1 Introduction to Decision Trees

Note that the fastest sorting algorithms we have seen have 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:

\[
\text{Is } X < Y \ ? \\
\quad \text{Yes: } [X, Y] \\
\quad \text{No: } [Y, X]
\]

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:

\[
\text{Is } X < Y? \\
\quad \text{Yes: Is } Y < Z? \\
\quad \quad \text{Yes: } [X, Y, Z] \\
\quad \quad \text{No: Is } Z < X? \\
\quad \quad \quad \text{Yes: } [Z, X, Y] \\
\quad \quad \quad \text{No: } [X, Z, Y] \\
\quad \text{No: Is } X < Z? \\
\quad \quad \text{Yes: } [Y, X, Z] \\
\quad \quad \text{No: Is } Y < Z? \\
\quad \quad \quad \text{Yes: } [Y, Z, X] \\
\quad \quad \quad \text{No: } [Z, Y, X]
\]

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):

\[
\text{Is } X < Y? \\
\quad \text{Yes: Is } Y < Z? \\
\quad \quad \text{Yes: } [X, Y, Z] \\
\quad \quad \text{No: Is } Z < X? \\
\quad \quad \quad \text{Yes: } [Z, X, Y] \\
\quad \quad \quad \text{No: } [X, Z, Y] \\
\quad \text{No: Is } X < Z? \\
\quad \quad \text{Yes: } [Y, X, Z] \\
\quad \quad \text{No: Is } Y < Z? \\
\quad \quad \quad \text{Yes: } [Y, Z, X] \\
\quad \quad \quad \text{No: } [Z, Y, X]
\]

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:

```plaintext
for i = 0 to n-1
    for j = 0 to n-i-2
            nothing
        else
        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 compare the first two elements, swapping if necessary, then the second two elements, swapping if necessary, and then the first two elements again, swapping if necessary.

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:

- If $X < Y < Z$ then starting with $A = [X, Y, Z]$, BubbleSort will decide if $X < Y$ (true), decide if $Y < Z$ (true), decide if $X < Y$ again (automatically true, no decision).

- If $X < Z < Y$ then starting with $A = [X, Y, Z]$, BubbleSort will decide if $X < Y$ (true), decide if $Y < Z$ (false) so swap to $A = [X, Z, Y]$, decide if $X < Z$ (true).


- If $Y < X < Z$ then starting with $A = [X, Y, Z]$, BubbleSort will decide if $X < Y$ (false) so swap to $A = [Y, X, Z]$, decide if $X < Z$ (true), decide if $Y < X$ (automatically true, no decision).

If \( Z < Y < X \) then starting with \( A = [X, Y, Z] \), BubbleSort will decide if \( X < Y \) (false) so swap to \( A = [Y, X, Z] \), decide if \( X < Z \) (false) so swap to \( A = [Y, Z, X] \), decide if \( Y < Z \) (false) so swap to \( A = [Z, Y, X] \).

Here is the tree:

```
Is X < Y ?

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

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

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

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

Proof. Observe that:

\[
\begin{align*}
\lg(n!) &= \lg(n) + \lg(n-1) + \ldots + \lg(2) + \lg(1) \\
&= [\lg(n - 0) + \lg(n - 1) + \ldots + \lg(n - \lfloor n/2 \rfloor)] + \text{[the rest]} \\
&\geq \lg(n - 0) + \lg(n - 1) + \ldots + \lg(n - \lfloor n/2 \rfloor) \\
&\geq \lg(n/2) + \lg(n/2) + \ldots + \lg(n/2) \\
&\geq (1 + \lfloor n/2 \rfloor)(\lg n - \lg 2) \\
&\geq \frac{n}{2}(\lg n - 1) \\
&\geq \frac{n}{4}\lg n \quad \text{If } n \geq 4
\end{align*}
\]

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

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

QED

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

Proof. The height $h$ of the corresponding decision tree indicates how many decisions must be made in the worst-case situation for a particular algorithm. The number of comparisons $c$ satisfies $c \geq h$ as discussed above.

A comparison-based algorithm needs to manage $n!$ permutations each of which needs to be a leaf in the decision tree. Since a tree of height $h$ has at most $2^h$ leaves and thus we must have $2^h \geq n!$.

From here we get $c \geq h \geq \lg(n!) = \Omega(n \lg n)$

QED

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

Proof. If each comparison takes constant time (which is standard for us) and if the number of comparisons is $\Omega(n \lg n)$ then the time requirement, which is at least that much, must also be $\Omega(n \lg n)$.

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 $Bn \lg n$. 

6
5 Thoughts, Problems, Ideas

1. Consider the following modification of BubbleSort:

```plaintext
for i = 0 to n-1
    for j = n-2 down to i
            nothing
        else
        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:

```plaintext
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:

```plaintext
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\}.