3, 1]$

Find the largest value in $x$ as fast as you can.

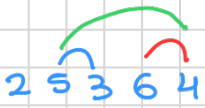
[2, 3, 100, 90, 80, 70, 3]

EXAM EX.

Let V be a n-dimensional array.

We say that $0 \leq i \leq j \leq n-1$ form an inversion

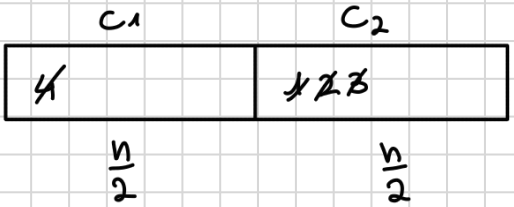
```
IF V[i] > V[j]
    c=0
    FOR i=0, ..., n-2        n-1 would also work
        FOR j=i+1, ..., n-1
            IF V[i] > V[j]:        } O(1)
                c+=1
RETURN c
```




1,2
1,4
3,4 $O(n^2)$

This solution would be ok, but by using MergeSort would be even better → $O(n \log n)$.

The other solution, based on MergeSort could make you get a 30 CUM LAUDE, and it looks like this:



Solve the following exercises.

exam_test

1. Let $G(V, E, w)$ be an undirected weighted graph, that is, let $V$ be the graph's set of nodes, let $E$ be the graph's set of edges, and let $w : E \to \mathbf{R}^+$ be the function that assigns positive weights to edges. (For instance, $V = \{1, 2, 3\}$, $E = \{\{1, 2\}, \{2, 3\}\}$, $w(\{1, 2\}) = 1.5$ and $w(\{2, 3\}) = 1$.)

   Consider the following claim: *"For any two nodes $v, v'$ of $V$, if $\pi$ is a shortest path in $G(V, E, w)$ from $v$ to $v'$, then $\pi$ is a shortest path from $v$ to $v'$ even in the graph $G(V, E, w')$, where $w'(e) = w(e)^2$ for each $e \in E$. (That is, the sequence of nodes of a shortest path does not change, if we square the weight of the edges.)"*

   Your task is to determine whether the claim is true or false. I.e., either prove the claim, or give a counterexample.

2. Consider a long rectilinear road, of length $t$ km, on the west-to-east axis. There are $k$ houses along this road, the first of which is at distance $d_1$ km from the western endpoint $w$ of the road, the second of which at distance $d_2$ from $w$, ..., and the last of which is at distance $d_k$ from $w$. A phone company would like to build antennas along the road so that each house can be served by its GSM network. If each antenna covers a radius of 10 km, what is the smallest number of antennas required for covering each house? Give an efficient algorithm for this problem (prove that it solves the problem, and bound its running time).