Combinatorial Algorithms

An algorithm is an effective method for solving a problem with a finite sequence of instructions.

Hilbert's Entscheidungsproblem: Posed in 1928, is there an algorithm which can decide if an arbitrary mathematical fact is true or false?

Note: The algorithm needn't provide justification, but mustn't make mistakes.

The Halting Problem

Suppose we had a formal definition of an algorithm. Then given some text T, can we decide if T is an algorithm?

Can we even decide if, given the input t, T halts (i.e., stops running)?

Define

$$\text{Halt}(T,t) = \begin{cases} \text{Yes} & \text{if } T \text{ halts when given the input } t, \\ \text{No} & \text{if } T \text{ goes into an infinite loop when given the input } t. \end{cases}$$

Suppose we can decide if T halts when given the input t, so we have an algorithm to compute Halt(T, t).

Now define

$$\text{Diag}(T) = \begin{cases} \text{returns "Yes" (and halts) if Halt}(T,T) \text{ is "No",} \\ \text{goes into an infinite loop if Halt}(T,T) \text{ is "Yes".} \end{cases}$$

But then, what does Diag(Diag) do?

If it returns "Yes" then Halt(Diag, Diag) is "No", but this means Diag(Diag) goes into an infinite loop!

If it goes into an infinite loop, then Halt(Diag, Diag) is "Yes", but that means that Diag(Diag) halts!

Moral: There are things that no computer can compute, just like there are theorems that no proof can prove.

Of course, when dealing with specific problems, we often know our algorithms will halt.

We now consider one of these specific problems.

Sorting algorithms

There are n children in a line, all of different heights.

We would like to sort them. What is the best way?

What is meant by "best"?

Let's suppose that we don't have a measuring stick, so tell if one child is taller than another we have to put them back-to-back, and this is a time consuming process.

So, we want to minimize the number of comparisons.

Bubblesort

Compare the first and second children, put the shorter one first.

Then compare the 2nd and 3rd.

Continue until we compare the $n-1$st and nth.

Example: Suppose we start with 41523.

1st & 2nd: 14523
2nd & 3rd: no change
3rd & 4th: 14253
4th & 5th: 14235

After the first "round", which consists of $n-1$ comparisons, we can be sure that the tallest child is last, but we can't be sure of anything else.

So, we need to repeat this for the $(n-1)$ shortest children, then the $(n-2)$ shortest, and so on.

$$\text{Total \# comparisons} = \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} i = \binom{n}{2}.$$

[Nice animation on Wiki page for "Comparison sort"]

A lower bound

How few comparisons could we get away with?

We have to compare every child at least once (or else we would have no information about that child), so

$$\text{Total \# comparisons} \geqslant \frac{n}{2}.$$

We can improve this by noting that the "comparison graph" must be connected, so

$$\text{Total \# comparisons} \geqslant n-1.$$

A better lower bound

Suppose we could sort all permutations of $[n]$ with m comparisons.

If we do m comparisons, we'll get at most 2^m different sets of results.

Every permutation must be sorted differently, so we must have

$$2^m \geqslant n!$$

Since $n! \approx \left(\frac{n}{e}\right)^n$, this shows that

$$m \gtrsim n \log n.$$

MergeSort

Split the permutation into two halves, as equally as possible.

Then sort those halves, to get lists

$$a_1 < a_2 < \cdots < a_k$$

and

$$b_1 < b_2 < \cdots < b_\ell.$$

Now <u>merge</u> these lists, which requires at most $n-1$ comparisons.

<u>Example</u>: Start with 421563.
Sort 421 and 563...
Then merge 124 and 356.

Let $M(n)$ denote the # of comparisons that MergeSort needs to sort a list with n elements. Then:

$$M(2k) = 2M(k) + (2k-1)$$
$$M(2k+1) = M(k+1) + M(k) + (2k).$$

Let's define a sequence $\{m_k\}$ by

$$m_k = M(2^k).$$

Then:

$$m_k = 2m_{k-1} + 2^k - 1$$

The generating function for $\{m_k\}$ is (check!)

$$\frac{1}{(1-2x)^2} + \frac{1}{1-x} - \frac{2}{1-2x},$$

So

$$m_k = (k-1)2^k + 1.$$

So if $n = 2^k$, then $k = \log_2 n$, and

$$M(n) = (\log_2 n - 1)n + 1.$$

This is therefore about the best sorting algorithm we could hope for.