Lower and Upper Bound Theory

Adnan YAZICI
Dept. of Computer Engineering
Middle East Technical Univ.
Ankara - TURKEY

Lower and Upper Bound Theory

How fast can we sort?

- Most of the sorting algorithms are **comparison sorts**: only use comparisons to determine the relative order of elements.
  - E.g., insertion sort, merge sort, quicksort, heapsort.
- The best worst-case running time that we've seen for comparison sorting is O(nlgn) .

  Is O(nlgn) the best we can do?

Lower-Bound Theory can help us answer this question.

Lower and Upper Bound Theory

- *Lower Bound*, L(n), is a property of the specific problem, i.e., sorting problem, MST, matrix multiplication, not of any particular algorithm solving that problem.
- *Lower bound theory* says that no algorithm can do the job in fewer than L(n) time units for arbitrary inputs, i.e., that every comparison-based sorting algorithm must take at least L(n) time in the worst case.
- L(n) is the minimum over all possible algorithms, of the maximum complexity.

Lower and Upper Bound Theory

- *Upper bound theory* says that for any arbitrary inputs, we can always sort in time at most U(n). How long it would take to solve a problem using one of the known algorithms with worst-case input gives us a *upper bound*.
- Improving an *upper bound* means finding an algorithm with better worst-case performance.
- U(n) is the minimum over all known algorithms, of the maximum complexity.
- Both upper and lower bounds are *minima* over the *maximum complexity* of inputs of size n.
- The ultimate goal is to make these two functions coincide. When this is done, the optimal algorithm will have L(n) = U(n).

Lower and Upper Bound Theory

There are few techniques for finding lower bounds.

Trivial Lower Bounds: For many problems it is possible to easily observe that a lower bound identical to n exists, where n is the number of inputs (or possibly outputs) to the problem.

- The method consists of simply counting the number of inputs that must be examined and the number of outputs that must be produced, and
- note that any algorithm must, at least, read its inputs and write its outputs.

Example-1: Multiplication of a pair of nxn matrices

- requires that $2n^2$ inputs be examined and
- n^2 outputs be computed, and
- the lower bound for this matrix multiplication problem is therefore $\Omega(n^2)$.

Lower and Upper Bound Theory

Example-2: Finding maximum of unordered array requires examining each input so it is $\Omega(n)$.

- A simple counting arguments shows that any comparison-based algorithm for finding the maximum value of an element in a list of size n must perform at least n-1 comparisons for any input.

- **Other Examples**?

Lower and Upper Bound Theory

Information Theory: The information theory method establishing lower bounds by computing the limitations on information gained by a basic operation and then showing how much information is required before a given problem is solved.

- This is used to show that any possible algorithm for solving a problem must do some minimal amount of work.
- The most useful principle of this kind is that the outcome of a comparison between two items contains one bit of information.

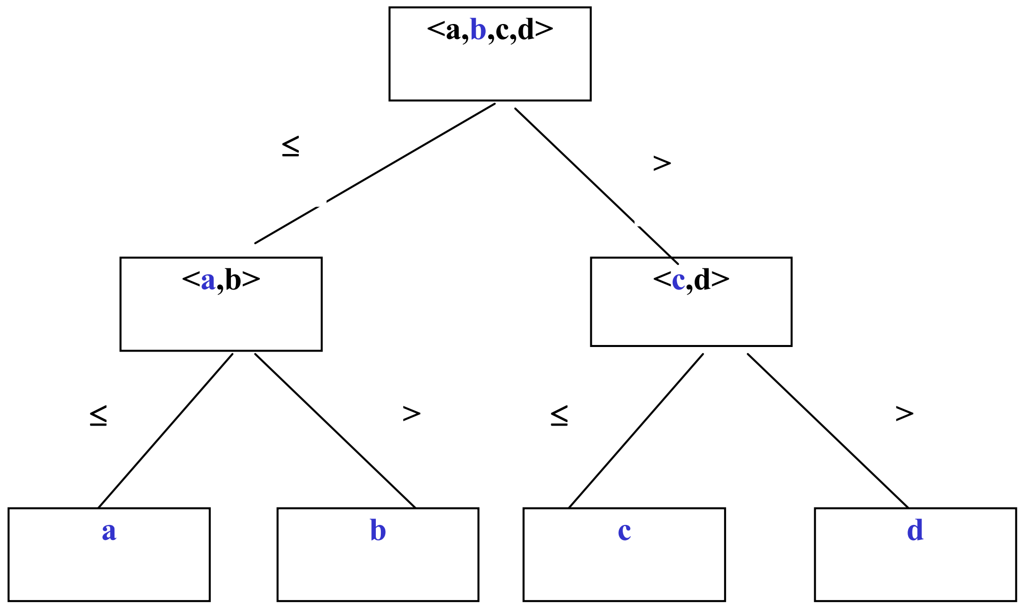
Lower and Upper Bound Theory

Example-1: For the problem of *searching an ordered list* with n elements for the position of a particular item,

Proof: There are n possible outcomes, input strings

- In this case $\lceil \lg n \rceil$ comparisons are necessary,
- So, unique identification of an index in the list requires $\lceil \lg n \rceil$ bits.
- Therefore, $\lceil \lg n \rceil$ bits are necessary to specify one of the m possibilities.

Lower and Upper Bound Theory



2 bits of information
is necessary.

Example-2: For the problem of *comparison-based sorting*:

- If we only know that the input is orderable then there are n! possible outcomes – each of the n! permutations of n things.
- Since, within the comparison-swap model, we can only use comparisons to derive information
- Then, from information theory, $\lceil \lg n! \rceil$ is a lower bound on the number of comparisons necessary in the worst-case to sort n things.

Lower and Upper Bound Theory

Proof: For the problem of comparison-based sorting,

- The result of a given comparison between two list of elements yields a single bit of information (0=False, 1 = True).
- Each of the n! permutations of {1, 2, …, n} has to be distinguished by the correct algorithm.
- Thus, a comparison-based algorithm must perform enough comparisons to produce n! cumulative pieces of information.
- Since each comparison only yields one bit of information, the question is what the minimum number of bits of information needed to allow n! different outcomes is, which is $\lceil \lg n! \rceil$ bits.

Lower and Upper Bound Theory

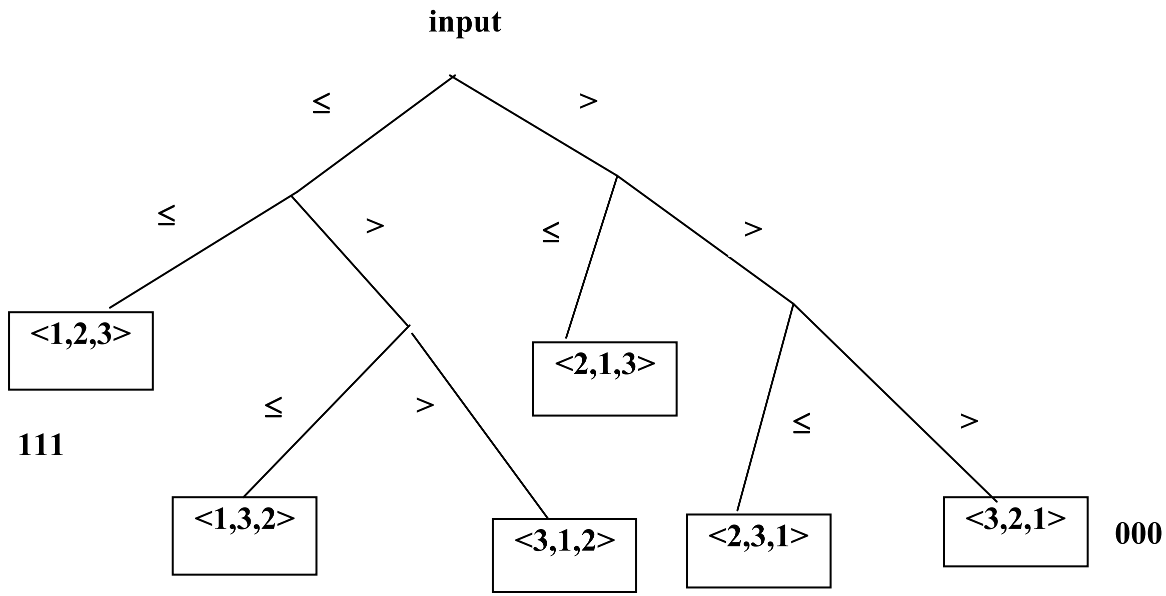


Figure: There are 3! = 6 possible permutations of the n input elements, so $\lceil \lg n! \rceil$ bits are required for $\lceil \lg n! \rceil$ comparisons for sorting n things, which is $\Theta(n\lg n)$.

Lower and Upper Bound Theory

How fast lgn! grow? We can bind n! from above by overestimating every term of the product, and bind it below by underestimating the first n/2 terms.

$$n/2 \times n/2 \times \ldots \times n/2 \times \ldots \times 2 \times 1$$
$$\leq n! = n \times (n-1) \times \ldots \times 2 \times 1$$
$$\leq n \times n \times \ldots \times n$$

$$(n/2)^{n/2} \leq n! \leq n^n$$
$$\tfrac{1}{2}(n\lg n - n) \leq \lg n! \leq n\lg n$$

This follows that $\lg n! \in \Theta(n\lg n)$

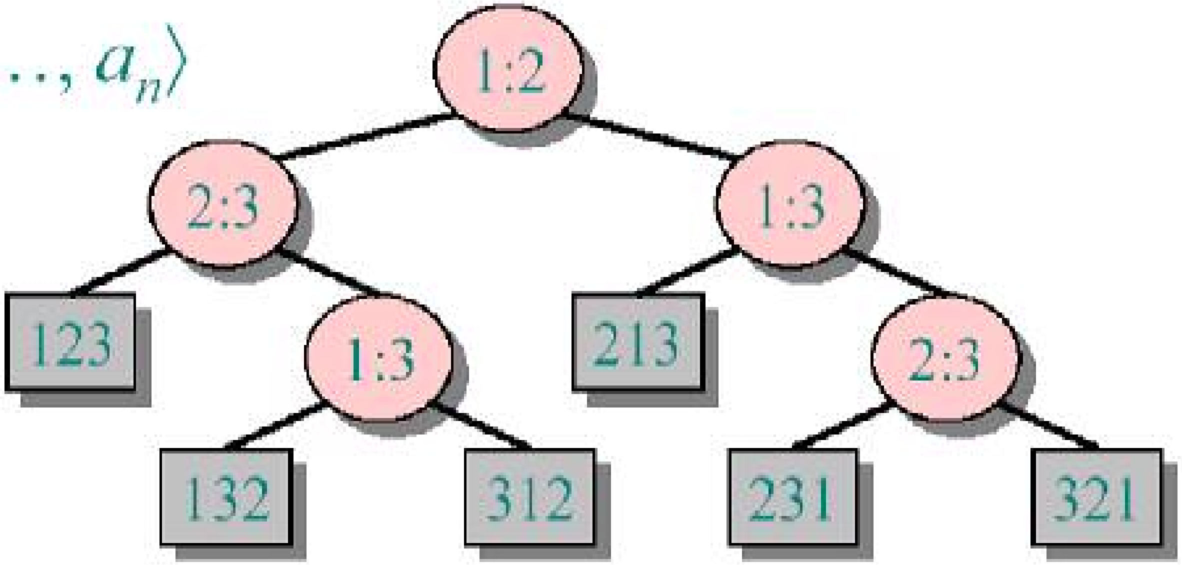
Decision-tree model

A decision tree can model the execution of any comparison based problem.

- One tree for each input size *n*.
- View the algorithm as splitting whenever it compares two elements.
- The tree contains the comparisons along all possible instruction traces.
- The running time of the algorithm = the length of the path taken.
- Worst-case running time = the height of tree.

Decision-tree example

Sort $\langle a_1, a_2, \ldots, a_n \rangle$

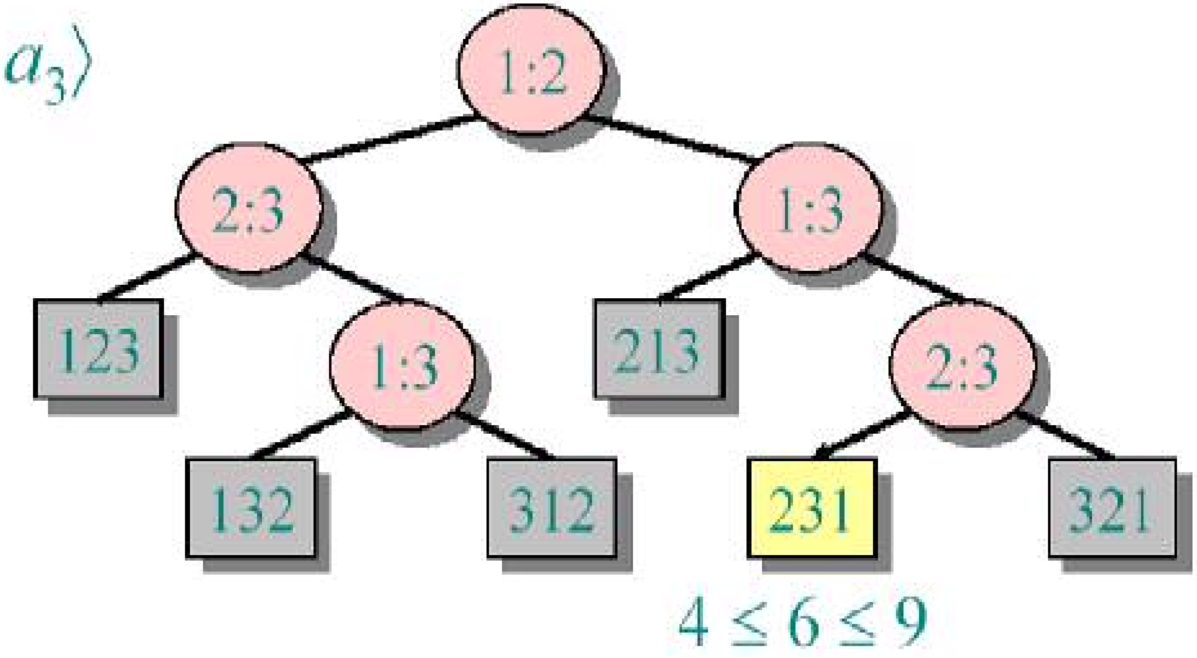


Each internal node is labeled $i{:}j$ for $i, j \in \{1, 2, \ldots, n\}$.

- The left subtree shows subsequent comparisons if $a_i \le a_j$.
- The right subtree shows subsequent comparisons if $a_i \ge a_j$.

Decision-tree model

Sort $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:



Each leaf contains a permutation $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ to indicate that the ordering $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$ has been established.

Lower bound for decision-tree sorting

Theorem. Any decision tree that can sort n elements must have height $\Omega(n \lg n)$.

Proof. The tree must contain $\geq n!$ leaves, since there are $n!$ possible permutations. A height-h binary tree has $\leq 2^h$ leaves. Thus, $n! \leq 2^h$.

$$
\begin{aligned}
\therefore\ h &\geq \lg(n!) && (\lg \text{ is mono. increasing}) \\
&\geq \lg((n/e)^n) && (\text{Stirling's formula}) \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n). \quad \square
\end{aligned}
$$

Decision-tree model for Searching

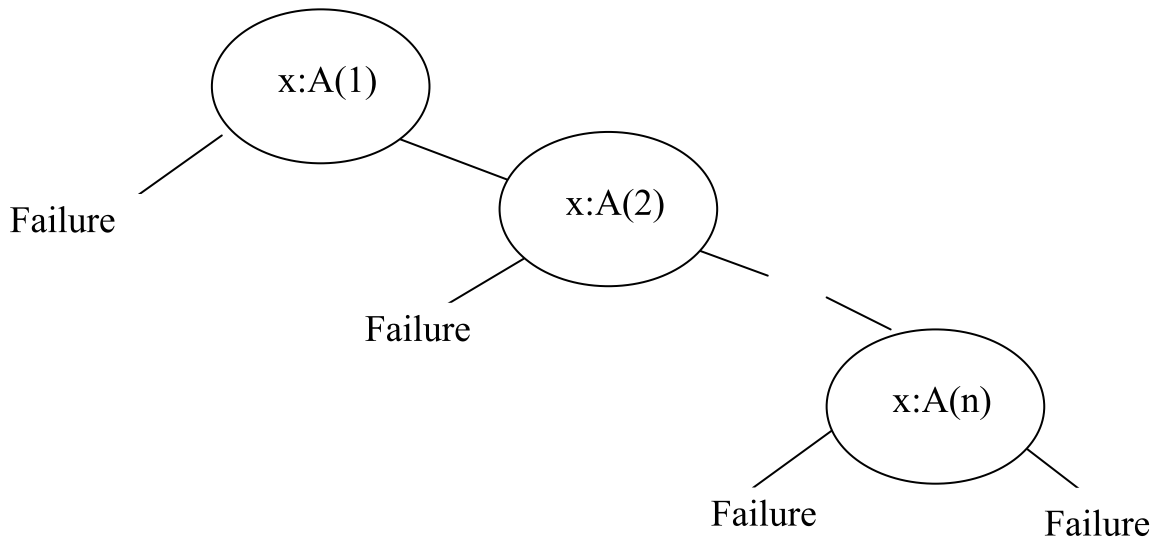


Figure: A comparison tree for a linear search algorithm

Decision-tree model for Searching

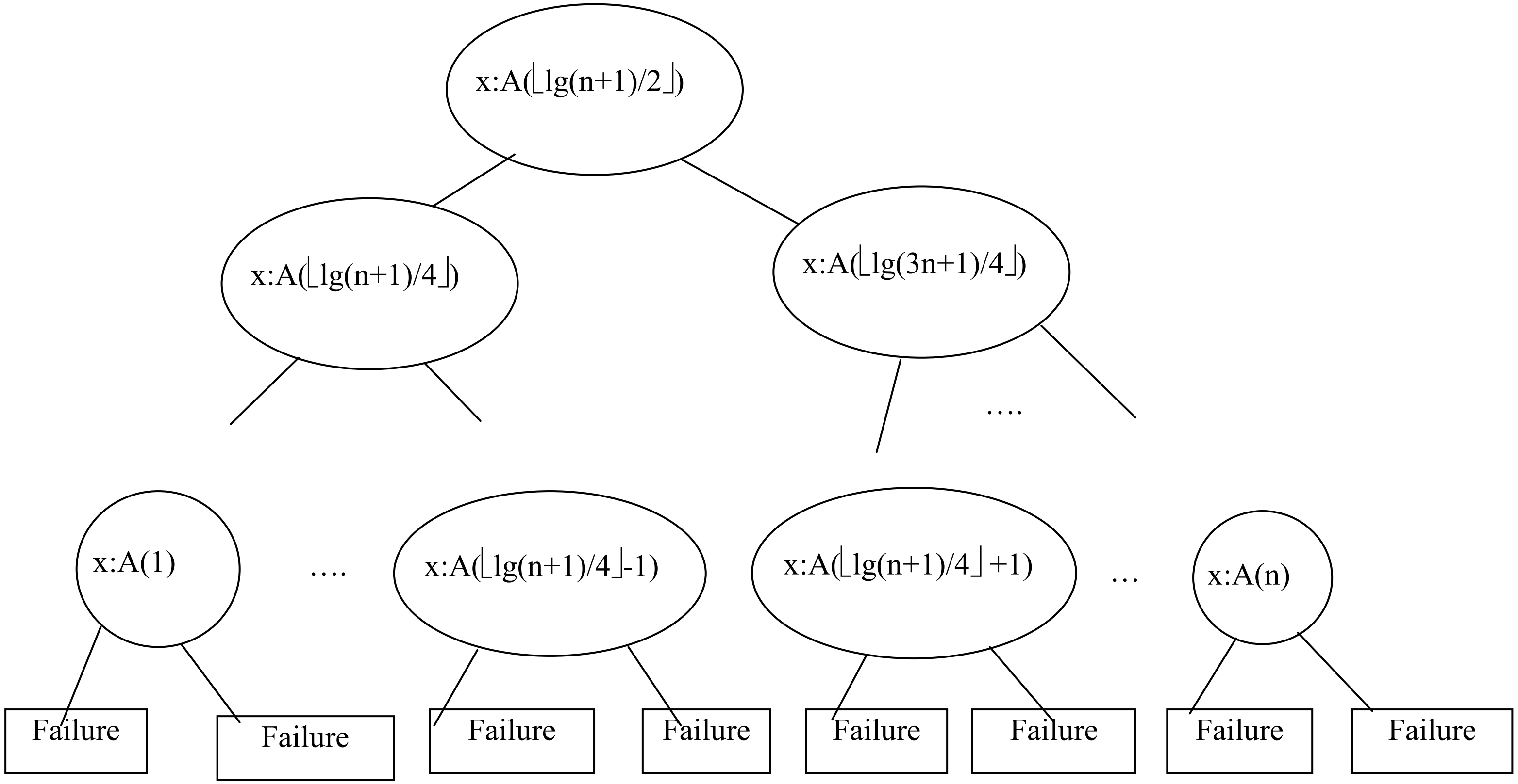


Figure: A Comparison tree for a search algorithm

Decision-tree model

Example: (*Lower bound for comparison-based searching on ordered input):*

Let A(1:n), $n \geq 1$, contain n distinct elements, ordered so that $A(1) < A(2) < \ldots < A(n)$. Let FIND(n) be minimum number of comparisons needed, in the worst case, by any comparison based algorithm to recognize if $x \in A(1:n)$. Then FIND(n) $\geq \lceil \lg(n+1) \rceil$.

Decision-tree model

Proof: Let us consider all possible comparison trees which model algorithms to solve the searching problem.

• FIND(n) is bounded below by the distance of the longest path from the root to a leaf in such a tree.

• There must be n internal nodes in all of these trees corresponding to the n possible successful occurrences of x in A.

• If all internal nodes of binary tree are at levels less than or equal to k (every height k-rooted binary tree has at most $2^{k+1} - 1$ nodes), then there are at most $2^k - 1$ internal nodes.

Thus, $n \leq 2^k - 1$ and $\text{FIND}(n) = k \geq \lceil \log (n+1) \rceil$.

• Because every leaf in a valid decision tree must be reachable, the worst-case number of comparisons done by such a tree is the number of nodes in the longest path from the root to a leaf in the binary tree consisting of the comparison nodes.

Oracles and Adversary Arguments

- Another technique for obtaining lower bounds consists of making use of an “oracle.”
- Given some model of computation such as comparison trees, the oracle tells us the outcome of each comparison.
- In order to derive a good lower bound, the oracle tries its best to cause the algorithm to work as hard as it might.
- It does this by choosing as the outcome of the next test, the result which causes the most work to be required to determine the final answer.
- And by keeping track of the work that is done, a worst-case lower bound for the problem can be derived.

Oracles and Adversary Arguments

Example: (*Merging Problem*) Given the sets A(1:m) and B(1:n), where the items in A and in B are sorted. Investigate lower bounds for algorithms merging these two sets to give a single sorted set.

•Assume that all of the m+n elements are distinct and $A(1) < A(2) < \ldots < A(m)$ and $B(1) < B(2) < \ldots < B(n)$.

•Elementary combinatorics tells us that there are C((m+n), n)) ways that the A's and B's may merge together while still preserving the ordering within A and B.

•Thus, if we use comparison trees as our model for merging algorithms, then there will be C((m+n), n)) external nodes and therefore at least $\lceil \log C((m+n), m) \rceil$ comparisons are required by any comparison-based merging algorithm.

•If we let MERGE(m,n) be the minimum number of comparisons need to merge m items with n items then we have the inequality

$$\lceil \log C((m+n), m) \rceil \leq MERGE(m,n) \leq m+n-1.$$

•The upper bound and lower bound can get arbitrarily far apart as m gets much smaller than n.

Problem Reduction

Another elegant means of proving a lower bound on a problem is to show that an algorithm for solving that a problem along with a transformation on problem instances, could be used to construct an algorithm to solve another problem for which a lower bound is known.

Problem Reduction

Example: An algorithm for finding the *Euclidean minimum spanning tree* of n points in the plane can be used to solve the *element uniqueness problem*, and must therefore take time $\Omega(n\lg n)$.

The reduction is quite simple:

- Suppose we want to determine whether any two of the numbers $x_1, x_2, \ldots, x_n$ are equal.
- We can solve this problem by giving any Euclidean MST algorithm the points $(x_1,0), (x_2,0), \ldots, (x_n,0)$.
- The two closest points are known to be joined by one of the (n-1) spanning-tree edges,
- So we can scan these edges in linear time, determine if any edge has zero length.
- Such an edge exists iff two of x_i are equal.
- Therefore, if the *spanning tree algorithm* could operate in less than O(nlgn) time, then the *element uniqueness problem* could be solved in less than O(nlgn) time too.