CMSC 351: Limitations on Comparisons

Justin Wyss-Gallifent

March 15, 2022

1 Introduction to Decision Trees . . . . . . . . . . . . . . 2
2 Decisions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
3 Decision Tree for An Algorithm . . . . . . . . . . . . . . . . . 4
4 Analysis of Comparison-Based Sorting . . . . . . . . . . . . . . 6
5 Thoughts, Problems, Ideas . . . . . . . . . . . . . . . . . . . . . 8
1 Introduction to Decision Trees

Note that the fastest sorting algorithms we have seen having a worst-case time complexity $T(n) = \mathcal{O}(n \log n)$. 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: \([Y, Z, X]\)

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 \(Y \neq Z\) 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.

Example 3.1. Consider BubbleSort. This implementation is specifically written so that all comparisons are <=.

```plaintext
for i = 0 to n-1
    for j = 0 to n-i-2
            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...


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 possible ways that the data might be related:

1. If $X < Y < Z$ then starting with $A = [X, Y, Z]$, BubbleSort will check...

2. If $X < Z < Y$ then starting with $A = [X, Y, Z]$, BubbleSort will check...
3. If $Z < X < Y$ then starting with $A = [X, Y, Z]$, BubbleSort will check...

4. If $Y < X < Z$ then starting with $A = [X, Y, Z]$, BubbleSort will check...

5. If $Y < Z < X$ then starting with $A = [X, Y, Z]$, BubbleSort will check...

6. If $Z < Y < X$ then starting with $A = [X, Y, Z]$, BubbleSort will check...

Here is the tree:

![Tree Diagram](attachment:image.png)

Notice that BubbleSort is the same as ImmortalSort!
4 Analysis of Comparison-Based Sorting

Lemma 4.0.1. We have \( \lg(n!) = \Theta(n \lg n) \).

Proof. First we’ll prove \( \Omega \):

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

Note: This last step isn’t obvious but basically:

\[
\frac{1}{2} n (\lg n - 1) \geq \frac{1}{4} n \lg n \iff \frac{1}{4} n \lg n \geq \frac{1}{2} n \iff \lg n \geq 2 \iff n \geq 4
\]

Now we’ll prove \( O \), which is easier.

\[
\lg(n!) = \lg(n) + \lg(n-1) + \ldots + \lg(2) + \lg(1) \leq \lg(n) + \lg(n) + \ldots + \lg(n) = n \lg n
\]

Together this is interesting, for \( n \geq 4 \) we have:

\[
\frac{1}{4} n \lg n \leq \lg(n!) \leq n \lg n
\]

QED

Fun Fact: The above proof also shows that:

\[
\lg(n!) \leq n \lg n \leq 4 \lg(n!)
\]

This means that \( n \lg n \in \Theta(\lg(n!)) \) so in fact \( n \lg n \) and \( \lg(n!) \) are exactly the same time complexity!

Theorem 4.0.1. Any comparison sort requires, in the worst-case, \( \Omega(n \lg n) \) comparisons.

Proof. A comparison-based algorithm needs to manage \( n! \) 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 tree then the number of leaves is at most (less than or equal to) $2^h$ and so we must have:

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

And so:

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

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

All together we get, using the Lemma above:

$$d = h \geq \lg(n!) \geq \frac{1}{4} n \lg n$$

$\text{QED}$

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

*Proof.* If each comparison-based decision takes constant time (which is standard for us) say $c_0$ then we have:

$$T(n) = c_0 d \geq \frac{1}{4} c_0 n \lg n$$

$\text{QED}$

It may seem strange to say that the worst-case is $\Omega(n \lg n)$ but this is meaningful because it’s saying that for large enough $n$ the worst-case can never be better (lower) than some constant $B n \lg n$.

Thus we will never come up with a comparison-based sorting algorithm which has worst case $\mathcal{O}(n)$, for example.
5 Thoughts, Problems, Ideas

1. Consider the following modification of BubbleSort:

   for i = 0 to n - 1
     for j = n - 2 down to i
         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]
       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
         minindex = j
       end
     end
   end

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