Hello Prof. Cai,

After class today we discussed some of the limitations of the RAM model for measuring the complexity of various algorithms. As a mental exercise, I decided to see if I could construct some silly pathological example which 'cheats' at sorting by exploiting the fact that we can use arbitrarily long words within our machine as atomic operations to perform sorting in $O(n+\log\log M)$ time, where n is the number of elements to be sorted and M is the size of the maximum element. I assume a RAM machine with constant cost arithmetic operations, $+, -, *, /, \&, |$ and comparison. If we also permit a constant time logarithm, which I admit may be asking too much of our magical RAM machine, then we can drop the $\log\log M$. As an outline, here is the strategy I use:

1. Compute an upper bound on the number bits per element
2. Pack all of the inputs from result into a single giant word
3. Perform a parallel 0/1 sort on the subwords with a cost of $O(1)$ per iteration
4. Unpack and return result

Let $S = \{s_1, s_2, ...s_n\}$ be the collection of elements to be sorted and without loss of generality assume n is even. To find the upper bound on the bits per s_i, we first find the maximum, M, at a cost of $O(n)$. Next we binary search on M to locate some power of 2 greater than said number (unless of course our RAM model permits unit cost logarithms, in which case we simply take $\log M$ and move on). The cost of this operation is $O(\log\log M)$.

Next, we construct a packed number v, an odd/even mask m and a guard flag f:

$$v = \sum_{i=1}^{n} s_i * 2^{3bi}$$

$$m = \sum_{i=1}^{n/2} (2^{6bi+b} - 2^{6bi})$$

$$f = \sum_{i=1}^{n} 2^{3b(i+2)}$$

Now we come to the heart of the matter which is the 0/1 sorting operation. To do this, define the following temporary quantities:

$$x = (v * 2^{3b})\&m$$

$$y = v\&(m * 2^{3b})$$

$$r = (x - y)\&(((f + y - x)\&(m * 2^{2b})) * 2^b)$$

Basically what we are doing is splitting v into odd/even components x, y. In r, we set each packed element to $x - y$ if $x > y$ or 0 otherwise using a standard bit twiddling trick. The flag f prevents overflow from one number from bumping into another. As a result, we may now combine r, x, y to get the following new vector:

$$v' = x + r + (y + r)/2^{3^b}$$

As a result, this will swap pairs of elements in x with their counterpart in y if and only if $x > y$. Repeating this process on various shifts of v gives the 0/1 algorithm, so as a result we may sort the internal vector in at most $O(n)$ steps. Note that since our assumed RAM computer is magical, finding each of these elements takes constant time since the above sums do not depend on v and could be computed while constructing v. To unpack v, we simply reverse the above described process, and so our total algorithm runs in $O(n)$ time.

As a final thought, it might be possible to remove the $O(\log \log M)$ term with some fancy arithmetic. If we could pack the numbers into the integers mod M, then there might be a way to make it work, but I couldn't figure out how to perform step 3 under these restrictions. It also might be possible to exploit a Godel numbering, but I couldn't figure out how to do parallel operations in that setting. If I come up with a better bound I will let you know.