Chapter 6

A Lower Bound Technique

We introduce a very general tool for proving lower bounds for randomized algorithms: Yao’s minimax principle. After presenting the principle, we will give a simple example of its application, namely a lower bound for game tree evaluation.

6.1 Yao’s minimax principle

Consider a problem II. Let $\mathcal{A}$ denote a class of deterministic algorithms for II. We assume that $\mathcal{A}$ denotes an enumerable set of algorithms, e.g., given in form of programs for RAMs or Turing machines. Let $\mathcal{I}$ denote a finite set of input instances, e.g., all bit-strings of length $n$. Given an algorithm $A \in \mathcal{A}$ and an instance $I \in \mathcal{I}$, let $C(A, I)$ denote the cost of $A$ on $I$, e.g., the time spent by $A$ on $I$, or the memory requirement of $A$ on $I$, or other cost measures. A randomized algorithm $R$ is defined by a probability distribution over $\mathcal{A}$. This probability distribution is denoted by $p : \mathcal{A} \to [0, 1]$.

Typically, an upper bound for a randomized algorithms shows a bound of the following form:

$$\exists p : \mathcal{A} \to [0, 1] : \forall I \in \mathcal{I} : E[C(A_p, I)] \leq F(\mathcal{A}, \mathcal{I})$$

where $A_p$ denotes an algorithm drawn at random from $\mathcal{A}$ according to $p$ and $F$ denotes a function depending on the class of algorithms and inputs. (Typically the class of algorithms is fixed and $F$ depends only on the length of the input.) Correspondingly, lower bounds are of the following form:

$$\forall p : \mathcal{A} \to [0, 1] : \exists I \in \mathcal{I} : E[C(A_p, I)] \geq f(\mathcal{A}, \mathcal{I})$$
for some suitable function \( f \). The problem in proving such a lower bound is that one needs to make a statement that holds for all probability distributions. Yao’s lemma shows that it is sufficient to make a statement on the existence of a probability distribution. This distribution, however, is on the set of inputs.

**Lemma 6.1 (Yao’s Lemma)** Let \( p \) be a probability distribution over a class of algorithms \( \mathcal{A} \). Let \( q \) be a probability distribution over a finite set of inputs \( \mathcal{I} \). Furthermore, \( A_p \) denotes an algorithm drawn at random from \( \mathcal{A} \) according to \( p \), and \( I_q \) denotes an input drawn at random from \( \mathcal{I} \) according to \( q \). Suppose \( \mathbf{E}[C(A_p, I)] \) is finite, for all \( I \in \mathcal{I} \). Then

\[
\max_{I \in \mathcal{I}} \mathbf{E}[C(A_p, I)] \geq \min_{A \in \mathcal{A}} \mathbf{E}[C(A, I_q)] .
\]

Thus, in order to show a lower bound of \( f(\mathcal{A}, \mathcal{I}) \) on the cost of all randomized algorithms in \( \mathcal{A} \) on problems from \( \mathcal{I} \), one needs to show only

\[
\exists q : \mathcal{I} \to [0, 1] : \forall A \in \mathcal{A} : \mathbf{E}[C(A, I_q)] \geq f(\mathcal{A}, \mathcal{I}) .
\]

The advantage of this technique lies in the flexibility in the choice of \( q \) and, more importantly, in the fact that one does not have to argue about randomized algorithms with arbitrary probability distributions but only about the behavior of deterministic algorithms for a fixed input distribution \( q \).

**Proof.** We prove Lemma 6.1. Fix \( p \) and \( q \). Then

\[
\max_{I \in \mathcal{I}} \mathbf{E}[C(A_p, I)] = \max_{I \in \mathcal{I}} \sum_{A \in \mathcal{A}} p(A) C(A, I) \\
\geq \sum_{I \in \mathcal{I}} q(I) \sum_{A \in \mathcal{A}} p(A) C(A, I) \\
= \sum_{A \in \mathcal{A}} \sum_{I \in \mathcal{I}} q(I) C(A, I) \\
\geq \min_{A \in \mathcal{A}} \sum_{I \in \mathcal{I}} q(I) C(A, I) \\
= \min_{A \in \mathcal{A}} \mathbf{E}[C(A, I_q)] ,
\]

where equation (\#) follows because \( \sum_{A \in \mathcal{A}} p(A) C(A, I) \) and \( \mathcal{I} \) are finite. \( \square \)
6.2 Application of the minimax principle

6.2.1 Deterministic game tree evaluation

A binary game tree is a complete binary tree of height \( 2k \), for integral \( k \), in which internal nodes at even distance from the root are labeled AND and internal nodes at odd distance are labeled OR. Each leaf has a value from \( \{0, 1\} \). Each AND node returns the smallest value of its two children, each OR node returns the largest value. Let \( n = 4^k \) denote the number of leaf nodes.

The goal of game tree evaluation is to compute the value returned by the root. The cost of an algorithm is the number of leaf nodes that are accessed. (Observe that the number of accessed leaf nodes dominates the running time of any reasonable algorithm.) Clearly, one can evaluate the tree at cost \( n \), e.g., by scanning through the tree in bottom-up fashion.

Furthermore, there is a simple argument showing that the worst-case cost of any deterministic algorithm is at least \( n \). Consider a single AND node whose two children are leaf nodes. If the node were to return 0, at least one of its children must contain 0. A deterministic algorithm evaluates the values of the children in a fixed order, and it can therefore determine that the value returned first is always a 1 such that the algorithm needs to access the other child, too. This argument can be extended to all levels of the tree in such a way that the algorithm is forced to access all leaf nodes.

6.2.2 Randomized game tree evaluation

To evaluate an AND node \( v \), the algorithm choose one of \( v \)'s children at random and evaluates it recursively. If 1 is returned by this recursive calls, then the other child is evaluated, too. If 0 is returned then the other child need not to be evaluated. The procedure for OR nodes is analogously with the roles of 0 and 1 interchanged.

**Theorem 6.2** The above algorithm has expected cost at most \( n^{\log_4 3} = n^{0.792...} \).

**Proof.** Let \( C(h) \) denote the expected cost for evaluating a subtree of height \( h \), for even \( h \). We will show that \( C(h) \leq 3^{h/2} \). Thus, the expected cost for evaluating the complete game tree is \( C(2k) \leq 3^k = 3^{\log_4 n} = n^{\log_4 3} \).

Consider any subtree of height \( h \), for even \( h \). Let \( v \) denote the root of this tree. Observe that \( v \) is labeled with AND because \( h \) is even. We distinguish two cases.
• **Case 1:** Assume the value of $v$ is 0. Then at least one of the children of $v$ has value 0 so that the second child does not need to be evaluated with probability $\frac{1}{2}$. As the expected cost for each evaluated child can be estimated by $2C(h - 2)$, we obtain

$$C(h) \leq 2C(h - 2) + \frac{1}{2} \cdot 2C(h - 2) = 3C(h - 2) .$$

• **Case 2:** Assume the value of $v$ is 1. Then both children of $v$ necessarily have value 1. Hence, each child has expected cost $C(h - 2) + \frac{1}{2} \cdot C(h - 2) = \frac{3}{2}C(h - 2)$. In this way,

$$C(h) \leq 2 \cdot \frac{3}{2}C(h - 2) = 3C(h - 2) .$$

Clearly, $C(0) = 1 = 3^0$. Thus, the recurrence implies $C(h) \leq 3^{h/2}$. □

### 6.2.3 A lower bound for game tree evaluation

One can use the minimax principle in order to show that the upper bound above is optimal. We will a result that is slightly weaker.

**Theorem 6.3** Every Las Vegas algorithm for game tree evaluation has expected cost at least $n^{0.694...}$.

**Proof.** As a first step, we replace the AND-OR game trees by NOR trees, that is, an internal nodes returns 1 iff both children return 0. It is an easy exercise to show that both structures are equivalent in the sense that they return the value at the root.

We have to describe a probability distribution $q$ on the inputs that leads to high expected cost for any deterministic algorithm $A$. Next we describe this distribution $q$. Set $r = (3 - \sqrt{5})/2$. Each leaf of the tree is i.u.r. set to 1 with probability $r$. Suppose both children of a node $v$ have value 1 with probability $r$. Then, the probability that the NOR operation of $v$ returns a 1 is

$$(1 - r)^2 = \left(\frac{\sqrt{5} - 1}{2}\right)^2 = \frac{3 - \sqrt{5}}{2} = r .$$

Thus, as a result of our input randomization $q$, the value of every node of the NOR tree is 1 with probability $r$. 38
Now let $v$ denote the root of a subtree of height $h$. A result of Tarsi [1983] shows that the expected cost $C(h)$ for evaluating $v$ is minimal if the value of one of the children of $v$ is evaluated before one inspects any leaf in the subtree rooted at the other child. If the value of the first child is 1 then one can possibly skip the evaluation of the second child. Otherwise, the values of both children need to evaluated. The probability for the latter event is at least $(1 - r)$. As a consequence,

$$C(h) \geq C(h - 1) + (1 - r)C(h - 1).$$

Hence, $C(h) \geq (2-r)^h$. Substituting $h = \log_2 n$ yields that the expected cost for evaluating the complete tree is at least $n^{0.694...}$. \qed