Chapter G1: Lower Bounds

In Algorithms, we are able to prove lower bounds only for very simple problems. We look here at some problems related to sorting.

G1.1 Information Theory

Suppose you have to guess a number between 1 and 100 by asking yes–no queries. You might start by asking “Is it 1?” Then “Is it 2?” and so on. But one can do better by asking first “Is it larger than 50?” and if the answer is yes then asking “Is it larger than 75?” and so on. In other words, we can perform a binary search. In the worst case the binary search will take 7 queries. (Try it!) The question is: Is 7 the best we can hope for?

The answer to that question is yes. And the argument is as follows. The problem is:

Determine a number between 1 and 100.

There are 100 possible solutions—call the set \( S \). Each step one asks a yes–no question. This splits the set up into sets \( S_Y \) and \( S_N \). If the person answers yes, then we are left with the set \( S_Y \). If she answers no, then we are left with the set \( S_N \). One of these sets must have at least 50 elements. So in the worst case we are left with at least 50 elements. In other words:

In the worst case the number of solutions comes down by a factor of at most 2 for each query.

We are done when there is only one possible solution remaining. So if we define \( I \) as the number of solutions to the problem then

The number of queries needed, in the worst case, is at least \( \log_2 I \)

In the example above, the lower bound is \( \log_2 100 \) which, rounded up, is 7. Note that:

- The lower bound does not assume a particular algorithm, but rather deals with the best one could hope for in any algorithm. It depends only on the number of solutions.

- The lower bound is also valid for the average case behavior of any algorithm. The idea is that one query will on average decrease the number of solutions by a factor of at most 2.

The information-theory lower bound is (essentially) the only general procedure out there for providing estimates of the time required by the best possible algorithm. In applying information theory, we must determine or estimate \( I \); this is not always so simple.
G1.2 Examples

Sorting.
Here the problem is: given a list of numbers, output them in sorted order. The solution
is the sorted list. For a list of \( n \) numbers there are \( I = n! \) solutions. So a lower bound
on sorting is given by

\[ \log_2(n!) \]

Since there is an approximation

\[ n! \approx \left(\frac{n}{e}\right)^n \]

It follows that \( \log_2(n!) \) is about \( n \log_2 n - O(n) \).

This means that any comparison-based sorting algorithm must take at least this many
comparisons (in the worst case). And hence the best we can hope for time-wise for a
sorting algorithm is \( O(n \log n) \). Merge sort comes close to this value.

Maximum.
Here the problem is: given a list of numbers, output the maximum. For a list of \( n \)
elements there are \( n \) answers. So a lower bound on maximum-finding is given by

\[ \log_2 n \]

However, there is no algorithm for finding the maximum that runs this fast. (This
problem is looked at again in the next section.)

Set-Maxima.
Suppose one has as input the values of \( A \), \( B \) and \( C \) and one must output the maximum in
each of the three sets \( \{A, B\} \), \( \{A, C\} \) and \( \{B, C\} \). Naively there are 2 possible answers
for the maximum in each set, so one might think that \( I = 2^3 \). However, this is not
correct. It cannot happen that the maximum in \( \{A, B\} \) is \( A \), the maximum in \( \{B, C\} \)
is \( B \) and the maximum in \( \{C, A\} \) is \( C \). (Why not?) In fact \( I = 6 \) (or \( 3! \)) as there is one
possible output answer for each ordering of the set \( \{A, B, C\} \).

G1.3 Adversarial Arguments

We want to know about the worst-case behavior of a comparison-based algorithm. Con-
sider, for example, computing the maximum of \( n \) elements using comparisons: this takes
at least \( n - 1 \) comparisons. The idea is that if one wants a winner out of \( n \) players, each
player except for the winner must lose at least one game, hence there must be \( n - 1 \)
games.

This type of argument can be generalized to what is called an **adversarial argument**: we assume that we are asking a question of an adversary who is making up the data as he goes along. Of course, he is not allowed to invalidate the answers to previous questions.
Consider the problem of computing the maximum and minimum of \( n \) elements simultaneously. For an adversarial argument, we must describe a strategy for the adversary. He will create two buckets: \( L \) and \( H \) for low and high. Initially, both buckets are empty. Each value in \( L \) will be less than each value in \( H \).

For each question (which is of the form “Is \( A[i] \) bigger than \( A[j] \)”), the adversary does the following.

1. If neither element has been assigned to a bucket, he arbitrarily puts one in \( L \), one in \( H \).
2. If only one element has been assigned to a bucket, then he puts the other element in the other bucket.
3. Otherwise the adversary does nothing.

Then to answer the question, the adversary chooses any answer that is compatible with previous information and respects the property that all of \( H \) is bigger than all of \( L \).

Obviously, the user must ask a question about each element (okay, we should assume \( n \geq 2 \)). So every element will eventually be assigned to a bucket. Furthermore, the user has to determine the maximum in \( H \) and the minimum in \( L \). Each question provides information about only one thing: (a) the assignment to buckets, (b) the maximum in \( H \), or (c) the minimum in \( L \).

Assigning to buckets takes at least \( n/2 \) comparisons. Determining the maximum in \( H \) and the minimum in \( L \) (which are disjoint) together takes \((|H| - 1) + (|L| - 1) = n - 2 \) comparisons. Thus, determining the maximum and minimum in the set takes at least \( 3n/2 - 2 \) comparisons.

**Exercises**

1. Sorting takes \( O(n \log n) \) comparisons. How long does it take to check whether a list is sorted or not?
2. Use an adversarial argument to give a \( 3n/2 - O(1) \) lower bound on the number of comparisons needed to find the median of a list of numbers.
3. Consider a set-maxima problem where one is given a list of \( n \) numbers and \( n \) subsets from this list, and one must find the maximum in each subset. Find an information-theoretic lower bound on the number of comparisons needed.