Comparison Sort:

Comparison based sorting algorithms: Sorts the elements by comparing pairs of them.

We can also interpret this as follows.

Suppose we have some objects, which are hidden in a box, The goal is to sort the objects based on their weight without any information except that obtained by placing two weights on the scale and seeing which one is heavier.

All the Algorithms we have Studied so far are Comparison sorts.

- Insertion Sort
- Merge sort
- Quick Sort
- Heap sort

We know that Merge Sort and Heap sort can sort n numbers in $O(n \log n)$ time.

In this class, we prove the following

Any comparison sort must make $\Omega(n \log n)$ comparisons in the worst case to sort n elements.

ie., Merge sort and Heap sort are asymptotically optimal and no comparison sort exists that is faster by more than a constant factor.

Main Result

Any comparison sort algorithm requires $\Omega(n \log n)$ comparisons in the worst case.

The decision tree model :

We can view Comparison Sort interms of decision trees

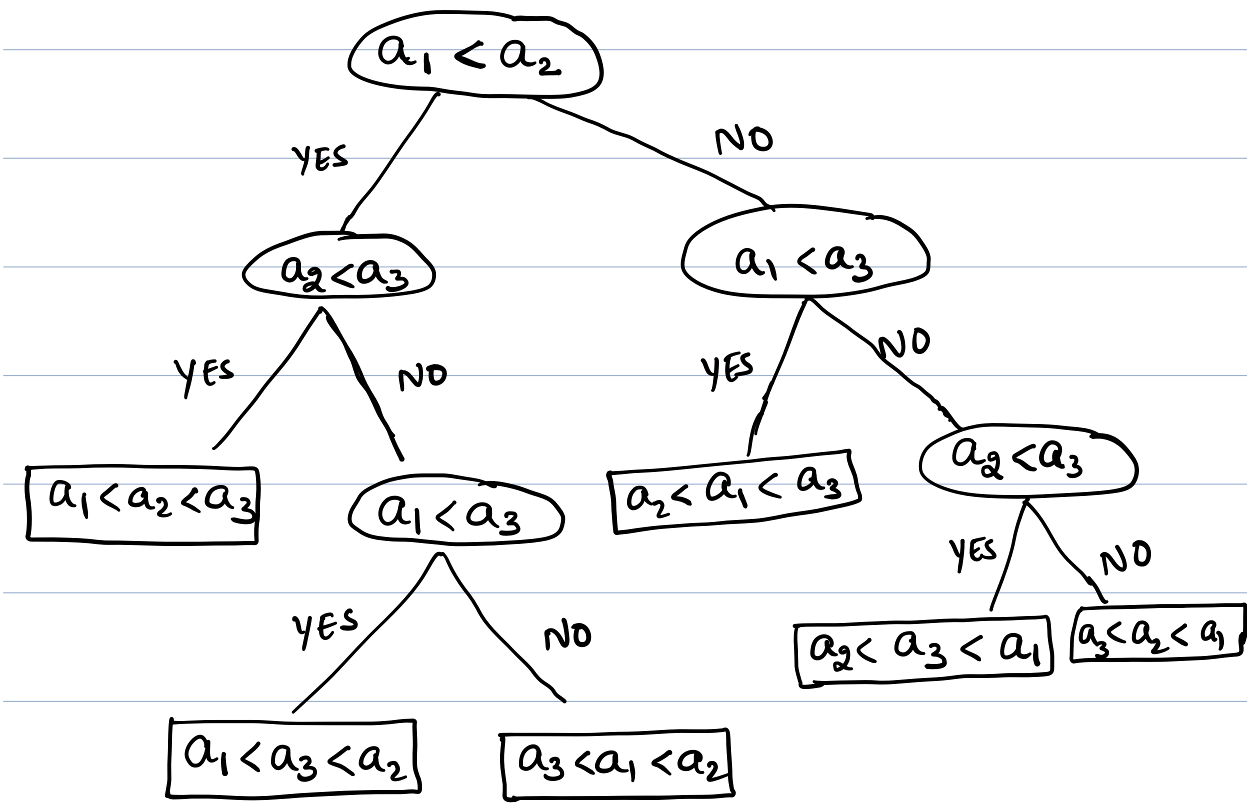
→ Full binary tree (every internal node has two children)

In decision tree each internal node corresponds to a comparison of pair of elements.

Say $A = \begin{array}{|c|c|c|} \multicolumn{1}{c}{1} & \multicolumn{1}{c}{2} & \multicolumn{1}{c}{3} \\ \hline a_1 & a_2 & a_3 \\ \hline \end{array}$ a_i's are distinct

Q: How insertion sort works on A?

Example: $A = \{a_1, a_2, a_3\}$, Assume all a_i's are distinct.



Number of leaves $= 6$

Height of the tree $= 3$

Decision tree		Algorithm
Internal node	$\longleftrightarrow$	Comparisons
leaf	$\longleftrightarrow$	Sorted ordering of the input
root-to-leaf Path	$\longleftrightarrow$	algorithm execution
length of the Path	$\longleftrightarrow$	Running time
height of the tree	$\longleftrightarrow$	Worst case running time

A lower bound on the heights of all decision trees in which every permutation appears as a reachable leaf is therefore a lower bound on the running time of any comparison sort algorithm.

Sorting lower bound:

In general, if the input has n numbers then the decision tree have $n!$ leaves. Each leaf is reachable from the root.

Consider a decision tree of height h with l leaves corresponding to a comparison sort on n elements.

$$n! \leq l \leq 2^h$$

$$h \geq \log(n!)$$

$$= \Omega(n \log n)$$

Counting Sort: [Non-Comparison sort]

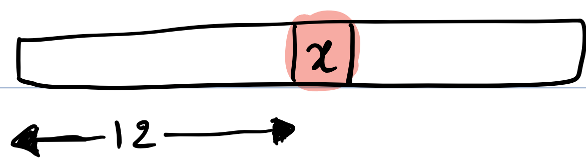
Assumption: Each of the n input elements is an integer in the range 0 to k, for some integer k.

When $k = O(n)$, then couting sort runs in $\Theta(n)$ time.

Idea of the Algorithm:

Counting sort algorithm determines for each input element x, the number of elements less than x and then places x directly into its position in the output array

Eg: if 12 elements are less than x then x will be at 13^{th} position in the output



Algorithm

COUNTING-SORT(A, B, k)

let $C[0\,..\,k]$ be a new array
for $i = 0$ **to** k
    $C[i] = 0$
for $j = 1$ **to** $A.length$
    $C[A[j]] = C[A[j]] + 1$
// $C[i]$ now contains the number of elements equal to i.
for $i = 1$ **to** k
    $C[i] = C[i] + C[i-1]$
// $C[i]$ now contains the number of elements less than or equal to i.
for $j = A.length$ **downto** 1
    $B[C[A[j]]] = A[j]$
    $C[A[j]] = C[A[j]] - 1$

Example:

$k = 5$

A

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

$C[0, \ldots k]$

C

0	1	2	3	4	5
2	0	2	3	0	1

C[i] contains the number of element equal to i

C

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

C[i] contains the number of element less than or equal to i

$B[1, \ldots n]$ is the output array

B =

1	2	3	4	5	6	7	8
						3	

C =

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

B =

1	2	3	4	5	6	7	8
	0					3	

C =

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

B =

1	2	3	4	5	6	7	8
	0				3	3	

C =

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

By repeating the above process, the final sorted output array B is

B =

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

Running time

Counting-Sort(A, B, k)

```
let C[0..k] be a new array          — Θ(1)
for i = 0 to k
    C[i] = 0                        Θ(k)
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1           Θ(n)
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]          Θ(k)
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]               Θ(n)
    C[A[j]] = C[A[j]] - 1
```

Total running time $= \Theta(1) + \Theta(k) + \Theta(n) + \Theta(k) + \Theta(n)$

$= \Theta(k+n)$

If we have $k = O(n)$ then

running time is $\Theta(n)$.

Remarks:

- Counting sort beats the lower bound $\Omega(n \log n)$

- This is not a comparison sort

Other Non-Comparison sort Algorithms

Radix sort

Refer to 8.3 & 8.4

Coreman's text book.

Bucket sort

Stable Sort : A sorting algorithm is stable if two objects with equal values appear in the same order in the sorted output as they appear in the input array.

Insertion Sort, Merge Sort are stable.

① Quick Sort is not Stable

Example

4 2 1 4 3 → pivot

2 4 1 4 3

2 1 4 4 3

2 1 3 4 4