

+ +

Two applications in Computer Science

Sorting. Assume we are given n boxes, B_1, B_2, \dots, B_n . Box B_i contains a *distinct* number x_i which we are not allowed to see.

In each step, we are allowed to *compare* the contents of two boxes B_i, B_j of our choice i.e. determine whether $x_i < x_j$ or $x_i > x_j$.

How many comparisons do we have to do in order to be able to *sort* the boxes into increasing order of their contents?

More precisely let

$C(A, I)$ = number of comparisons of algorithm A on instance I

and

$$T(n) = \min_A \max_{I:|I|=n} C(A, I).$$

+ 1

+ +

A leaf v of the tree (a vertex with no descendants) is labelled by the order implied by the path from the root to v . Thus in the tree above, the leftmost leaf is labelled by 1,2,3 since the labels on the edges to the root are $x_1 < x_2$ and $x_2 < x_3$.

We can assume that for every path of the tree there is a (unique) ordering of the boxes which "satisfies" the labels of the path, otherwise we can remove some edges of the tree.

The *height* $h(T)$ of a tree T is the maximum number of edges in a path from the root to a leaf. The *depth* of a vertex v is the number of edges in the path from the root to v .

$$\max_I C(A, I) = h(T(A)). \quad (1)$$

We will prove the following

Lemma 1 A binary tree of height h has at most 2^h leaves.

+ 3

+ +

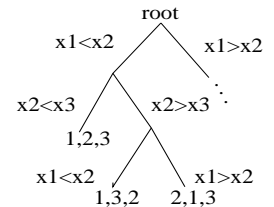
Theorem 1

$$T(n) = n \log_2 n + O(n)$$

i.e. there exist c_1, c_2 such that

$$n \log_2 n - c_1 n \leq T(n) \leq n \log_2 n + c_2 n.$$

Proof We represent each algorithm A by a *binary decision tree* $T(A)$:



In the above algorithm we first compare x_1, x_2 . If $x_1 < x_2$ we compare x_2, x_3 and so on.

+ 2

+ +

Now for any A , $T(A)$ has $n!$ leaves, one for each possible ordering of the boxes. Lemma 1 implies then that for all A ,

$$\begin{aligned} h(T(A)) &\geq \log_2(n!) \geq \log_2((n/e)^n) \\ &= n \log_2 n - O(n). \end{aligned}$$

Thus

$$T(n) \geq n \log_2 n - O(n).$$

+ 4

+

+

+

+

Merge Sort

Proof of Lemma 1 By induction on the height h . It is simple for $h = 1$ and so assume that any tree of height $h' < h$ has at most $2^{h'}$ leaves and let T be a binary tree of height h . Suppose it has k leaves.

Delete all the leaves of T of depth h and the edges incident with them. We obtain a tree T' of height $h - 1$ which has k_1 leaves which are also leaves of T and k_2 leaves which are immediate ancestors of leaves of T of height h . The result follows from

$$k_1 + k_2 \leq 2^{h-1} \text{ and } k \leq k_1 + 2k_2.$$

□

$MS(n)$

If $n \geq 2$;

Partition boxes into 2 sets of size $\lfloor n/2 \rfloor, \lceil n/2 \rceil$;

Apply $MS(\lfloor n/2 \rfloor)$ to first set to get sorted list L_1 ;

Apply $MS(\lceil n/2 \rceil)$ to second set to get sorted list L_2 ;

MERGE the 2 sorted lists:

Create sorted list L as follows:

Repeatedly delete the minimum x of $front(L_1), front(L_2)$ until L_1, L_2 are both empty;

Place x at the back of list L

At the end of this procedure, L contains the boxes in sorted order.

+

5

+

6

+

+

+

+

Let $C(n)$ be the maximum number of comparisons that $MS(n)$ uses to sort n elements. It is not easy to see what this is directly, instead we set up a recurrence.

$C(1) = 0$ and

$$C(n) \leq C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + n - 1. \quad (2)$$

Lemma 2

$$C(n) \leq n \log_2 n + n$$

Proof By induction on n . This clearly true for $n = 1$. Assume $n \geq 2$.

Case 1: $n = 2m$ is even.

Applying (2) we see that

$$\begin{aligned} C(n) &\leq 2(m \log_2 m + m) + n - 1 \\ &= n \log_2 n + n - 1. \end{aligned}$$

+

7

+

8

Case 2: $n = 2m + 1$ is odd. Applying (2) we see that

$$\begin{aligned} C(n) &\leq m \log_2 m + m + (m + 1) \log_2 (m + 1) \\ &\quad + m + 1 + n - 1 \\ &= n \log_2 n + n - \alpha_n. \end{aligned}$$

where

$$\begin{aligned} \alpha_n &= 1 - \frac{n-1}{2} \log_2 \left(1 - \frac{1}{n}\right) - \frac{n+1}{2} \log_2 \left(1 + \frac{1}{n}\right) \\ &= 1 - (\log_e 2)^{-1} \sum_{k=0}^{\infty} \frac{n^{-(2k+1)}}{(2k+1)(2k+2)} \\ &\geq 0. \end{aligned}$$

□

Fast multiplication. Assume, that we have to multiply two decimal numbers, both are of length n . If we multiply them digit by digit, then we have to make n^2 (digit-by-digit) multiplications. How can we reduce this number for large n ?

Assume that $n = 2^k$ for some integer k and that the two numbers are $a = 10^{n/2}a_1 + a_2$, and $b = 10^{n/2}b_1 + b_2$ where a_1, a_2, b_1, b_2 are all $n/2$ digit numbers. Then

$$\begin{aligned} ab &= 10^n a_1 b_1 + 10^{n/2}(a_1 b_2 + a_2 b_1) + a_2 b_2 \\ &= 10^n a_1 b_1 + \\ &\quad 10^{n/2}((a_2 + a_1)(b_1 + b_2) - a_1 b_1 - a_2 b_2) \\ &\quad + a_2 b_2 \end{aligned}$$

What has been gained?

Let $a_k = M(2^k)$ so that $a_0 = M(1) = 1$ and

$$a_k \leq 3a_{k-1} + 2^{k+2}.$$

Dividing by 3^k we get

$$\begin{aligned} \frac{a_k}{3^k} &\leq \frac{a_{k-1}}{3^{k-1}} + 4 \times \left(\frac{2}{3}\right)^k \\ \frac{a_{k-1}}{3^{k-1}} &\leq \frac{a_{k-2}}{3^{k-2}} + 4 \times \left(\frac{2}{3}\right)^{k-1} \\ &\vdots \\ \frac{a_1}{3} &\leq a_0 + 4 \end{aligned}$$

Let $M(n)$ denote the total number of 2 digit multiplications and additions needed if we carry out the computation as indicated. Then

$$M(n) \leq 3M(n/2) + 4n.$$

$3M(n/2)$ comes from the 3 multiplications, $a_1 b_1, a_2 b_2$ and $(a_2 + a_1)(b_1 + b_2)$.

The $4n$ comes from the remaining additions.

Multiplying by a power of 10 is ignored, and only involves $O(n)$ work.

So

$$\begin{aligned} \frac{a_k}{3^k} - 1 &\leq 4 \left(1 + \frac{2}{3} + \dots + \left(\frac{2}{3}\right)^k\right) \\ &< \frac{4}{1 - \frac{2}{3}} = 12. \end{aligned}$$

So

$$\begin{aligned} a_k &\leq 13 \times 3^k = 13 \times 3^{\log_2 n} \\ M(n) &\leq 13 \times n^{\log_2 3}. \end{aligned}$$

If n is not a power of 2 then we can pad it out with $< n$ zeros to make it one. Thus

$$M(n) = O(n^{\log_2 3}).$$