Chapter 34 (I)
Computational Complexity

In *algorithmics*, we showed that a problem can be solved in $O(f(n))$ for some $f(n)$, which should be reduced as much as possible.

With *complexity theory*, we try to find a function $g(n)$ as large as possible for which we have to prove that *any* algorithm that can correctly solve *all* the instances of that problem must necessarily take time of $\Omega(g(n))$.

We call such a $g(n)$ a *lowbound on the complexity of that problem*. We are completely satisfied if we can show $f(n) \in \Theta(g(n))$. (?)

We now address a more difficult issue: Given a problem, we want to find out the minimum time needed to solve it, taking into account *all* the possible algorithms.
The game of 20 questions

**Question:** Let your friend pick up a positive number between 1 and one million. How many questions do we need to ask to figure out the number?

**A solution:** A million.

**A better one:** Ask first “Is the number between 1 and 500,000?” If your friend answers “yes”, you ask “Is the number between 1 and 250,000?” Otherwise, you ask, instead, “Is the number between 500,001 and 1,000,000?” etc.

As each question cuts the number by half, and $1,000,000 < 2^{20}$, this approach will find the number in *at most* 20 steps.

**Question:** Is there an even better solution?

**Answer:** No!

**Question:** Why not?
The complexity of this game

Let $S_i$ be the set of candidates for the number after the $i^{th}$ question has been asked and let $k_i$ be its size. For example, $S_0 = [1, 10^6] = \{x | 1 \leq x \leq 10^6\}$ and $k_0 = 10^6$.

Let $Q_i$ be the $i^{th}$ question, and let $Q_i(n)$ be the answer to that question for the number $n$. For example, if $Q_1$ is “Is the number larger than 20”, then $Q_1(15)$ is “no”, while $Q_1(25)$ is “yes”.

Let $Y_i$ be $\{n \in S_{i-1} | Q_i(n) = “yes” \}$ and let $N_i$ be $\{n \in S_{i-1} | Q_i(n) = “no” \}$. For example, we have that

$$Y_1 = \{n | n \in [21, 1,000,000]\}.$$  

and

$$N_1 = \{n | n \in [1, 20]\}.$$
The punch line

As for every number, the answer is either “yes” or “no”, but not both, we have that for all \( i \geq 1 \), \( Y_i \cup N_i = S_{i-1} \) and \( Y_i \cap N_i = \emptyset \), i.e., \( Y_i \) and \( N_i \) are disjoint. Therefore,

\[
k_{i-1} = |S_{i-1}| = |Y_i| + |N_i|.
\]

Now, either \( Y_i \) or \( N_i \) has to contain at least \( \lceil k_{i-1}/2 \rceil \) numbers (Cf. Next page). Moreover, when either \( Y_i \) or \( N_i \) is cut, it might always be the case that the smaller one is cut (Stay tuned... Page 9), i.e., \( k_i \geq \lceil k_{i-1}/2 \rceil \) for each \( i \geq 1 \). (Notice that for all \( n, n = \left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \).)

Thus, it is possible that \( k_{19} \geq 2 \) (Cf. Next page.), meaning we need to ask at least one more question, i.e., at least 20 questions in total.
Here are the details...

1. Assume that $Y_i < \lceil k_{i-1}/2 \rceil$ and $N_i < \lceil k_{i-1}/2 \rceil$, i.e., $Y_i \leq \lceil k_{i-1}/2 \rceil - 1$ and $N_i \leq \lceil k_{i-1}/2 \rceil - 1$. Thus,

$$k_{i-1} = |S_{i-1}| = Y_i + N_i \leq 2 \lceil k_{i-1}/2 \rceil - 1.$$  
If for some $n \geq 0, k_{i-1} = 2n$, we would get $2n \leq 2(n - 1)$, which could not be true. Otherwise, $k_{i-1} = 2n + 1$, then we would have $2n + 1 \leq 2n$, which could not be true, either.

Thus, by de Morgan’s law, either $Y_i \geq \lceil k_{i-1}/2 \rceil$ or $N_i \geq \lceil k_{i-1}/2 \rceil$.

For example, if $k_{i-1} = 7$, then either $|Y_i| \geq 4$, or $|N_i| \geq 4$.

2. When for $i \geq 1, k_i \geq \lceil k_{i-1}/2 \rceil$, we have the lower bounds for $k_i, i \in [1, 19]$, as follows: 500,000; 250,000; 125,000; 62,500; 31,250; 15,625; 7,813; 3,907; 1,954; 977; 489; 245; 123; 62; 31; 16; 8; 4; 2.

Thus, $k_{19} \geq 2$. 

5
The complexity of sorting

We have learned several sorting algorithms: *insertion sort*, *bubble sort* and *selection sort* take $O(n^2)$; *quicksort* takes $O(n \log(n))$ on average; finally, *heapsort* and *mergesort* take at worst $O(n \log(n))$. All of these algorithms are based on key comparison.

Since any such algorithm corresponds to a decision tree, which must contain at least $n!$ leaves. As we saw earlier, it must have height at least $\lceil \log(n!) \rceil$ and an average height of at least that much as well. As $\log(n!) \in \Theta(n \log(n))$, the complexity of key comparison based sorting is $\Omega(n \log(n))$.

Hence, such sorting algorithms as heapsort, mergesort, and quicksort are the best we could get, if we sort things out based on key comparison.
Adversary arguments

Decision tree is a tool of *information-theoretic arguments*, in the form that “to get this much information, you have to make at least this much effort.” For example, to find out the relative order of all the \( n \) element, you have to make at least \( n \log n \) comparisons.

Sometimes, *adversary arguments* are more useful.

The basic idea is that, for a given problem on an input, which is unspecified except for its size, whenever the algorithm checks the input and asks a question, an intensional *demon*, an adversary, answers in a way that will force the algorithm to work harder.
What’s happening?

Essentially, the demon tries to keep the algorithm uncertain of the correct answer as long as possible.

If the algorithm claims to know the answer prematurely, the demon must be able to construct an input that is consistent with all of its answers to the questions, and show that the correct solution to this input just constructed is different from the output of the algorithm.

Since this just constructed input could be the actual input, it shows that the algorithm has not done enough.

In other words, the problem at hand is more complicated, thus deserving more time.
The 20 question problem revisited

In the game of 20 questions, the demon doesn’t need to pick a number first, it can wait until a question has been asked and then pick up a number in the bigger range of the two, thus forcing to cut off the smaller range.

After asking 19 questions, as we saw earlier, at least two numbers remain, which are consistent with all the previous answers.

If the algorithm claims to know the number at that moment or earlier, the demon can always pick up a number from the leftover and argue that this number is what in his mind.

Therefore, we have to ask at least 20 questions, but not 19 or less.
Is this graph connected?

Consider an algorithm that tests for a given graph with \( n \) vertices, if it is connected. The only questions allowed are of the form that “Is there an edge between vertices \( i \) and \( j \)?”

There is an algorithm, via DFS search, to solve this problem in \( O(|V| + |E|) = O(|V|^2) \).

As there are only two verdicts (Yes, \( G \) is connected; or no, it is not.), any decision tree contains only two leaves.

Thus, following the information theoretic approach, the only conclusion is that the algorithm needs to ask at least one question, which is a lower bound, but obviously useless.
The demon is kicking in

Now, we use an adversary argument to prove that the complexity of this problem is indeed $O(n^2), |V| = n$, by constructing a graph along the way.

Given a graph $G(V, E)$, the demon divides $V$ into two subsets, $V_1$ and $V_2$, of sizes $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$, respectively. For example, if $|V| = 15$, then $|V_1| = 7$, and $|V_2| = 8$.

Whenever the algorithm asks a question, the demon answers “yes” if and only if both $i$ and $j$ fall into the same set. Thus, it would say “no” if $i$ and $j$ fall into different sets.
Now what?

If the algorithm claims $G$ is connected, the demon simply presents the graph where all the vertices of either $V_1$ and $V_2$ are connected, but none belonging to different sets is.

Such a graph is consistent with all the information the demon has provided to the algorithm, although it may contain more information than those provided to the algorithm.

On the other hand, this graph is not connected since there is no path between $V_1$ and $V_2$. 😞
The other side

On the other hand, if the algorithm claims the graph is disconnected, the demon simply adds into the above graph an additional edge to connect \( i \in V_1 \) and \( j \in V_2 \), which the algorithm has not asked about. Such a pair is guaranteed to exist because of the assumption that the algorithm has yet to check all such paths.

Again this graph is consistent with the information the demon has provided to the algorithm, but it is connected. 😞

Hence, the algorithm must ask about the presence of all those crossing edges, \( \Theta(n^2) \) of them, to correctly decide the connectivity of any graph.