# Lec7_Counting sort, radix sort, lower bounds for sorting and searching

## Lower Bounds

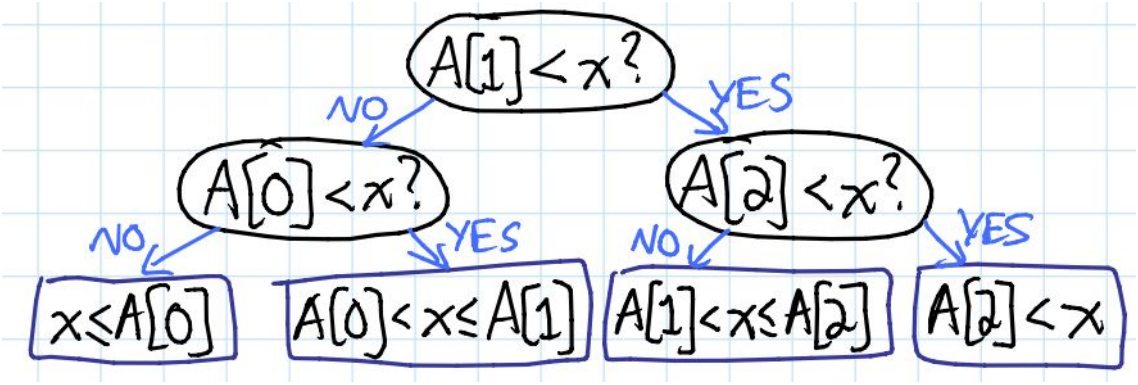
- Searching among n preprocessed items requires Ω(lg n) time
  ⇒ binary search, AVL tree search optimal
- Sorting n items requires Ω(n * lg n)
  ⇒ merge sort, heapsort, AVL sort optimal

## Comparison model of computation

- Input items are black boxes (Abstract Data types)
- Only support comparisons( <, >, >=, etc.)
- time costs = # comparisons

## Decision tree

- Any comparison algorithms can be viewed /specified as a tree of all possible comparison outcomes & resulting output, for a particular n :
- example)




| Decision Tree | Algorithm |
|---|---|
| Internal node | Binary Decision |
| Leaf | Found answer |
| Root to leaf | Algorithm Execution |
| Path length | Running Time |
| Height of the tree | Worst case running time |

## Lower Bound of Searching

- # of leaves ≥ # possible answers
  ≥ n( at least 1 per A[i])
- Decision tree is binary
  ⇒ height ≥ lg Θ(n) = lg n ± Θ(1)

## Lower Bound of Sorting

- Leaf specifies answers as permutation
- All n! are possible answers
  ⇒ # leaves ≥ n!
  ⇒ height ≥ lg n! = Ω(n lg(n)) → by Sterling's formula

## Linear - Time Sorting

- If n keys are integers ∈ {0,1,2, … , k-1} can do more than compare them
  ⇒ lower bounds don't apply
- If k = n ^ (O(1)), can sort in O(n) time

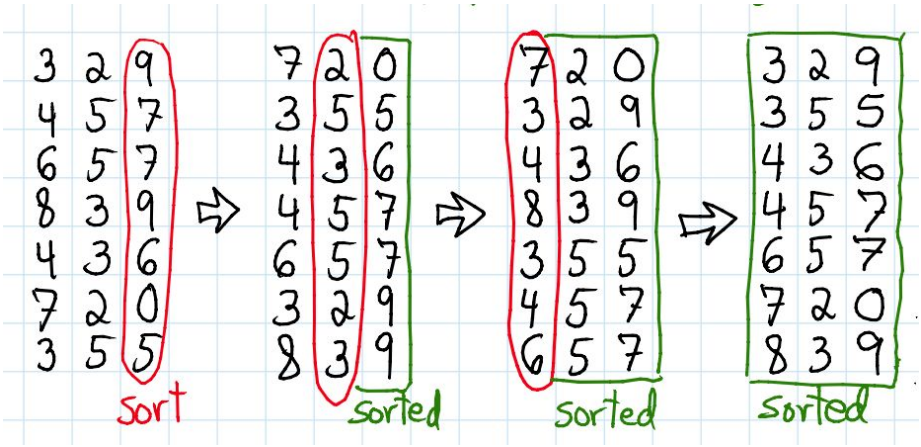
## Types of Sorts

- Counting Sort
  - Code

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

  - Time : Θ(n+k) / Space :  Θ(n+k)
- Radix Sort
  - Imagine integer in base b
    ⇒ d = logb(k) digits ∈ {0,1,..., b-1}

- Sort by least significant digits to the most significant digit
  ⇒ don't mess up previous sorting




- Time
  ⇒ Θ(n+b) per digit
  ⇒ Θ((n+b)*d) = Θ((n+b)*logb(k)) → minimized when b = n
  ⇒ Θ((n)*logn(k))
  ⇒ O(cn) if k ≤ n^c