CSC 263 Lecture 13

December 5, 2006

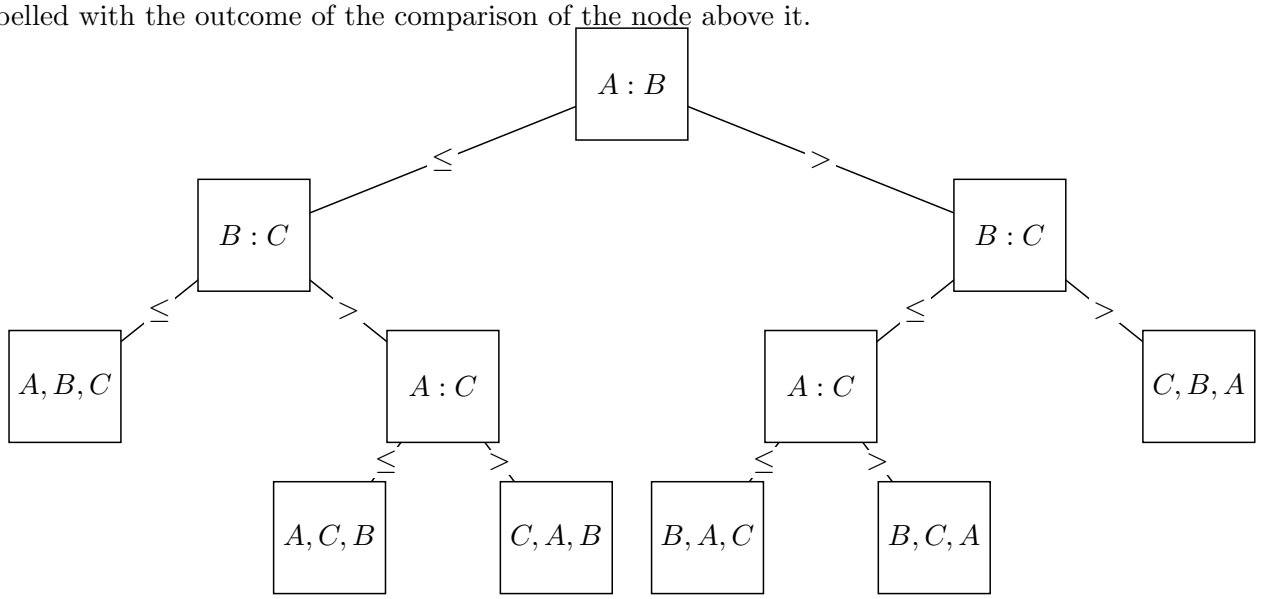
21 Lower bounds

Definition. *A* comparison-based *algorithm is an algorithm where the behaviour of the algorithm is based only on the* comparisons *between elements.*

In a comparison-based sort, we only use comparisons between elements to gain information about the order of a sequence. Given two elements a_i and a_j, we perform one of the tests: $a_i < a_j$, $a_i <= a_j$, $a_i = a_j$, $a_i => a_j$, or $a_i > a_j$. We can say that each test returns one of the following outcomes: ($\leq$,>), (<,$\geq$), ($\leq$,$\geq$), (<, =, >), (=, $\neq$)

We can express a comparison sort as a *decision tree.*

Example
Let's look at a particular decision tree for sorting the set $\{A, B, C\}$. Internal nodes represent comparisons of the two elements in the node. Leaf nodes represent the result of the sort. Each edge is labelled with the outcome of the comparison of the node above it.



Note that this is only a particular decision tree for this sort. There are other possibilities. The decision tree depends on the algorithm that we are using. Observe that this decision tree has 6 leaves and every permutation of the elements occurs as some leaf node.

The length of the longest path from the root of the decision tree to a leaf represents the worst-case number of comparisons that the sorting algorithm performs. Therefore, worst-case number of comparisons is the height of the decision tree.

How can we find the algorithm with the smallest worst-case number of comparisons? We find the decision tree with the smallest height.

In other words, to find the worst-case running time of the "best" algorithm (i.e., the one with the smallest worst-case number of comparisons), we want to find a *lower bound* on the height of the decision trees.

Some useful facts:

1. The number of ways to sort n numbers is $n!$ (each permutation of the numbers is possible). This implies that the number of leaves in the decision tree is at least $n!$ (there may be duplicate leaves).

2. There are at most 3 outcomes (branches) at any level of the tree: ($<$, $=$, $>$). Recall the other possible outcomes are: ($\leq$,$>$), ($<$,$\geq$), ($\leq$,$\geq$), ($=$, $\neq$).

3. A ternary tree of height h has at most 3^h leaves. Alternatively, a tree with at least 3^h leaves must have height at least h.

Thus, we can conclude that the decision tree for sorting has height at least $\log_3 n!$.

Since $\log_3 n! \in \Theta(n \log n)$, then $h \in \Omega(n \log n)$. This implies that the worst-case number of comparisons for any comparison-based sorting algorithm is in $\Omega(n \log n)$.

21.1 Showing that $\log_3 n! \in \Theta(n \log n)$

First, recall that $\log_2 n! \in \Theta(n \log_2 n)$ due to Stirling's Approximation. It is then easy to show that $\log_3 n! \in \Theta(n \log_3 n)$.

Then,

$$\begin{array}{rcll} k & = & \log_3 f(n) & \\ 3^k & = & f(n) & \text{(take both sides as the exponent of 3)} \\ k \log_2 3 & = & \log_2 f(n) & \text{(take the } \log_2 \text{ of both sides)} \\ \log_3 f(n) & = & \log_2 f(n) / \log_2 3 & \text{(replace } k \text{ with } \log_3 f(n), \text{ and divide both sides by } \log_2 3) \end{array}$$

Therefore $\log_3 n! \in \Theta(\log_2 n!)$ and $\log_3 n! \in \Theta(n \log n)$.

21.2 Another example

Let's find a lower bound on the worst-case time needed to compute the following problem:

Is there an element that appears exactly 5 times and another element that appears exactly 7 times in an n-element array of integers?

Consider the following algorithm:

Step 1 Sort all the elements.

Step 2 Scan through the elements and count whether one element appears 5 times and another appears 7 times.

The total worst-case time of this algorithm is in $O(n \log n)$ So, either the lower bound on the worst-case time needed is $\Omega(n \log n)$ or it is something smaller.

Let's compute the lower bound using the decision tree method. Consider the following:

1. There are two possible solutions to this problem: Yes and No. This implies that the number of leaves in the decision tree is at least 2 (there may be duplicate leaves).

2. There are at most 3 outcomes (branches) at any level of the tree: ($<$, $=$, $>$)

We can conclude that the decision tree for sorting has height at least $\log_3 2$ Since $\log_3 2$ in $\Omega(1)$, we have a lower bound of $\Omega(1)$.

But, our lower and upper bounds don't match, and there doesn't seem to be a constant time algorithm that solves this problem. What do we do?

21.2.1 The adversary method

This method pits our algorithm against an adversary. Our algorithm asks a question and then tries to reduce choices as quickly as possible (i.e. tries to learn the input). The adversary answers questions such that at least one input is consistent with the answers and tries to keep the legal set of inputs as big as possible (i.e. it tries to make you do as much work as possible).

Example
The game "Guess Who?" This is a game where you try to determine a particular person by asking questions about them. I always answer correctly but keep the answer so that as many people as possible are still available (because I haven't chosen the answer yet). You are our algorithm and I am the adversary.

We will use an adversary argument to solve the problem above. Suppose the algorithm only take n-1 steps. Then, it only reads n-1 spots in the array. The adversary let the algorithm read the following input: x x x x x y y y y y y y . . . . The algorithm is correct, and saw 5 x's and 7 y's so it answers 'yes', but it did not see one of the input values.

We have the following cases:

1. The algorithm did not see a '.' value If the adversary changes the input so the '.' becomes an 'x', we have: x x x x x y y y y y y y . . x . The algorithm behaves the same way, and answers incorrectly!

2. The algorithm did not see a 'x'.

3. The algorithm did not see a 'y'.

For case each, we can change the input so that the algorithm answers incorrectly!

Therefore, we conclude that the algorithm needs to take at least n steps. This implies that a lower bound for the worst-case time for ANY algorithm to solve this problem is in $\Omega(n)$. This is better than the $\Omega(1)$ lower bound from the decision tree method.

Exercise. *Can we find an algorithm that takes linear time or is our lower bound still too low?*

21.3 Lower bounds for comparison based search

21.3.1 Searching a sorted array

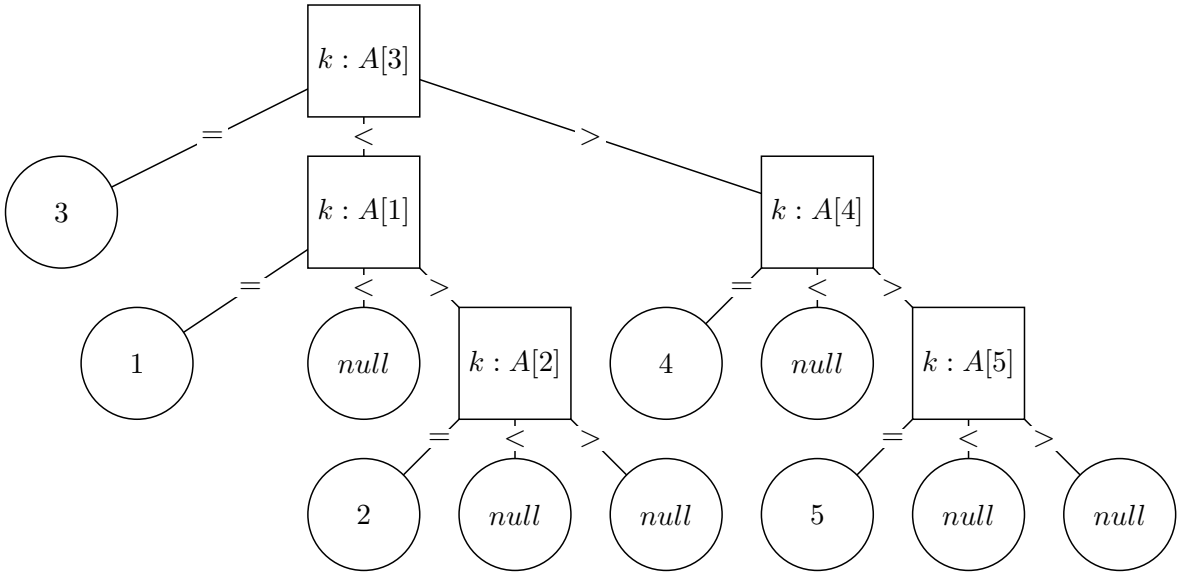
Example
The algorithm for binary search is as follows (where we return null when the key k is not found):

```
BinarySearch(A, k)
  low = 1
  high = size(A)

  repeat until (high < low)
    mid = floor((high-low)/2) + low
    if A[mid] = k
      return mid
    else if A[mid] < k
      low = mid + 1
    else if A[mid] > k
      high = mid -1

  return null
```

The decision tree is as follows:



Some observations:

1. Some leaves are duplicates (i.e. the ones that return null).
2. There are $n + 1$ unique leaves (we can return each of the 5 keys, and null if the value is not in the array).

3. Each non-leaf node has three children.

For *any* comparison-based search algorithm A, we can prove a lower bound of $\Omega(logn)$ as follows:

- The number of distinct outcomes is n+1 in the worst-case (i.e. when all values in the array are unique).
- The corresponding decision tree T_A has at least $n + 1$ leafs.
- There are at most three outcomes at any tree level (=, <, >).
- The decision tree has height at least $\log_3 n$ (since a ternary tree of height h has at most 3^h leaves).
- Since $\log_3 n \in \Omega(\log n)$, we have a lower bound of $\Omega(\log n)$.

21.3.2 Searching an unsorted array

Sometimes the decision tree technique to determine a lower bound is not powerful enough to give a good lower bound. In particular, consider any algorithm for searching on an unsorted array.

- The decision tree has $n + 1$ distinct leafs
- The tree has 2 outcomes at each level (=, $\neq$).
- The tree has height $\log_2 n$.
- We have a lower bound of $\Omega(\log n)$.

But, our lower bound is too low! Search on an unsorted array actually has a lower bound of $\Omega(n)$. Intuitively, if we only look at $n - 1$ elements, we can't be sure if an element is not in the array or if it is the element we haven' looked at.

Exercise. *Why is there a discrepancy between the decision tree lower bound and our intuitive understanding of the lower bound?*