# Chapter 1. Lower bounds

(Luc Devroye, January 12, 2019)

**Table of contents**



## §1.1. Lower bounds: introduction

These notes are intended for my undergraduate students at McGill University. All errors and typos are for the students at the University of Waterloo. And any confusion caused by my explanations is meant for the pseudo-scholars at the University of Toronto.

We are studying algorithms that obtain solutions through the use of an oracle, i.e., a device that takes a general input and outputs an answer from a finite set. A $k$-oracle has $k$ possible answers. Examples:

(i) THE BINARY COMPARISON ORACLE. Here the oracle takes two keys as input, $x$ and $y$, and replies "yes" if $x < y$ and "no" otherwise. It is the standard oracle used by computer scientists for problems involving searching and sorting.

(ii) THE $k!$-ORACLE FOR SORTING $k$ KEYS. Here the oracle takes $k$ keys as input, $x_1, \ldots, x_k$ and replies with a permutation $(\sigma_1, \ldots, \sigma_k)$ of $(1, \ldots, k)$ that sorts the input: $x_{\sigma_1} < x_{\sigma_2} < \cdots < x_{\sigma_k}$. This model is appropriate when we have access to a special chip that is capable of sorting.

(iii) THE BINARY IDENTIFICATION ORACLE. Here the oracle takes two keys as input, $x$ and $y$, and answers the question "is $x = y$?" This oracle is used in algorithms that verify passwords: $x$ is the hidden password, and $y$ is a password typed in by a user.

(iv) AN OLD-FASHIONED SCALE (OR: THE TERNARY COMPARISON ORACLE). A scale can compare two inputs, $x$ and $y$, but has three possible answers: $x < y$, $x = y$, $x > y$. It is, therefore, a ternary oracle.

The complexity

$$T(A; x_1, \ldots, x_n)$$

of a given algorithm $A$, operating on input $x_1, \ldots, x_n$, is the number of times it uses the oracle before halting. The fact that the complexity depends upon both algorithm and input is inconvenient. In many cases, we are interested in the worst-case time of an algorithm,

$$T_n(A) = \max_{x_1,\ldots,x_n} T(A; x_1, \ldots, x_n).$$

This is a function of $n$ and the algorithm. One can say that the study of algorithms is the hunt for methods that minimize, or nearly minimize, $T_n(A)$. But for a given problem, how large is the minimum,

$$\min_A T_n(A)?$$

The sub-field of lower bounds is concerned with methods for computing lower bounds for

$$\min_A T_n(A) = \min_A \max_{x_1,\ldots,x_n} T(A; x_1, \ldots, x_n).$$

With a good lower bound in hand, we have a gauge for the performance of a given algorithm. How close does $T_n(A)$ come the universal lower bound?

Note that a particular algorithm may run faster than the universal lower bound on some inputs. Indeed, the lower only applies to the worst input. And the worst input for one algorithm is not necessarily the worst input for another one.

There are three general paradigms for computing lower bounds:

(i) The decision tree (or information-theoretic) method.
(ii) The method of witnesses.
(iii) The method of adversaries.

We will only deal with the decision tree method in this course.

## §1.2. Lower bounds via decision trees

Every oracle-based algorithm can be visualized as a decision tree. In this tree, every node represents one use of the oracle, and the root stands for the first use of the oracle. Each of the $k$ replies of the oracle leads to a different subtree. When an algorithm halts, we insert a leaf in the decision tree. A particular run of an algorithm leads us from the root to a leaf via a path of length $\ell$ (say). This number, $\ell$, is the number of times the oracle was consulted before the algorithm halted. It is also equal to the depth of that leaf. If there are $L$ possible answers to the problem, then there must be at least $L$ different leaves. If $d_u$ denotes the depth of leaf $u$ and $\mathcal{L}$ is the set of leaves, then the worst-case complexity of a given algorithm is equal to the height of its decision tree,

$$h = \max_{u \in \mathcal{L}} d_u.$$

We will show below that any $k$-ary tree with $L$ leaves has

$$h \geq \log_k L,$$

and therefore,

$$\min_A T_n(A) \geq \lceil \log_k L \rceil.$$

THEOREM 1.1. *Any $k$-ary tree with $L$ leaves has height $\geq \log_k L$.*

PROOF. We show by induction on the height $h$ of the tree that $L \le k^h$. For $h = 0$, we can only have one leaf, the root. Therefore, $L = 1 \le k^h = 1$. Take a general $h > 0$ and assume that the assertion is true for all smaller heights. Then the root has up to $k$ subtrees. Let the number of leaves in the $i$-th subtree be $L_i$, and let its height be $h_i$. By the induction hypothesis, since $\max_i h_i \le h - 1$,

$$L_i \le k^{h_i}, 1 \le i \le k,$$

and thus

$$L = \sum_{i=1}^{k} L_i \le \sum_{i=1}^{k} k^{h_i} \le k \times k^{h-1} = k^h. \square$$

EXAMPLE 1. SORTING. Let us sort $n$ different numbers using a binary comparison oracle. It is clear that $L \ge n!$. Therefore,

$$\min_A T_n(A) \ge \lceil \log_2 n! \rceil.$$

Note that by comparison with an integral, we have

$$\begin{aligned} \log_2 n! &= \sum_{i=1}^{n} \log_2 i \le \int_1^{n+1} \log_2 x \, dx \\ &= (n+1)\log_2(n+1) - n \log_2 e \end{aligned}$$

and

$$\begin{aligned} \log_2 n! &= \sum_{i=1}^{n} \log_2 i \ge \int_1^{n} \log_2 x \, dx \\ &= n \log_2 n - (n-1) \log_2 e \end{aligned}$$

The difference between upper and lower bound is less than $\log_2(n+1)$. (One can obtain slightly sharper estimates by using Stirling's formula, $n! \sim (n/e)^n \sqrt{2\pi n}$, but we won't go there.) We know that mergesort requires not more than $n \log_2 n$ comparisons, and therefore, it comes within $O(n)$ of the lower bound, which is very good. It should be noted that computer scientists to this day do not know an algorithm that matches the $\lceil \log_2 n! \rceil$ lower bound for all $n$.

EXAMPLE 2. SEARCHING. We are given a sorted array with elements $x_1 < \cdots < x_n$, and an item $x$. We must locate $x$ in this array if it is present. Otherwise it must declare that it is not there. For all this, we have access to a binary comparison oracle. Now, what constitutes a leaf in the decision tree of an algorithm? If we know that $x$ is there, then there are just $n$ replies—this is the "successful search problem". In that case, $L = n$. However, in the general case, we might as well have asked to either report a match ($x = x_i$) or report an interval ($x_i < x < x_{i+1}$), because any correct algorithm must be able to produce this interval when $x$ is not in the array. Since there are $n + 1$ intervals that can be reported (the first and last one being half-infinite), we have $L = n + (n + 1) = 2n + 1$. Therefore,

$$\min_A T_n(A) \ge \lceil \log_2(2n+1) \rceil.$$

The standard binary search algorithm is optimal for this problem.

EXAMPLE 3. FINDING A SUBSET. The trivial problem here is report all items in an unsorted set $\{x_1, \ldots, x_n\}$ that are smaller in value than $x$ using a binary comparison oracle. Clearly, we have $2^n$ possible answers, one for each subset of the data. Therefore, for any algorithm $A$, we must have

$$\min_A T_n(A) \geq \log_2 2^n = n.$$

EXAMPLE 4. MERGING TWO SORTED LISTS. We are given two sorted lists, $A = (x_1, \ldots, x_n)$, and $B = (y_1, \ldots, y_m)$. To avoid uninteresting touble, we assume that all elements are different. Also, $n \geq m$, without loss of generality. An algorithm for merging these lists into one big sorted list has

$$L \stackrel{\text{def}}{=} \binom{m+n}{m}$$

possible answers, since it suffices to mark the positions of the elements of $B$ in final merged list. Therefore, with a binary comparison oracle, we have

$$\min_A T_n(A) \geq \left\lceil \log_2 \binom{m+n}{m} \right\rceil.$$

But do we have a sense of how large this really is? When $m = n$, we have, using Stirling's formula,

$$\binom{m+n}{n} = \binom{2n}{n} = \Theta\left(\frac{2^{2n}}{\sqrt{n}}\right)$$

so that the lower bound is $2n + o(n)$. A standard merge requires at most $2n - 1$ comparisons, and is thus close to optimal. When $m = 1$, the lower bound reduces to $\lceil \log_2(n+1) \rceil$. Thus, insertion of the sole element of $B$ in $A$ by binary search looks nearly optimal. But how about general $1 \leq m \leq n$? To deal with this, observe that

$$\binom{m+n}{m} = \frac{(m+n)(m-1+n)\cdots(1+n)}{m \times (m-1) \times \cdots \times 1} \geq \left(\frac{m+n}{m}\right)^m.$$

Therefore, our lower bound is

$$\min_A T_n(A) \geq m \log_2(1 + n/m).$$

To find an algorithm that is optimal or nearly so, we will develop a hybrid between the standard merge algorithm, and repeatedly inserting the elements from $B$ in $A$ by binary search. The worst-case complexity of the latter algorithm is at most $m + m \log_2 n$, and does not quite match the lower bound. The finger list algorithm first picks $m - 1$ equally spaced elements from $A$ and puts them in a (sorted) finger list, $F$. The elements are at positions $\lfloor in/m \rfloor$ in $A$, with $1 \leq i \leq m - 1$. They cut $A$ into $m$ shorter sorted lists, called minilists, each of length smaller than $n/m$. The algorithm proceeds as follows:

(i) Use standard merge to merge $F$ and $B$, using no more than $2m - 2$ comparisons. Each element of $B$ now knows which minilist it should join.

(ii) For each minilist in turn, insert the elements of $B$ that should join it using binary search. Observe that the sorted set into which binary insertion should be done is at most of size $n/m$. So the overall complexity of this is bounded from above by

$$m\,(1 + \log_2 n/m)\,.$$

The overall complexity of the finger list algorithm is bounded by

$$2m - 2 + m + m \log_2 n/m < 3m + m \log_2 n/m.$$

This is at most $3m$ more than the lower bound.

## §1.3. Guessing games

One popular guessing game is Mastermind. The house hides four pins in a certain order, where each pin is white, green, blue, red, yellow or black. Repetitions are allowed. The objective is to guess the four pins by asking questions to the house, which acts as an oracle. The oracle's input is a "guess", an ordered set of coloured pins. Its answer consists of two numbers, the number of pins that are of the correct color in the correct position, and the number of pins in the wrong position (after taking out the correct guesses). One can verify that the oracle has only 14 possible answers, namely (0,0), (0,1), (1,0), (0,2), (1,1), (2,0), (3,0), (2,1), (1,2), (0,3), (4,0), (2,2), (1,3), (0,4). Note that (3,1) is impossible.

Since there are $4^6$ possible outcomes, the lower bound, in terms of number of guesses is

$$\left\lceil \log_{14} 6^4 \right\rceil = 3.$$

The rules of the game require that one must end with a perfect guess, so one more round is needed, for a lower bound of four. In 1977, Knuth obtained this lower bound and designed an algorithm for playing Mastermind that always terminates in at most five moves.

## §1.4. Exercises

**Exercise 1.1.** Assume that we are given a comparison oracle with three outputs ($<$, $=$, $>$). We say that a set of numbers $x_1, \ldots, x_n$ is sorted if $x_1 \leq x_2 \leq \cdots \leq x_n$. Show that if $A$ is an algorithm for sorting, then

$$T_n(A) \geq \lceil \log_2 n! \rceil.$$

In other words, the bound for the two-output comparison oracle ($\leq$, $>$) is still valid.

**Exercise 1.2.** LOWER BOUNDS VIA DECISION TREES. Let the data consist of an $m \times m$ matrix of different integers that are sorted along all rows (increasing from left to right) and all columns (increasing from bottom to top). Set $n = m^2$. We have a comparison oracle at our disposal.

(i) Using the decision tree method, show that the worst-case number of times the oracle has to be consulted to sort these elements is $\Omega(n \log n)$.
(ii) Design an algorithm that can sort the elements in time at most $cn \log_2 n$ where $c < 1$.

**Exercise 1.3.** LOWER BOUNDS VIA DECISION TREES. Consider $n$ coins that are all of the same unknown weight except one. The following computational model is assumed: we have a scale on which we may place any number of coins on both sides. The scale tells us which side is heavier, if any. Each such use of the scale takes one time unit. The answers below should be exact (no big oh, for example).

(i) What is the decision tree lower bound for the number of time units (scale uses) needed to identify the unequal-weight coin?

(ii) And what is the decision tree lower bound for the number of time units (scale uses) needed to identify the unequal-weight coin if it is known beforehand that the special coin is heavier than the others?

(iii) If the special coin is heavy, and $n = 720$, how can you find it using a minimal number of weighings?

**Exercise 1.4.** LOWER BOUNDS VIA DECISION TREES. Let the data consist of an $n \times n$ matrix of different integers that are sorted along all rows (increasing from left to right) and all columns (increasing from bottom to top). We have a binary comparison oracle at our disposal. Let $x$ be an irrational number. Using the decision tree method, derive a lower bound for the worst-case number of times the oracle has to be consulted to identify all elements smaller than $x$. Answers that are suboptimal (in their dependence upon $n$) are not acceptable.

**Exercise 1.5.** Give the decision tree lower bound for merging a sorted list of length $n$ with $m$ unsorted items when we have a comparison oracle for "$x < y$". Provide efficient methods for this problem. You may wish to consider two different algorithms, depending upon which is larger, $m$ or $n$.

**Exercise 1.6.** MASTERMIND. Show that the general lower bound for Mastermind in case of $c$ colours and $n$ pins is

$$\frac{n \log c}{\log\left(\frac{n(n+3)}{2}\right)}.$$

(This attributed to Duchet, 1983, but follows easily from the argument in our set of notes.)