Counting Sort, Radix Sort, Lower Bounds of Sorting and Searching

Today: Linear-Time Sorting

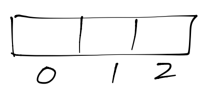
□ Comparison Model

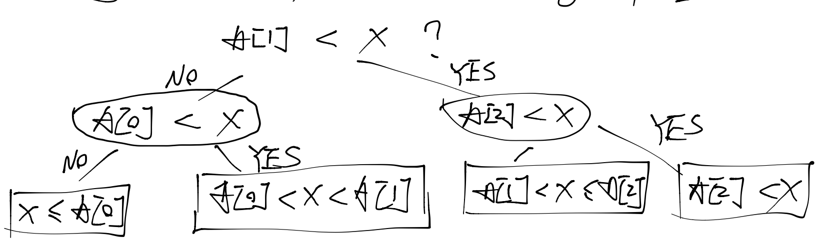
- All input items are black boxes (ADTs)
- Only Operation allowed are comparisions (<, ≤, >, ≥, =)
- Time cost = # of Comparisions

◦ Decision Tree

Any comparision algorithm can be viewed as a tree of all possible Comparisions & their outcomes, and their resulting Answer

e.g. binary search for $n=3$





decision tree:	Algorithm
Internal node	binary decision (Comparison)
leaf	found answer
root to leaf	algorithm excuation
length of path	running time
height of root	worst-case running time

∘ Searching lower bounds

n preprocessed items, finding a given item among them in Comparisons model requires $\Omega(\lg n)$ worst case.

<u>Proof</u>

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

∘ Sorting lower bound

$A[i] < A[j]$

No　　　　Yes

$\boxed{A[5] \leq A[7] \leq A[1] \leq \cdots\cdots}$

\# of comparisons of sorting in comparison model is $\geq n \lg n$

<u>Proof</u>

— decision is binary & # leaves $\geq$ # possible answers $= n!$

$$\Rightarrow \text{height} \geq \lg(n!) = \lg(n(n-1)(n-2)\cdots 1)$$
$$= \lg n + \lg(n-1) + \cdots + \lg 2 + \lg 1$$
$$= \sum \lg(i)$$
$$\geq \sum_{i=n/2}^{n} \lg(i)$$
$$\geq \sum_{i=n/2}^{n} \lg\left(\frac{n}{2}\right) = \sum_{i=\frac{n}{2}}^{n} (\lg n - 1)$$
$$= \frac{n}{2} \lg n - \frac{n}{2}$$
$$= \Omega(n \lg n)$$

□ Linear-Time Sorting (Integer Sorting)

- Assume k keys sorting are integers $\in \{0, 1, \cdots k-1\}$ (& each fits in a word)
- Can do a lot more than comparisons
- for k not too big can sort in $O(n)$ time

◦ Counting Sort

e.g. 3 5 7 5 3 3 6

↳ 3 3 5 5 5 6 7

array for counting

0	0
1	0
2	0
3	2
⋮	

L = array of k empty lists $\quad (O(k))$

    for j in range(n):

        $L[key(A[j])]$.append($A[j]$) $\quad (O(1))$ $\Big\} O(n)$

    output = [ ]

    for i in range(k):

        output.extend($L[i]$) $\quad O(|L_i| + 1)$ $\Big\} O(n+k)$

Algorithm run $O(n+k)$

if $k = \Theta(n)$, then its $O(n)$

◦ Radix Sorted

- Imagine each integer as base b
  - ↳ # digits $d = \log_b k$
- Sort ints by the least significant digits → Sort 2nd least sig digs → ...... (totally d digits), using Counting Sort

  $\rightarrow O((n+b) \cdot d) = O((n+b) \cdot \log_b k)$

  $\underline{\underline{\text{min when } b = \Theta(n)}}$ $O(n \log_n k)$

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