Lower Bounds in a Decision-Based Model

Decision Trees and
Adversarial Arguments

Lower Bounds

Recall that a ***problem*** can have upper and lower bounds.

The upper bound of a problem is basically the worst case runtime of the best algorithm that we have available so far.

The lower bound is the amount of work that any (including the optimal) solution would *still* **have to do** in the **worst** case.

NOTE: The lower bound is ***not*** defined by the runtime of the best algorithm that *we* can think of, or that can be created using the techniques of which *we* know.

Some math of potential interest…

Claim: $\left(n/2\right)^n \in o\left(n^n\right)$

Proof:

$\forall c>0, \exists n_0 \in \mathbf{Z}^+,$

$\forall n \in \mathbf{Z}^{\geq n_0}, (n/2)^n < c \cdot n^n$

Sorting

We've seen how QuickSort has a worst-case of n^2 yet its expected runtime is $n\log n$.

In practice, for really small lists, InsertionSort works pretty well.

However, when talking about an **upper-bound** for the general problem of "comparison-based sorting" an algorithm such as MergeSort is what we would point to since its worst case is $n\log n$.

What about the lower bound for the problem?

Lower Bound for Sorting

Let's think about a decision tree that represents data comparisons.

The leaves of the tree would end up representing the different ways the program could end, thus being the different possible re-orderings of the input values $a_1..a_n$.

How many different re-orderings of the input values $a_1..a_n$ exist?

How many levels are required in our decision tree?

$\log(n!) \in \Theta(n \log n)$

Now that we know the height of the optimal decision tree would be at least $\log(n!)$ we need to prove it is in $\Theta(n \log n)$.

Big-O:

$\exists c, n_0 > 0, \forall n \in \mathbf{Z}^{\geq n_0}, \log(n!) \leq cn \log n$

Big-Ω:

$\exists c, n_0 > 0, \forall n \in \mathbf{Z}^{\geq n_0}, cn \log n \leq \log(n!)$

Merging Two Lists

Earlier, we looked at the problem where we have two *ordered* lists of equal sizes that we want to merge into a single ordered list.

We looked at an algorithm to do this as part of looking at MergeSort. If the length of each list was m, then in the worst case the algorithm for merging two lists runs in $2m-1$ time.

This sets an upper bound to the problem for us.

What is the lower bound for the problem in the worst-case?

Searching a List

When searching an *unordered* list for a specific value, what is the lower bound?

Note that there multiple ways to generate potential lower bounds, just like there are many ways to write algorithms for the upper bound.

We look for higher and higher lower bounds and lower and lower upper bounds, until (ideally) they match.

Searching an Ordered List

When searching an *ordered* list for a specific value, what is the lower bound?

Finding the 2nd Smallest

We can find the 2nd smallest element of an unordered list of n values in (n-1)+(n-2) or 2n-3 time "trivially".

Can we do better? We know in the comparison-based model that by using information we gain along the way we can find the minimum and maximum at the same time using only 3n/2-2 comparisons.

What if I said we can find the 2nd smallest element using only $(n-1)+(\lceil \log_2(n) \rceil - 1)$ comparisons.

What about other decisions?

A "comparison" can be thought of as the decision "Given these two values, how would you order them?"

The decision tree approach that we used to determine a lower bound for searching an ordered list and sorting an unordered list can be used with decisions *other than* comparisons.

For example, what if what we counted as a single "decision" was:

- Given 8 elements from the list and a search key, does that key match any of the 8 elements?
- Given 20 elements from the list, what is the smallest?
- Given half the elements from the list, what is the largest?
- Given sqrt(n) elements from the list, how would you order them?