COUNTING SORT, RADIO SORT, LOWER BOUNDS FOR SORTING

LINEAR TIME SORTING

- comparison model
- lower bounds
  - searching: $\Omega(\lg n)$
  - sorting: $\Omega(n \lg n)$
- $O(n)$ sorting algorithms
  - counting sort
  - radix sort

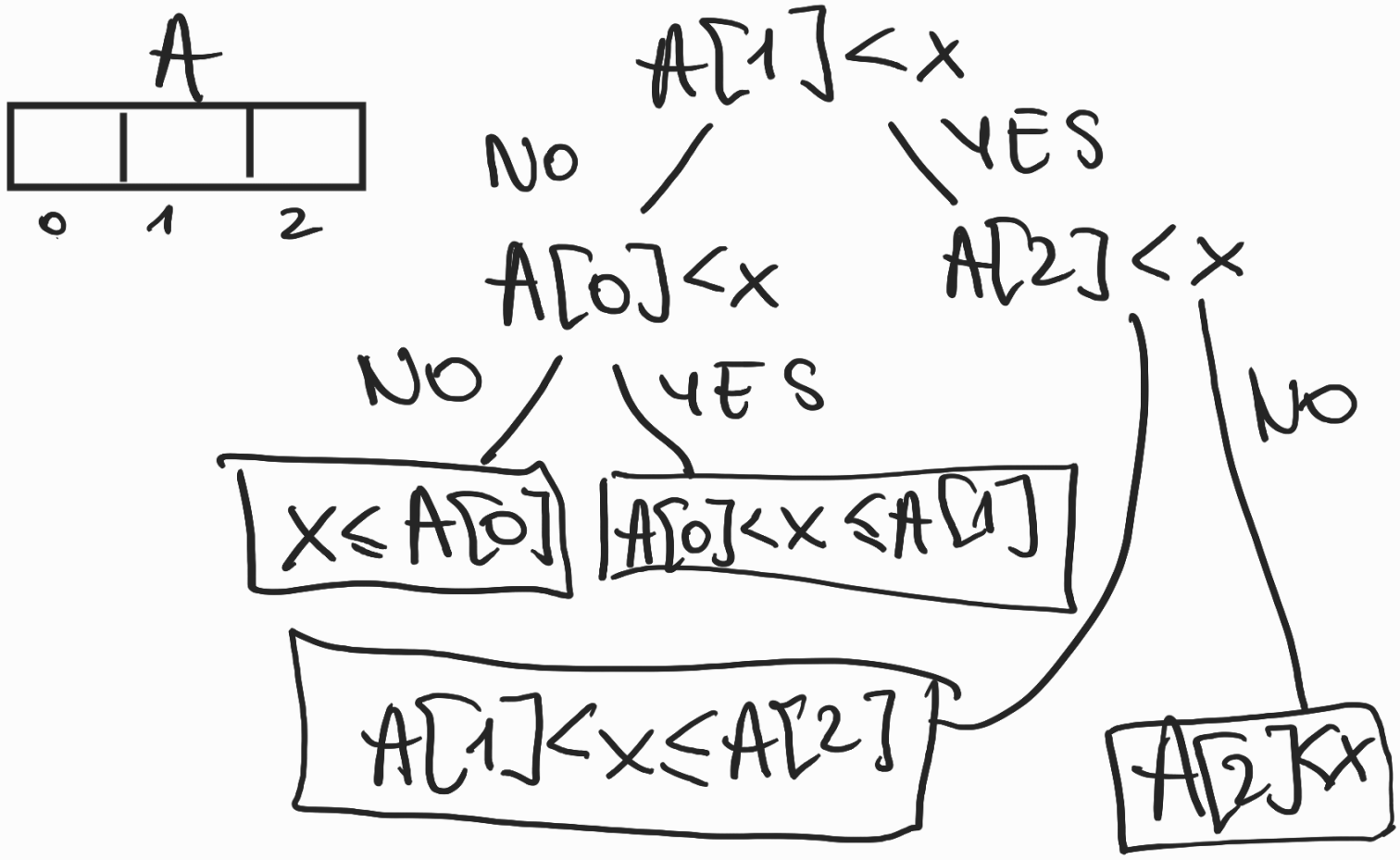
COMPARISON MODEL:

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

DECISION TREE: any comparison algorithm can be viewed as

a tree of all possible comp.s & their outcomes and resulting answer, for any particular n

eg. binary search



DECISION TREE	ALGORITHM
internal node	binary decision (comparisons)
leaf	answer

root-to-leaf	algorithm execution
path length	running time
height of tree	worst case

SEARCHING LOWER BOUND:

- n preprocessed items
- finding a given item among them in comparision model requires $\Omega(\lg n)$ in the worst case

Proof:

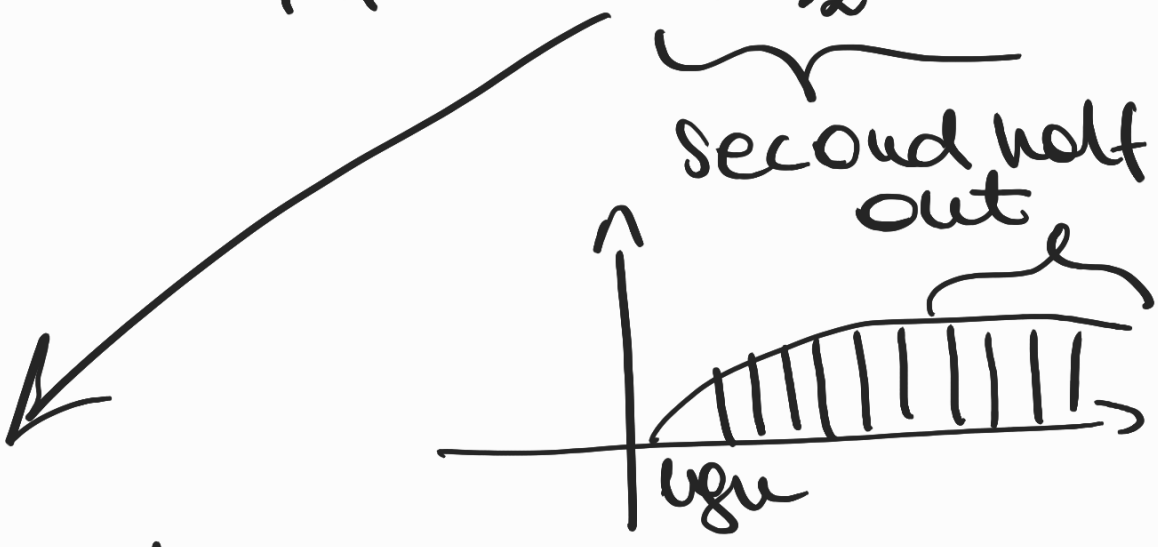
- decision try is binary & must have $\geq n$ leaves, one for each answer
- height $\geq \lg n$ □

SORTING LOWER BOUNDS:

- decision tree is binary & # leaves $\geq$ # possible answers $= n!$
- height $\geq \lg(n!) = \lg(n(n-1)(n-2)\cdots 1)$

$$\lg n + \lg(n-1) + \dots + \lg 1 =$$

$$= \sum_{i=1}^{n} \lg 1 \geq \sum_{i=n/2}^{n} \lg i$$



$$\sum_{i=n/2}^{n} \lg i \geq \sum_{i=n/2}^{n} \lg n/2 = \sum_{i=n/2}^{n} (\lg n - 1) =$$

$$= \frac{n}{2}(\lg n - 1) = \frac{n}{2}\lg n - \frac{n}{2} = \Omega(n \lg n)$$

LINEAR-TIME SORTING

(Integer sorting)

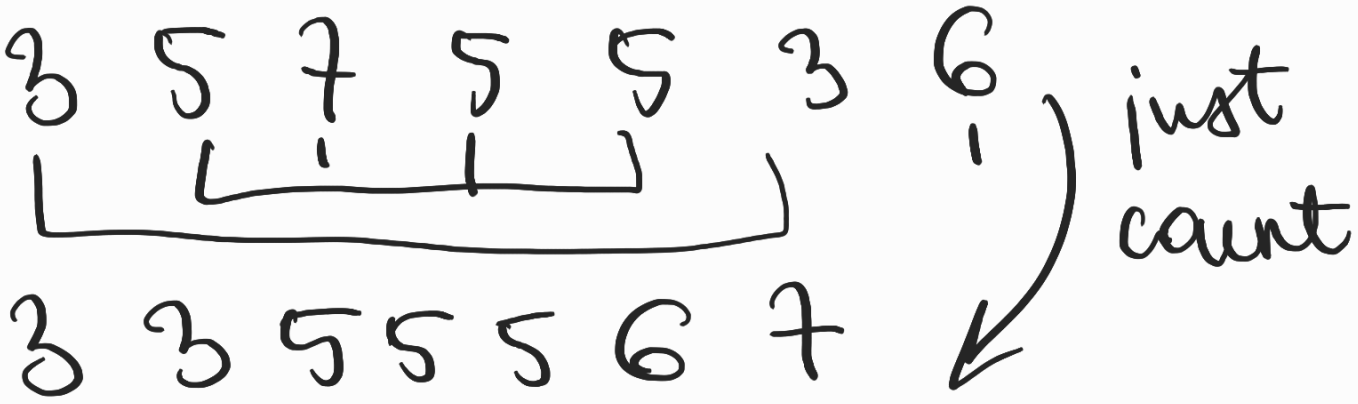
- assume n keys sorting are integers $\in \{0, 1, \dots, k-1\}$

(& each fits in a word)

- can do a lot more than

comparisons
- for $k = n^{O(1)}$ can sort in $\Theta(n)$ time

COUNTING SORT:




```
L = array of k empty lists
for j in range(n):
    L[key(A[j])].append(A[j])
output = []
for i in range(k):
    output.extend(L[i])
```

$O(k)$

$O(1)$

$O(n+k)$

GEEKS FOR GEEKS

1 4 1 2 7 5 2

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0

COUNT

0	1	2	3	4	5	6	7	8	9
0	2	2	0	1	1	0	1	0	0

MODIFY

0	1	2	3	4	5	6	7	8	9
0	2	4	0	5	6	0	7	0	0

2+2 4+1 5+1 6+1

0	1	2	3	4	5	6	7	8	9
0	2	4	4	5	6	6	7	7	7

RADIX SORT

- imagine each integer as base b
- # digits $= d = \log_b k + 1$

{ sort ints by last sig digit
d :
{ sort next

Sort using counting sort by digit $O(n+b)$

Total time: $O((n+b)\cdot d) =$
$= O((n+b)\log_b k)$

min. when $b \in \Theta(n)$
$= O(n \log_n k)$

if $k \leq n^c$ then $O(n \cdot c)$

RADIX SORT

- only to sort numbers

- we sort the numbers from least significant digit to most significant digit
- we use counting sort as a subroutine to sort

170	45	75	90	802	24	2	66

170	80	802	2	24	45	75	66

802	2	24	45	66	170	75	80

2	24	45	66	75	80	170	802