CMSC 351: Limitations on Comparisons

Justin Wyss-Gallifent

October 16, 2023

1 Introduction to Decision Trees

Note that the fastest sorting algorithms we have seen having a worst-case time complexity $T(n) = \mathcal{O}(n \lg n)$. We might ask if there is some sorting algorithm which has a smaller worst-case time complexity.

Before answering this question let's take a second to observe that all the sorting algorithms we've looked at so far (BubbleSort, SelectSort, InsertSort, HeapSort, MergeSort, and QuickSort) are all comparison-based. What this means is that they all work by comparing pairs of numbers repeatedly in various ways. This seems like an obvious necessity because we're trying to sort and sorting is based on some sort of comparison.

Before we check out some non-comparison based sorting algorithms let's take an abstract look at these sorting algorithms. What we mean by that is - let's look at them as a whole, rather than individually.

2 Decisions

Every comparison-based sort algorithm involves a number of comparisons as it manages the data. Most of those comparisons lead to decisions such as "If $X > Y$ do this, otherwise do that". There always a minimum number of decisions that the algorithm must make. Consider these examples of lists and sorting them.

Example 2.1. Suppose a list is $[X, Y]$ and it needs to be sorted. We'll implement ImmortalSort. Only one decision needs to be made:

```
Is X < Y ?
├─ Yes: [X, Y]
└─ No: [Y, X]
```

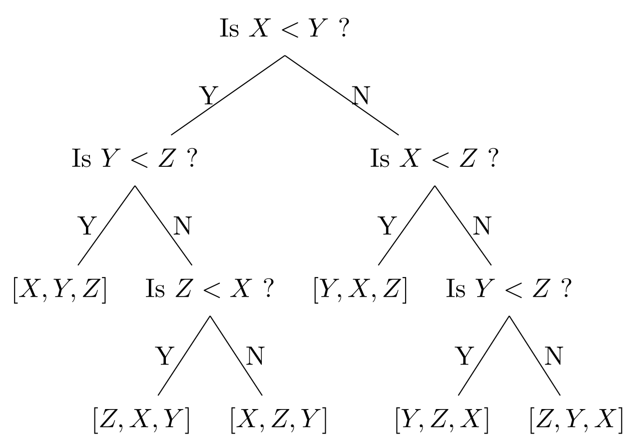
The answer to this question is sufficient to sort the data and this question is absolutely necessary.

Example 2.2. Suppose a list is $[X, Y, Z]$ and it needs to be sorted. We'll implement ImmortalSort. Several decisions need to be made. Let's think them out in a nested manner:

Is $X < Y$?
- Yes: Is $Y < Z$?
  - Yes: $[X, Y, Z]$
  - No: Is $Z < X$?
    - Yes: $[Z, X, Y]$
    - No: $[X, Z, Y]$
- No: Is $X < Z$?
  - Yes: $[Y, X, Z]$
  - No: Is $Y < Z$?
    - Yes: $[Y, Z, X]$
    - No: $[Z, Y, X]$

Observe that at different junctions we need to think differently. For example if $X < Y$ and $Y < Z$ then we're done at $[X, Y, Z]$ but if $X < Y$ and $Y \not< Z$ then we could have either $[Z, X, Y]$ or $[X, Z, Y]$ and we need another decision.

Observe that this series of decisions forms a full binary tree (every non-leaf node has exactly two children):



This series of decisions is not the only way we could sort the data. We could have started with a different initial question, for example.

3 Decision Tree for An Algorithm

Each and every comparison-based sorting algorithm must make a series of decisions as it sorts the data and they might be different from method to method.

The decision tree for a comparison-based sort algorithm is then the full binary tree which displays all the decision branches which arise from the algorithm's comparisons.

Consider BubbleSort. This implementation is specifically written so that all comparisons are `<=`.

```
for i = 0 to n-1
    for j = 0 to n-i-2
        if A[j] <= A[j+1]
            nothing
        else
            swap A[j] and A[j+1]
        end
    end
end
```

Let's see how this works on a list of length 3. Before proceeding observe that with a list of length 3, BubbleSort will check...

- Is $A[0] < A[1]$? Swap if false.
- Is $A[1] < A[2]$? Swap if false.
- Is $A[0] < A[1]$ again? Swap if false.

This implementation of BubbleSort may make useless comparisons and we should not count those as decisions. We'll note as we go through.

Assume the original list is $A = [X, Y, Z]$ unsorted. There are six possibilities:

1. Starting with $A = [X, Y, Z]$, if $X < Y < Z$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	True	$[X, Y, Z]$
$A[1] < A[2]$	$Y < Z$	True	$[X, Y, Z]$
$A[0] < A[1]$	$X < Y$	Previously True	$[X, Y, Z]$

2. Starting with $A = [X, Y, Z]$, if $X < Z < Y$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	True	$[X, Y, Z]$
$A[1] < A[2]$	$Y < Z$	False so Swap	$[X, Z, Y]$
$A[0] < A[1]$	$X < Z$	True	$[X, Z, Y]$

3. Starting with $A = [X, Y, Z]$, if $Z < X < Y$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	True	$[X, Y, Z]$
$A[1] < A[2]$	$Y < Z$	False so Swap	$[X, Z, Y]$
$A[0] < A[1]$	$X < Z$	False so Swap	$[Z, X, Y]$

4. Starting with $A = [X, Y, Z]$, if $Y < X < Z$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	False so Swap	$[Y, X, Z]$
$A[1] < A[2]$	$X < Z$	True	$[Y, X, Z]$
$A[0] < A[1]$	$Y < X$	Reflexively True	$[Y, X, Z]$

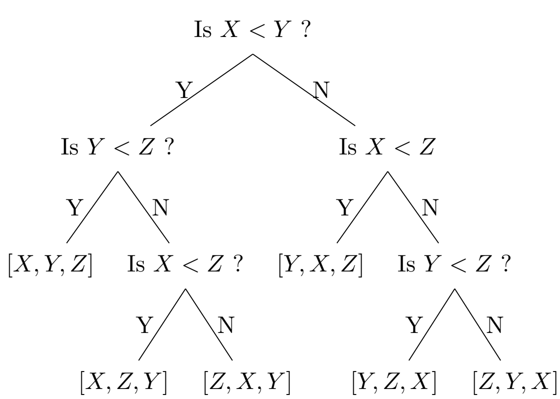
5. Starting with $A = [X, Y, Z]$, if $Y < Z < X$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	False so Swap	$[Y, X, Z]$
$A[1] < A[2]$	$X < Z$	False so Swap	$[Y, Z, X]$
$A[0] < A[1]$	$Y < Z$	True	$[Y, Z, X]$

6. Starting with $A = [X, Y, Z]$, if $Z < Y < X$:

Alg Comparison	List Comparison	Result	A is now
$A[0] < A[1]$	$X < Y$	False so Swap	$[Y, X, Z]$
$A[1] < A[2]$	$X < Z$	False so Swap	$[Y, Z, X]$
$A[0] < A[1]$	$Y < Z$	False so Swap	$[Z, Y, X]$

Here is the corresponding tree:



4 Analysis of Comparison-Based Sorting

We have the following interesting theorem. We really only need the Ω part here but the whole thing is interesting:

Theorem 4.0.1. We have $\Omega(\lg(n!)) = \Omega(n \lg n)$ and $\mathcal{O}(\lg(n!)) = \mathcal{O}(n \lg(n))$ and consequently $\Theta(\lg(n!)) = \Theta(n \lg n!)$.

Proof. We do this in steps:

(a) Suppose that $f(n) = \Omega(\lg(n!))$. We claim that $f(n) = \Omega(n \lg n)$.

Observe that:

$$\begin{aligned}
\lg(n!) &= \lg(n) + \lg(n-1) + ... + \lg(2) + \lg(1) \\
&= [\lg(n-0) + \lg(n-1) + ... + \lg(n - \lfloor n/2 \rfloor)] + [\text{the rest}] \\
&\geq \lg(n-0) + \lg(n-1) + ... + \lg(n - \lfloor n/2 \rfloor) \\
&\geq \underbrace{\lg(n/2) + \lg(n/2) + ... + \lg(n/2)}_{1 + \lfloor n/2 \rfloor \text{ terms}} \\
&\geq (1 + \lfloor n/2 \rfloor)(\lg n - \lg 2) \\
&\geq \frac{1}{2} n(\lg n - 1) \\
&\geq \frac{1}{4} n \lg n \quad \text{If } n \geq 4
\end{aligned}$$

Consequently if $f(n) \geq B \lg(n!)$ then $f(n) \geq \frac{1}{4} B n \lg n$ and the result follows.

(b) Suppose that $f(n) = \Omega(n \lg n)$. We claim that $f(n) = \Omega(n!)$.

Obseve that:

$$\lg(n!) = \lg(n) + \lg(n-1) + ... + \lg(2) + \lg(1) \leq \lg n + \lg n + ... + \lg n + \lg n = n \lg n$$

Consequently if $f(n) \geq B n \lg n$ then $f(n) \geq B \lg(n!)$ and the result follows.

(c) Suppose that $f(n) = \mathcal{O}(\lg(n!))$. We claim that $f(n) = \mathcal{O}(n \lg n)$.

Using the calculation from (b) we see that if $f(n) \leq C \lg(n!)$ then $f(n) \leq C n \lg n$ and the result follows.

(d) Suppose that $f(n) = \mathcal{O}(n \lg n)$. We claim that $f(n) = \mathcal{O}(n!)$.

Using the calculation from (a) we see that if $f(n) \leq C n \lg n$ then $f(n) \leq 4 \lg(n!)$ and the result follows.

$\mathcal{QED}$

Let's look now and how this relates to decision trees. First we have:

Theorem 4.0.2. Any comparison sort requires, in the worst-case, $\Omega(n \lg n)$ (comparison-based) decisions.

Proof. A comparison-based algorithm needs to decide between $n!$ possible permutations of the list and each permutation will be a leaf in the decision tree so the number of leaves must be $n!$.

In general if h is the height of a binary tree then the number of leaves is less than or equal to 2^h (which would be a perfect binary tree) and so we must have:

$$\#\text{ Leaves} = n! \leq 2^h$$

And so:

$$h \geq \lg(n!)$$

Let d be the number of decisions necessary in the worst case. In a worst-case scenario we follow the tree as far down as possible, thus $d = h$ and so:

$$d \geq \lg(n!)$$

Thus by the lemma $d = \Omega(n \lg n)$.

$\mathcal{QED}$

Corollary 4.0.1. Any comparison sort has worst-case time complexity $\Omega(n \lg n)$.

Proof. Each (comparison-based) decision takes constant time and the result follows immediately. $\mathcal{QED}$

Consider what this is saying more specifically. We know that our various (comparison-based) sorting algorithms have best- and worst-cases. Often these are the same, like Bubble Sort ($\Theta(n^2)$) and Merge Sort ($\Theta(n \lg n)$), and sometimes they are not, like Insertion Sort (best-case $\Theta(n)$ and worst-case $\Theta(n^2)$) and Quick Sort (best-case $\Theta(n \lg n)$ and worst-case $\Theta(n^2)$).

When we think of worst-case we think of "one or more (annoying) lists". So for example when we say Merge Sort is worst-case $\Theta(n \lg n)$ we are observing that there are some n_0, B such that for each $n \geq n_0$ there are one or more (annoying) lists which take time at least $Bn \lg n$.

What this theorem is saying is that we can never do better in the worst-case, meaning that if our sorting algorithm is comparison based then there are some n_0, B such that for each $n \geq n_0$ there are one or more (annoying) lists which will take time at least $Bn \lg n$.

5 Thoughts, Problems, Ideas

1. Consider the following modification of BubbleSort:

```
for i = 0 to n-1
    for j = n-2 down to i
        if A[j] <= A[j+1]
            nothing
        else
            swap A[j] and A[j+1]
        end
    end
end
```

   Write down the decision tree for this algorithm as applied to the set $\{X, Y, Z\}$.

2. Consider the following pseudocode for InsertSort:

```
for i = 0 to n-1
    key = A[i]
    j = i-1
    while j >= 0 and key < A[j]
        A[j+1] = A[j]
        j = j - 1
    end
    A[j+1] = key
end
```

   Write down the decision tree for this algorithm as applied to the set $\{X, Y, Z\}$.

3. Consider the following pseudocode for SelectionSort:

```
for i = 0 to n-2
    minindex = i
        for j = i+1 to n-1
        if A[j] < A[minindex]
            minindex = j
        end
    end
    A[i] <-> A[minindex]
end
```

   Write down the decision tree for this algorithm as applied to the set $\{X, Y, Z\}$.