

Linear-Time Sorting:

- Comparison model
- Lower bounds
 - Searching: $\Omega(\lg n)$
 - Sorting: $\Omega(n \lg n)$

$O(n)$ [with some restrictions] Sorting } different model of computation (RAM models)

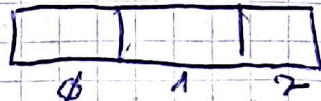
Algorithms:

- Counting Sort
- Radix Sort

Comparison Model:

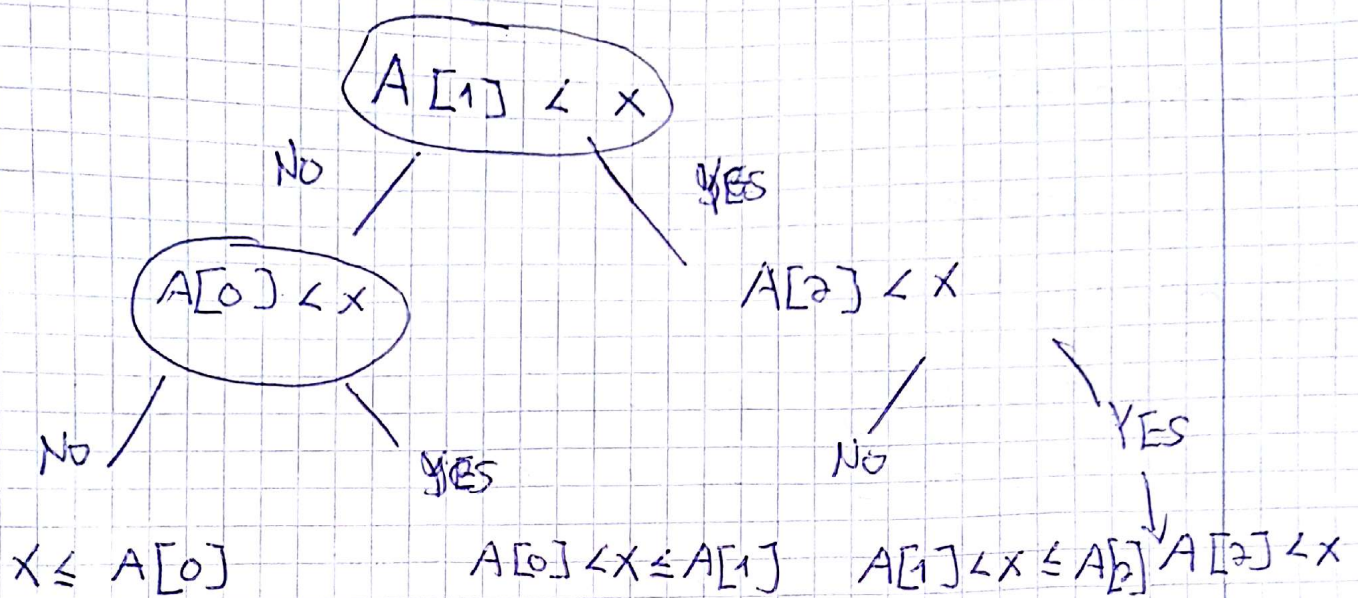
- all input items are black boxes (ADTs)
- only ^{operations} ~~allowed~~ allowed are comparisons ($<, >, \leq, \geq, =$)
- time cost = # comparisons

Notion of decision tree: any comparison algorithm can be viewed as tree of all possible ~~outcomes~~ comparison and their outcomes, and resulting answer.

Binary Search: A 

e.g. Binary Search for $n=3$

→ next page



We don't actually want to represent an algorithm like this unless we want to just trying to analyze it, or prove time complexity

<u>decision tree</u>	<u>algorithm</u>
internal node	binary decision
leaf	found answer
root-to-leaf	algorithm execution
path length	running time
height of the tree	Worst-case

Searching lower bound:

n - preprocessed items [no matter what we do it still requires $\Omega(\log n)$]

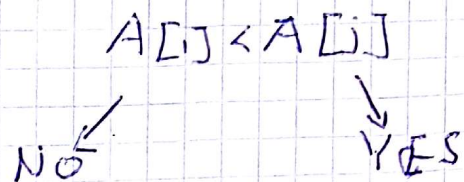
Finding a given item among them in the comparison model requires $\Omega(\log n)$ in worst case

Proof:

- decision tree is binary & must have $\geq n$ leaves - one for each answer $\Rightarrow \text{height} \geq \log n$

Sorting lower bound:

For sorting we claim $\Omega(n \log n)$



leaf $A[5] \leq A[7] \leq A[\dots]$

Decision tree is binary and the number # leaves

\geq # possible answers = $n!$ [number of permutations]

$$\Rightarrow \text{height} \geq \log(n!) = \log(n(n-1)(n-2)\dots 1) =$$

$$\log(n) + \log(n-1) + \dots + \log 2 + \log 1 = \sum_{i=1}^n \log i$$



last half: $\sum_{i=n/2}^n \log i \geq \sum_{i=n/2}^n \log(n/2) = \sum_{i=n/2}^n \log n - 1 =$

$$\frac{n}{2} \log n - \frac{n}{2} = \Omega(n \log n)$$