Notes on Lower Bounds

Jinho Yoon

March 14, 2022

This is the augmented transcript of lectures given by Luc Devroye on February 1st and February 3rd, 2022 for an Honours Data Structure and Algorithms class (COMP 252). The subject was regarding lower bounds and methods of attaining them.

Lower Bounds: An Introduction

AN ORACLE IS AN ABSTRACT DEVICE that outputs an answer for some given input. Examples of oracles are:

- BINARY COMPARISON ORACLE: the oracle takes two inputs x and y, and answers "yes" if $x < y$, and "no" otherwise.
- k!-ORACLE FOR SORTING k KEYS: the oracle takes k inputs, $x_1, \ldots, x_k$ and returns the inputs sorted in order.

  This model is appropriate when we have access to a special chip capable of sorting.
- BINARY IDENTIFICATION ORACLE: the oracle takes two inputs x and y, and returns the answer to "is $x = y$?"

  This oracle is used in algorithms that verify passwords.
- TERNARY COMPARISON ORACLE (A SCALE): the oracle takes inputs x and y and returns one of three potential answers: $x < y, x = y$, and $x > y$.

THE COMPLEXITY OF AN ALGORITHM can be defined as the number of times an oracle has been used.

Say the time complexity of a given algorithm A on input $x_1, \ldots, x_n$ is given as $T(A, x_1, ...x_n)$.

Then, the worst-case time complexity of an algorithm is

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

where we manipulate the inputs $x_1, ...x_n$.

THE LOWER BOUND COMPLEXITY OF A PROBLEM is the time complexity of the fastest algorithm on the worst-case input. In equation form, we write,

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

In other words, we are getting the *minimum of a maximum*.

If $T_n(A) = \Theta(n), T_n(A) \geq c \times n$, this means that for all algorithms, there are worst-case inputs that lead to a time complexity of $\geq c \times n$.

3 TECHNIQUES FOR CALCULATING THE LOWER BOUND:

- The method of decision trees (information theoretic lower bounds)
- The method of witnesses
- The method of adversaries

We will first discuss **the method of decision trees**.

The Method of Decision Trees

Every oracle-based algorithm can be visualized as a decision tree[1]. Each node in a decision tree corresponds to one usage of the oracle, and the k replies possible by the oracle correspond to the k subtrees. A leaf indicates that an answer has been reached and the algorithm has halted.

[1] Luc Devroye. Chapter 2. Lower bounds. McGill University, February 2022

THE LOWER BOUND OF AN ALGORITHM corresponds to the height h of the decision tree by the formulas:

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

THEOREM: Any k-ary decision tree with L leaves has height $h \geq \log_k L$.

PROOF: We use induction to show that $L \leq k^h$.

- BASE CASE: For $h = 0$, there is only one leaf. Thus $L = 1 \leq k^h$.
- INDUCTION HYPOTHESIS: Assume $L \leq k^h$ is true up to $h - 1$, $(h > 0)$.
- INDUCTIVE STEP: Create a decision tree with a root and k subtrees each of height $h - 1$. The new decision tree has height h. As there are $\leq k^{h-1}$ leaves in each k subtree, for the entire decision tree, $L \leq k \times k^{h-1} = k^h$.

  Remark: when doing induction on trees, add to the root, not at the bottom, as the tree may be imbalanced, not "flush", at the bottom.
- CONCLUSION: $L \leq k^h$ for all $h \geq 0$. □

Applying $\log_k$ to both sides, you get $h \geq \log_k L$.

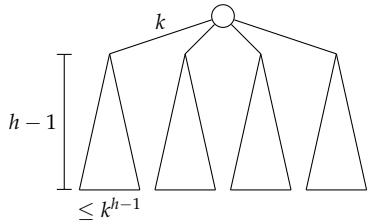
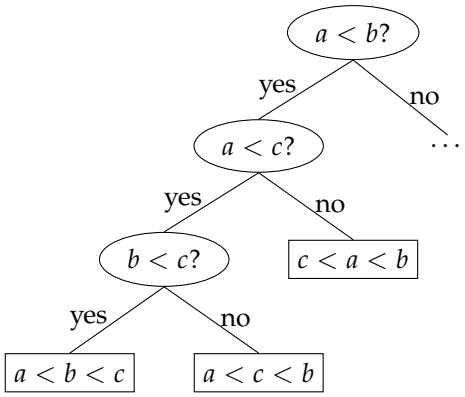


Figure 1: Visualization of proof of lower bound formula. Add a new root at the top with k subtrees of $h - 1$ height. It is clear that the new total number of leaves $\leq k \times k^{h-1}$.

1. Sorting a list of numbers

Sorting a, b, and c

PROBLEM: Sort the numbers a, b, and c using a binary comparison oracle.
ANALYSIS: Using a binary comparison oracle, we can make the decision tree as shown below:



Once fully expanded, the decision tree has 6 leaves, for the $3! = 6$ permutations of a, b, and c.

Using our theorem from above, we get

$$\min_A T_n(A) \geq \lceil \log_2 3! \rceil = 3.$$

Our lower bound of 3 steps corresponds to the height of the decision tree.

Sorting n numbers

PROBLEM: Sort a list of n numbers.

ANALYSIS: For sorting a list of n numbers, there are $n!$ (permutations) possible answers. Thus, there are $n!$ leaves. With a binary comparison oracle, $k = 2$ (a binary decision tree).

Using our theorem from above, we get, [2]

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

[2] *Remark:* There is no sorting algorithm as of yet that matches this $\lceil \log_2 n! \rceil$ lower bound for all n.

Note that, using logarithm arithmetic and integrals,

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

Therefore, we have

$$\min_A T_n(A) \geq n \log_2 n - (n-1) \log_2 e.$$

Effective sorting algorithms like mergesort take $n \log_2 n + O(n)$ comparisons, and are thus close to our lower bound.

2. *Mastermind*

(i) The Game

Mastermind is a game in which one player (the *codemaker*) makes a sequence of 4 pegs (that can each be one of 6 colors), and the other player (the *codebreaker*) must guess this sequence in 12 tries.

The game is illustrated in Figure 2 on the right: the first row of circles is the code created by the *codemaker*, and the second row is the *codebreaker*'s guess.

For every guess, the oracle tells the codebreaker the number of pegs with the right color *and* position, and the number of pegs with the right color but in the *wrong* position.

(For the guess in Figure 2, the oracle would give an answer (2, 1): the 2nd and 3rd pegs are in the right position, while the 4th one is in the wrong position.)

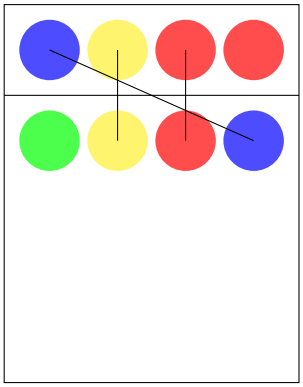


Figure 2: The game of Mastermind visualized.

(ii) Lower Bound by Method of Decision Trees

There are 14 possible answers that the oracle can give:

(right, wrong)
$(0,0)$
$(0,1)\ (1,0)$
$(0,2)\ (1,1)\ (2,0)$
$(0,3)\ (1,2)\ (2,1)\ (3,0)$
$(0,4)\ (1,3)\ (2,2)\ \cancel{(3,1)}\ (4,0)$
= 14 possible answers.

The answer (3,1) is impossible as if 3 pegs are in the right position, then the last peg cannot be in the wrong position.

Therefore, with $k = 14$, we have a 14-ary decision tree. The number of leaves = 6^4 as there are 6 colors and 4 pegs. Thus, our lower bound[3] is

$$\min_A T_n(A) \geq \lceil \log_{14} 6^4 \rceil = \lceil 4 \log_{14} 6 \rceil = 3.$$

As Mastermind requires one more guess to be made as our correct answer, we need $3 + 1 = 4$ steps to win.

[3] Luc Devroye. Chapter 2. Lower bounds. McGill University, February 2022

3. Simple Problems

The table below lists the lower bounds of some simple problems involving n inputs consisting of numbers, using a binary comparison oracle. Each lower bound is acquired by applying the formula $\lceil \log_2 L \rceil$ where $L =$ the number of leaves / answers.

Problem	**# of answers**	**Lower bound**
Report all numbers	2^n	n
Search for x (successful)	n	$\lceil \log_2 n \rceil$
Search for x (unsuccessful)	$n+1$	$\lceil \log_2 (n+1) \rceil$
Search for x (general)	$2n+1$	$\lceil \log_2 (2n+1) \rceil$
Sorting	$n!$	$\lceil \log_2 n! \rceil$

REMARK: a search is **successful** when x is in the list of n numbers. Therefore, there are n possibilities. A search is **unsuccessful** when x is *not* in the list; therefore x is either smaller than the smallest input, larger than the largest input, or in between the n inputs. There are $n+1$ possibilities. A **general** search is a search that can be successful or unsuccessful. Therefore there are $n + (n+1) = 2n+1$ possible answers.

4. Merging two sorted sets

PROBLEM: There are two sorted sets, set X with n elements $x_1 < \cdots < x_n$, and set Y with m elements $y_1 < \cdots < y_m$. Assume $m \leq n$ and assume that all elements are different.

(i) Non-Optimal Solutions

STANDARD MERGE APPROACH: Iterate through all elements in both sets in order and use a similar merge method as in a mergesort algorithm. As each element is visited once, the worst-case time taken is $= n + m - 1$.

REPEATED BINARY SEARCH: If m is relatively small, we have a faster algorithm than the standard merge approach by doing binary search for elements of Y to locate where each element should be placed in X.

As there are $n+1$ possible places y_i can be placed in X (see unsuccessful search above), one binary search can be done using not more

than $\lceil \log_2(n+1) \rceil$ comparisons. For all m elements, thus, the time taken is $\leq m\lceil \log_2(n+1) \rceil = m + m\log_2(n+1)$.

However, the repeated binary search approach only works when m is small. In order to find an algorithm for general m, we will take inspiration from the lower bound.

(ii) Lower Bound

The number of possible answers (leaves) is

$$L = \binom{m+n}{m}$$

as there are m elements being inserted into a new merged list of $m+n$ elements.

Thus, with a binary comparison oracle, we have a lower bound of

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

As

$$\binom{m+n}{m} = \frac{(m+n)(n+m-1)\dots(n+1)}{m(m-1)\dots 1},$$

if we split up the fraction as seen below, we can derive a lower bound for L with the smallest fraction $\frac{(m+n)}{m}$.

$$\begin{aligned}\binom{m+n}{m} &= \overbrace{\boxed{\frac{(m+n)}{m}}}^{\text{smallest}} \times \frac{(m+n-1)}{(m-1)} \times \dots \frac{(n+1)}{1} \\ &\geq (\frac{m+n}{m})^m.\end{aligned}$$

Therefore, our updated lower bound is

$$\min_A T_n(A) \geq m\log_2(1+\frac{n}{m}).$$

Sometimes the lower bound can inspire the optimal solution. The $\frac{n}{m}$ in our lower bound indicates to us that perhaps we divide our list of n elements in to m groups.

(iii) Optimal Solution

1. Create a "finger list" that contains the elements in X that have position that is a multiple of $\frac{n}{m}$.

   In other words, add $x_{\frac{n}{m}}, x_{\frac{2n}{m}}, \dots, x_{\frac{m-1}{m}n}$ to the finger list.

2. Merge the set Y and the finger list (using the standard merge method).

3. Insert the elements of Y by repeated binary search into the respective ranges defined by the finger list as shown in Figure 3.

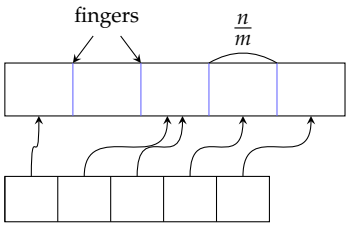


Figure 3: Repeated binary search into the ranges defined by the finger list. The merged list from Step 2 tell us which range the the element belongs to.

TIME COMPLEXITY: Each step of the algorithm costs:

1. Creating the finger list: $\leq m - 1$.
2. Merging the set Y and the finger list: $\leq 2m$.
3. Inserting Y by repeated binary search: $\leq m \times \lceil \log_2(1 + \frac{n}{m}) \rceil$.

In total, the time complexity of our algorithm is

$$\begin{aligned} T_n &\leq (m-1) + 2m + (m \times \lceil \log_2(1 + \frac{n}{m}) \rceil) \\ &\leq m \lceil \log_2(1 + \frac{n}{m}) \rceil + 3m - 1. \end{aligned}$$

Our time complexity is close but still greater than our lower bound of $m \log_2(1 + \frac{n}{m})$.

5. Median of 5 numbers

There are examples where the method of decision trees does not provide us with the proper lower bound. Such an example is the median-of-5-numbers problem.

WITHOUT DECISION TREES, we know there are $5 \times \binom{4}{2} = 30$ possible answers. Each of the five numbers can be the median, and the order of the numbers smaller or larger do not matter. As there are two numbers smaller than the median, there are $\binom{4}{2}$ permutations for each 5 possible medians. Hence, the formula $5 \times \binom{4}{2} = 30$. The lower bound is thus $\lceil \log_2 30 \rceil = 5$.

THE DECISION TREE, however, using a binary comparison oracle has 36 leaves, which gives the lower bound of $\lceil \log_2 36 \rceil = 6$. The lower bound we derived without decision trees is smaller!

The Method of Witnesses

The witness is the proof provided as evidence for having achieved the correct output. It is a rather weak method.

PROBLEM: Find the maximum of n numbers using a binary comparison oracle.
ANALYSIS: Make a comparison graph between vertices of all the numbers and let each comparison draw an edge from i to j if $i < j$. This graph is a directed acyclic and connected graph.

A graph that connects all n numbers must have at least $n - 1$ vertices, and all the edges will be pointed towards the root. The root is the maximum. See Figure 4.

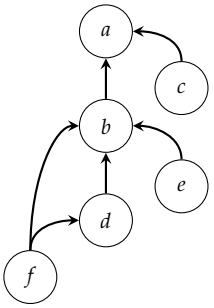


Figure 4: A comparison graph of 6 numbers. We can see that the maximum number is a. The graph is our "witness".

Hence, this graph is a "witness" of the maximum. As there are at least $n-1$ comparisons,

$$\min_A \max_{x_1,\ldots,x_n} T(A; x_1, \ldots, x_n) \geq n-1.$$

The Method of Adversaries

THE ADVERSARY, also referred to as "the devil"[4], controls the oracle in the hopes of slowing the algorithm down as much as possible.

The adversary achieves this by assigning certain values to the input items while replying to queries in a consistent manner. In other words, **the adversary constructs a bad input on the spot**. [5]

The user/algorithm has no idea that the values of the inputs are being decided on the spot as the oracle is being used.

IT IS A FREQUENT MISCONCEPTION that the adversary knows what the algorithm does. This is not the case. Do not make your adversary as according to some algorithm you already have in mind.

[4] Sally A. Goldman and Kenneth J. Goldman. Adversary Lower Bound Techniques. Washington University in St. Louis, 2007

[5] *Remark:* the method of adversaries can give a different lower bound than the method of decision trees.

1. Guessing a password

PROBLEM: guess a password x that has n bits and can be the binary expansion of any integer between 0 and $2^n - 1$. We have a binary identification oracle.

ADVERSARY STRATEGY: the adversary can always answer "wrong password" for the first $2^n - 1$ guesses, and finally declaring "correct password" on the last 2^nth guess. The password x is decided as the 2^n-th guess.

We thus have a lower bound of

$$\min_A \max_x T(A;\, x) \geq 2^n.$$

2. Another Proof for Decision Tree Lower Bound

With a k-ary decision tree, we know the following to be true:

$$\text{Size of subtree at the root (level 1): } \geq \frac{L}{k}$$
$$\text{Size of subtree at level } l\text{: } \geq \frac{L}{k^l}.$$

THE ADVERSARY CAN FORCE THE ALGORITHM to take l steps, or l levels down the decision tree as long as $\frac{L}{k^l} \leq 1$. In other words, the adversary forces the algorithm to take a number of steps l until it reaches a leaf (≤ 1).

Therefore, we get the equation

$$\text{As } \frac{L}{k^l} \leq 1,\ l \geq \lceil \log_k L \rceil.$$

As $l =$ the number of calls the algorithm makes to the oracle,

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

3. *Finding Largest and Smallest*

PROBLEM: given a binary comparison oracle, find the largest and smallest of n items.

Let us first devise an effective algorithm before analyzing the lower bound.

(i) Algorithm

NAIVE APPROACH: Iterate through the list and find the largest element in $n-1$ comparisons. Then, iterate through the list again (excluding the newly found largest element), and find the smallest element in $n-2$ comparisons.

$$T_n(A) = \underbrace{(n-1)}_{\text{largest}} + \underbrace{(n-2)}_{\text{smallest}} = 2n-3.$$

But we can do better than $2n-3$ with divide-and-conquer.

DIVIDE-AND-CONQUER APPROACH:
First, compare all n items in pairs as shown in Figure 5. Then compare all the "winners" (the larger item in the pair) to find the largest value, and compare all the "losers" (the smaller item in the pair) to find the smallest value.

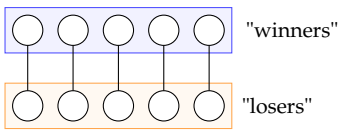


Figure 5: the divide-and-conquer approach; pair up the items, and find the largest of the "winners" and the smallest of the "losers".

The complexity of the algorithm is:

- Pairwise comparisons: $\lfloor \frac{n}{2} \rfloor$ comparisons,
- Compare "winners"/"losers": $2 \times (\lfloor \frac{n}{2} \rfloor - 1)$ comparisons.

In total [6],

$$\begin{aligned} T_n(A) &= \left\lfloor \frac{n}{2} \right\rfloor + 2 \times \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 \right) + (2 \text{ if } n = \text{odd}) \\ &= 3 \left\lfloor \frac{n}{2} \right\rfloor - 2 + (2 \text{ if } n = \text{odd}). \end{aligned}$$

We will see that this complexity is close to our lower bound derived by the method of adversaries.

[6] If $n =$ odd, then, at the end, compare the odd one out with the largest from the "winners" and the smallest from the "losers". This adds 2 comparisons.

(ii) Adversary's Strategy

Here the adversary assigns values to an element as soon as that element is first presented to the oracle. The values are set by the "clock" variable t, which is the number of uses of the oracle at that time.

Remember that the elements are initially all unvalued, and **are assigned values by the adversary** over time to ensure the worst-case inputs. The adversary's rules, visualized in Figure 6, are:

1. If the algorithm asks for a comparison between **two unvalued elements** at time t, assign values $+t$ and $-t$.
2. If the algorithm asks for a comparison between one unvalued element and one with value $+v$, assign the value $-t$ on the unvalued element. (If input $-v$, then assign value $+t$).
3. If the algorithm asks for a comparison between two valued elements, reply truthfully.

Remark: what is meant by "reply truthfully" is that the adversary cannot assign any new values, or lie.

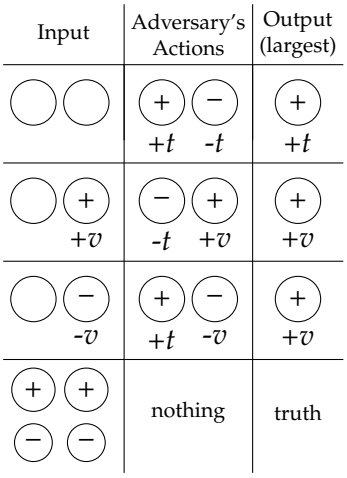


Figure 6: The adversary's rules depending on input. The value of an item, if known, is shown under the item. The sign inside the item indicates what group the item is put in (+ for group P, – for group N). The oracle outputs the *larger* of the two inputs.

Every time a new element is given a value, place that element in its respective group P or N: P holds all elements that have positive values, while N holds elements that have negative values. Each time a comparison is made between two elements of the same sign (+ and + or – and –), an edge is drawn between the elements to indicate a comparison, as in a comparison tree. This process is visualized in Figure 7. A comparison tree with all elements in the group has the maximum / minimum element as its root.

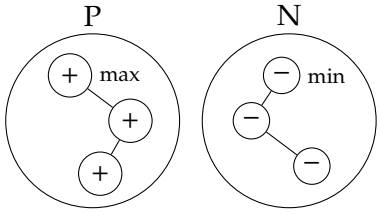


Figure 7: The groups P and N contain comparison trees.

The number of comparisons forced to be made by the adversary on the algorithm: at least $\lceil \frac{n}{2} \rceil$ comparisons of rules 1 or 2 must be made in order to "empty" the original bag of n numbers into P and N.

A comparison tree in P with all elements of P requires $|\mathrm{P}| - 1$ comparisons, while a comparison tree in N with all elements of N requires $|\mathrm{N}| - 1$ comparisons.

Therefore, the lower bound, or total number of comparisons, is

$$\begin{aligned}
\min_A T_n(A) &\geq \left\lceil \frac{n}{2} \right\rceil + (|\mathrm{P}| - 1) + (|\mathrm{N}| - 1) \\
&= \left\lceil \frac{n}{2} \right\rceil + n - 2 \qquad [\text{as } |P| + |N| = n] \\
&= 3 \left\lfloor \frac{n}{2} \right\rfloor - 2 + (2 \text{ if } n = \text{odd}).
\end{aligned}$$

4. Finding the Largest and 2nd Largest

PROBLEM: Find the largest and second largest element out of n elements.
NAIVE APPROACH: Use $n-1$ comparisons to find the largest element, and then use $n-2$ comparisons (excluding the largest) to find the second largest. Time complexity: $\geq 2n-3$.

(i) Tournament Algorithm

In a tournament, every match is a pairing of two elements. At the first round, n teams pair up into $\frac{n}{2}$ matches. The winners move up to the next round, where $\frac{n}{2}$ teams pair up into $\frac{n}{4}$ matches, and so on until 1 team remains.

We can use this tournament idea for the n elements, where every match is a comparison. The tournament can be represented as a complete binary tree with n leaves.

The $n-1$ matches in a tournament correspond to $n-1$ non-leaf nodes. Hence, the tree has $2n-1$ nodes. The root is the winner of the tournament, the largest element.

THE SECOND LARGEST must have played with the winner at some point on the winner's path to the root. The winner's path is visualized in Figure 8. The winner's path has the same height as the tree, $\lfloor \log_2(2n-1) \rfloor$.

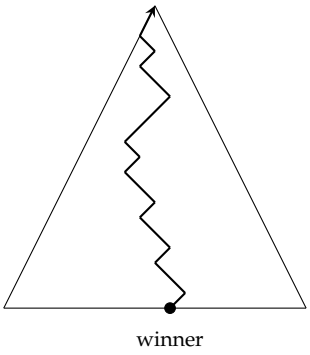


Figure 8: The winner's path in the tournament tree; note that the second largest must have played with the winner on the victory path.

Therefore, for the tournament algorithm,

$$\begin{aligned} T_n(A) &\leq \underbrace{(n-1)}_{\text{\# of matches}} + \underbrace{(\lfloor \log_2(2n-1) \rfloor)}_{\text{winner's path}} \\ &= n + \log_2 n + O(1). \end{aligned}$$

We will derive a lower bound by method of adversaries that shows that the tournament algorithm is optimal, up to a constant term.

(ii) Adversary's Strategy

The adversary's strategy is to mimic a tournament to determine the values of the elements. *Note:* this is not to say that the adversary is aware of what type of algorithm is being used.

Each element has a weight and a value. Each element starts with a weight of 1.

As long as an element has a weight, its value has not been determined yet.

The adversary's rules, visualized in Figure 9, are:

1. If the algorithm asks for a comparison between **two elements with weights** w and w' and no values, where $w \geq w'$, assign weight $w + w'$ to the element with previous weight w and assign weight zero and value t to the element of previous weight w'. In other words, the element with the greater weight *swallows* the weight of the other element, and the loser w' element loses its weight and is given the value t.
2. If the algorithm asks for a comparison between one element with a value and another element with weight w and no value, return the weighted element as the larger. Do not assign any new values.
3. If the algorithm asks for a comparison between **two unweighted elements**, tell the truth.

Input	Adversary's Actions	Output (largest)
w w' $(w \geq w')$	$w+w'$ 0 $+t$	$w+w'$
0 w	nothing	w
0 0	nothing	truth

Figure 9: The adversary's rules depending on input. The adversary changes the weight (what is inside the item) and value (what is below). The weight of an item is set to zero when the value is set. The oracle outputs the *larger* of the two inputs.

(iii) Lower Bound

Eventually, the weight of the largest element is n, as it has *swallowed* all the other elements' weights. As Rule 1 of our adversary states, an element with weight w can only add weight w' under the condition $w \geq w'$. In other words, the winner can at most double its weight at each comparison. After k comparisons, the winner has weight $\leq 2^k$. At the root, the winner has weight $n \leq 2^k$. Thus, the winner was involved in $k \geq \lceil \log_2 n \rceil$ comparisons.

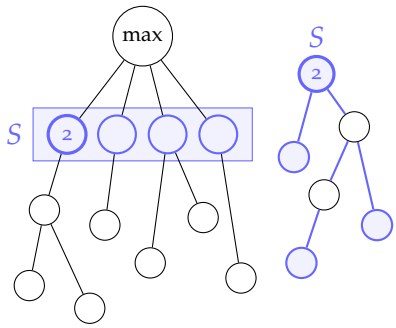


Figure 10: Comparison trees created by the tournament adversary. The blue tree on the right is the comparison tree of $|S| - 1$ size that derives the second largest. The second largest is labeled as 2.

The comparison tree of the tournament as in Figure 10 shows that the winner has at least $\lceil \log_2 n \rceil$ children with weight > 0 at the time of their comparison with the largest. Call the set of $\lceil \log_2 n \rceil$ compared nodes as the set S. The second largest element must be in this set S. There exists a comparison tree that visits all nodes and includes all comparisons between nodes in S and the root. This comparison tree has $n - 1$ edges.

For the second largest element to be decided, there must exist a separate comparison tree with the second largest element as the root, that includes all the other elements of S. Therefore, the two comparison trees [7] taken together give us the lower bound of

$$\begin{aligned} \#\text{ of comparisons} &\geq (n-1) + (|S|-1), \\ \min_A T_n(A) &\geq n + \lceil \log_2 n \rceil - 2. \end{aligned}$$

As we can see, our tournament algorithm is very close to the lower bound.

[7] *Remark:* A comparison graph must be distinguished from a comparison *tree*, which unlike a graph cannot have any cycles. The adversary creates a comparison graph, but the comparison *tree* within is what is necessary.

5. *Finding the median of 2n + 1 items*

PROBLEM: Find the median of $2n + 1$ items.
ANALYSIS: We will show that the adversary will force any algorithm to make at least $3n$ comparisons.

(i) Adversary's Strategy

The adversary's strategy is to assign n negative values and n positive values, and only after those $2n$ values have been assigned, assign the last item the value of 0 (which will be the median).

The adversary has rules that ensure that n values will be negative and positive each. The positive values are grouped together in group P and negative values are in group N. If P has n elements, then it is full, and thus no more elements should be added to P. The same applies to N. We will once again use the oracle's "clock" for these values. The adversary's rules, visualized in Figure 11, are:

1. If the algorithm asks for a comparison between **two unvalued elements**,
   - If neither group P or N has n elements, assign one element to value $+t$ and the other $-t$, where t is the clock time.
   - If one group has n elements (it is full), assign both elements to the other group with values t and $t + \frac{1}{2}$ (if P) or $-t$ and $-t - \frac{1}{2}$ (else).
   - If one group has n elements and the other has $n - 1$ elements, assign one element to the non-full group, and assign the other element the value 0 (the median).
2. If the algorithm asks for a comparison between **one unvalued element** and one valued element x,
   - If x is positive and $|N| < n$, set the unvalued element to value $-t$. If $|N| = n$, give value $+t$.
   - If x is negative and $|P| < n$, set the unvalued element to value $+t$. If $|P| = n$, give value $-t$.

   If $|N| = |P| = n$, give value of 0 (the median).
3. If the algorithm asks for a comparison between **two valued elements**, then answer truthfully.

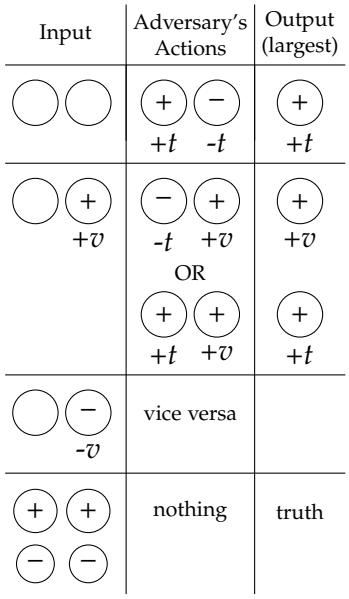


Figure 11: The adversary's rules depending on input. The adversary sets values on items. The values are shown below an item. ⊕ means that the element is in P and ⊖ means that it is in N. When only one input has a negative value (third row), the adversary acts in the same way as with one positive value (second row) but with the signs flipped.

(ii) Lower Bound

All $2n+1$ elements must be given values, requiring $\geq n$ comparisons to put every element into either P or N. There must be a comparison tree in each group that connects all elements in the group to the median in the middle (see Figure 12). Therefore, the comparison tree made in each group has $n+1$ nodes, which requires n comparisons per tree. In other words, $2n$ comparisons must be made to form the two comparison trees.

The total number of comparisons must be $\geq 3n$.

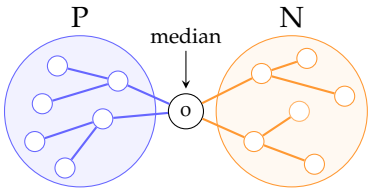


Figure 12: Comparison trees of the adversary; all elements are connected to the median.

References

Luc Devroye. Chapter 2. Lower bounds. McGill University, February 2022.

Sally A. Goldman and Kenneth J. Goldman. Adversary Lower Bound Techniques. Washington University in St. Louis, 2007.