Contents

1. LB by adversary arguments
2. LB by decision trees

Section outline

1. **LB by adversary arguments**
   - LB for finding duplicates in a sorted list by comparisons

LB for finding duplicates in a sorted list is $(n-1)$ by comparisons

1. Assume there exists an algorithm $\mathcal{A}$ which runs in $(n-2)$ comparisons which correctly finds duplicates in an ordered list of size n
2. Let X be a list such that $x_i = 2i$ for $i = 1$ to n, where all elements are unique.
3. Run algorithm $\mathcal{A}$ on input X; since it only takes $(n-2)$ comparisons, there is at least 1 element which is not compared to its next element
4. Find that element $x_i = 2i$ and set $x_{i+1} = 2i$; note that previously, $x_{i+1} = 2(i+1)$
5. Rerun the algorithm; it will report no duplicates, as it did before, but wrongly this time

Thus, using a comparison based scheme it is not possible to find duplicates in a sorted list in less than $(n-1)$ comparisons

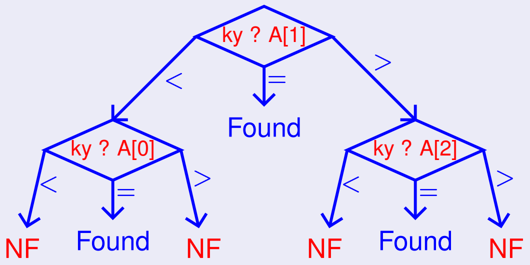
Section outline

2 LB by decision trees

- Lower bound for searching by comparison
- Lower bound for sorting by comparison

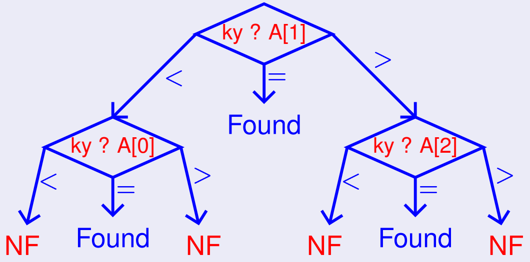
Lower bound for searching by comparison

Decision tree for binary searching in an array of three elements



Lower bound for searching by comparison

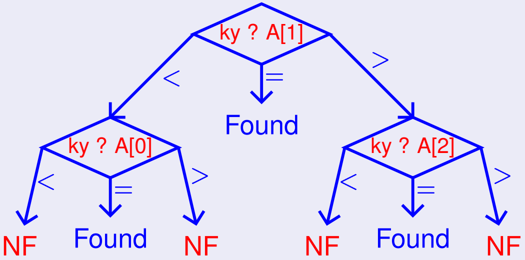
Decision tree for binary searching in an array of three elements



- Decision tree must have one node for comparing **ky** to each **A[i]**
- An internal node produces at most two non-leaf nodes
- At most 2^k comparison nodes at level k, no comparisons at last level

Lower bound for searching by comparison

Decision tree for binary searching in an array of three elements

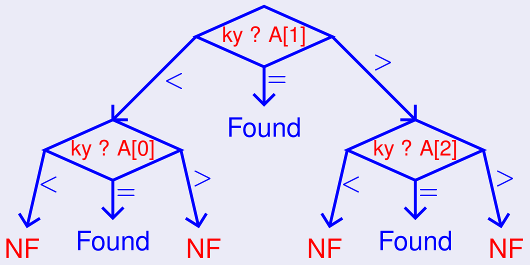


- Decision tree must have one node for comparing **`ky`** to each **`A[i]`**
- An internal node produces at most two non-leaf nodes
- At most 2^k comparison nodes at level k, no comparisons at last level
- Max comparisons in k levels: $\sum_{0}^{k-1} 2^i = 2^k - 1$

- Min n comparisons needed: $2^k - 1 \geq n \Rightarrow k \geq \lg(n+1) \Rightarrow k \in \Omega(\lg n)$

Lower bound for searching by comparison

Decision tree for binary searching in an array of three elements

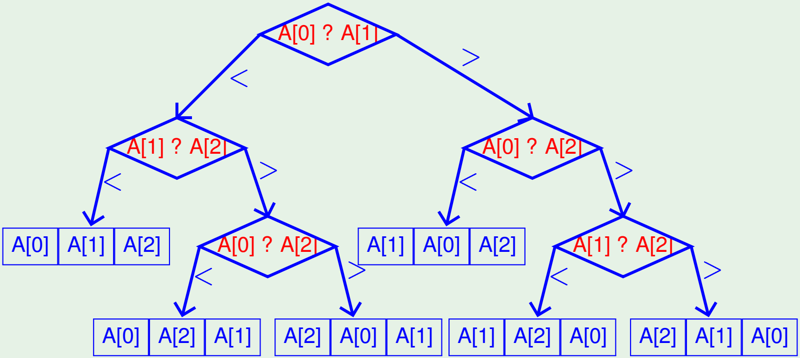


- Decision tree must have one node for comparing **ky** to each **A[i]**
- An internal node produces at most two non-leaf nodes
- At most 2^k comparison nodes at level k, no comparisons at last level
- Max comparisons in k levels: $\sum_{0}^{k-1} 2^i = 2^k - 1$

- Min n comparisons needed: $2^k - 1 \geq n \Rightarrow k \geq \lg(n+1) \Rightarrow k \in \Omega(\lg n)$
- Binary search achieves this lower bound and so is asymptotically optimal

Lower bound for sorting by comparison

Example (A decision tree of comparison based sorting for three distinct elements)



Comparison based sorting algorithms

- A sorting algorithm is comparison based if the comparisons A[i] $<$ A[j], A[i] $\leq$ A[j], A[i] $=$ A[j], A[i] $\geq$ A[j], A[i] $>$ A[j] are the only ways in which it decides on the action on the input elements

Comparison based sorting algorithms

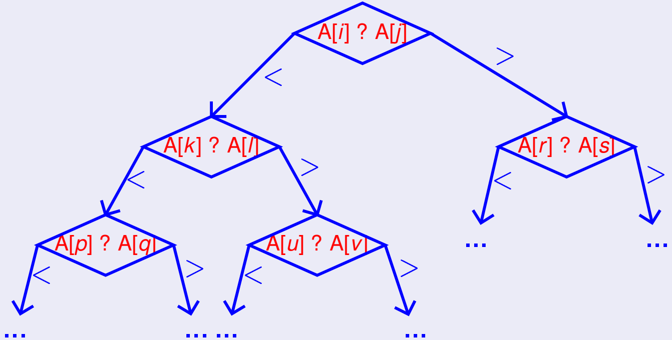
- A sorting algorithm is comparison based if the comparisons A[i] $<$ A[j], A[i] $\leq$ A[j], A[i] $=$ A[j], A[i] $\geq$ A[j], A[i] $>$ A[j] are the only ways in which it decides on the action on the input elements
- Bubble sort, selection sort, insertion sort, quick sort, merge sort, heap sort (to be covered) are all comparison based

Comparison based sorting algorithms

- A sorting algorithm is comparison based if the comparisons A[i] $<$ A[j], A[i] $\leq$ A[j], A[i] $=$ A[j], A[i] $\geq$ A[j], A[i] $>$ A[j] are the only ways in which it decides on the action on the input elements
- Bubble sort, selection sort, insertion sort, quick sort, merge sort, heap sort (to be covered) are all comparison based
- What about counting sort and radix sort?

Decision tree for sorting by comparison

Decision tree for binary searching in an array of distinct elements



- Possible traces of a comparison based sorting algorithm can be captured by a decision tree
- Each node has three outcomes ($<$, $=$, $>$) in general, but two if the keys are distinct (a restricted case)

LB for comparison based sorting is $n \lg n$

- Each leaf is a permutation of elements based on the decision sequence
- n! permutations *must* be covered

LB for comparison based sorting is $n \lg n$

- Each leaf is a permutation of elements based on the decision sequence
- $n!$ permutations *must* be covered
- Let the binary decision tree be of height h
- It will have at most 2^h leaves – terminal decisions
- These terminal decisions must cover at least the $n!$ possibilities
- Thus, $n! \leq 2^h \Rightarrow h \geq \lg(n!)$

LB for comparison based sorting is $n \lg n$

- Each leaf is a permutation of elements based on the decision sequence
- $n!$ permutations *must* be covered
- Let the binary decision tree be of height h
- It will have at most 2^h leaves – terminal decisions
- These terminal decisions must cover at least the $n!$ possibilities
- Thus, $n! \leq 2^h \Rightarrow h \geq \lg(n!)$
- Now $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$, therefore,
  $\lg(n!) \geq \lg\left(\frac{n}{2}^{\frac{n}{2}}\right) = \frac{n}{2}\lg\left(\frac{n}{2}\right) = \frac{n}{2}(\lg n - \lg 2)$
- Thus, $h \geq \lg n! \geq \frac{n}{2}(\lg n - \lg 2) \Rightarrow$ $h \in \Omega(n \lg n)$

LB for comparison based sorting is $n \lg n$

- Each leaf is a permutation of elements based on the decision sequence
- $n!$ permutations *must* be covered
- Let the binary decision tree be of height h
- It will have at most 2^h leaves – terminal decisions
- These terminal decisions must cover at least the $n!$ possibilities
- Thus, $n! \leq 2^h \Rightarrow h \geq \lg(n!)$
- Now $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$, therefore,
  $\lg(n!) \geq \lg\left(\frac{n}{2}^{\frac{n}{2}}\right) = \frac{n}{2}\lg\left(\frac{n}{2}\right) = \frac{n}{2}(\lg n - \lg 2)$
- Thus, $h \geq \lg n! \geq \frac{n}{2}(\lg n - \lg 2) \Rightarrow$ $h \in \Omega(n \lg n)$
- No loss of generality in considering the special case of all distinct elements, as this case *must* be covered by the sorting technique