Random Access Machine

Unit-cost RAM is the main model used in this course. This is similar to C or any other standard imperative language.

- Any operation costs one unit.
- Indirect addressing available.

A variation is the log cost RAM where an operation on an integer $x$ costs $\log(1 + x)$, this is more accurate but more tedious.

Interesting model: Boolean circuit

Boolean circuits are bit oriented with input bits, $x_1, \ldots, x_n$ and operands

- $\wedge$ = "and"
- $\lor$ = "or"
- $\neg$ = "negation"

The size is measured as the number of logic gates.
Example: XOR

\[
\begin{array}{cccc}
    & x_1 & \rightarrow & x_2 \\
    \swarrow & & & \searrow \\
x_1 & & & x_2
\end{array}
\]

which has size 3.

**Turing machines**

Will not be used in this course.

**Interesting model: Comparison model**

A common claim is that it requires \( n \log(n) \) time to sort \( n \) integers. However, this is only true in the comparison model where the only way to get information about the input is asking “Is \( x_i \geq x_j \)?”

**Theorem 0.1** Sorting requires \( \log_2(n!) \) comparisons in this model where \( ! \) is the factorial function \( n! = n \times (n-1) \times (n-2) \times ... \times 1 \).

**Proof:** The output is an ordering \( l_1, l_2, l_3, ..., l_n \) (i.e., a permutation of \( n \)) where \( x_{l_1} \geq x_{l_2} \geq x_{l_3} \geq ... \geq x_{l_n} \).

There are \( n! \) possible answers and the correct answer is found by using information about the input. If we ask \( T \) questions “\( x_i \geq x_j \)?” we have \( 2^T \) possible answer sets and each answer set gives an output. Note that each output is possible!

The number of answer sets is \( \geq \) the number of outputs which means that \( 2^T \geq n! \) and \( T \geq \log_2(n!) \).

An approximation of \( n! \) by Stirling’s formula is \( n! \sim (\frac{n}{e})^n \). This gives us that \( \log_2(n!) \sim n \log_2 n - n \log_2 e + o(n) \).

**Sorting algorithms**

Quicksort is average case \( O(n \log n) \) but it gives the wrong leading constant. Mergesort and Heapsort, are sorting algorithms that gives the optimal constant one in front of \( n \log n \).
Sort $n$ numbers with 64 bits each.

How long does it take to sort these?

Proposed algorithm:

- Initialize $2^{64}$ counters to 0.
- for $i = 1$ to $n$ increase counter $C_{x_i}$.
- Read off answer.

Time $n + O(1)$. An objection to this is that in real life $n$ is between $2^{20}$ and $2^{45}$ as smaller $n$ are “easy” and larger $n$ we cannot even read the input. This algorithm is ”cheating” as the $O(1)$ is the dominating term.

Radix sort

Sorts $n$ numbers in range $0...n^k - 1$ in time $\sim kn$.

Bucket sort

Bucket sort is a similar algorithm.

- Make $n$ buckets, say $n = 2^{22}$.
- Put elements in bins given by the first $\log n$ bits (22 bit).
- Sort bins recursively, now with 42 bit numbers.

There will be 3 levels of recursion and the time in $n$ is $3n + \text{book-keeping}$.

Is sorting in reality $O(n)$ time?

Think about what is the best/worst combination of $n$ and $w$ (the number of bits in the numbers)? For $w \leq 3\log n$ sorting can, as discussed above, be done in linear time! On the other hand if $w$ is very large then each comparison may take a very long time. One model that strikes a reasonable balance is the following. We want to sort $n$ numbers each with $w$ bits and we allow simple machine operations of $w$-bit numbers at unit cost. The current world record in this model is an algorithm that sorts in $O(n \log \log n)$ time by A. Andersson, T. Hagerup, S. Nilsson and R. Raman. A link to this is available on the homepage. It currently is unknown whether it can be done in $O(n)$!

Circuit model sorting

A circuit model of $n$ $w$-bit numbers with $m = wn$ input bits. Can sorting be done in $O(m)$ size? This is also unknown.
What we did not have time for

Given $n$ random integers each $w$ bit, i.e. $x_i \in 0...2^w - 1$ randomly. It is easy to sort in $O(n)$ time. See bucket sort with $n$ buckets. Let $s_x$ be the log $n$ most significant bits in $x$ and simply place $x$ in $B_{s_x}$ and sort the buckets by almost any method (even a with a quadratic sorting algorithm for the buckets, this can be proved to run in expected time $O(n)$). The proof was skipped but a sketch is available in the course notes in Section 18.5.