

Lower bound for comparison sorting

CS1 - Merge Sort $O(n \log n)$
 exp Quick Sort $O(n \log n)$
 Heap Sort $O(n \log n)$

} We can't do better than this for comparison based sorts!

If an alg sorts all of these, it MUST make a different decision somewhere in its decision tree.

Input
 1, 2, 3
 1, 3, 2
 2, 1, 3
 2, 3, 1
 3, 1, 2
 3, 2, 1

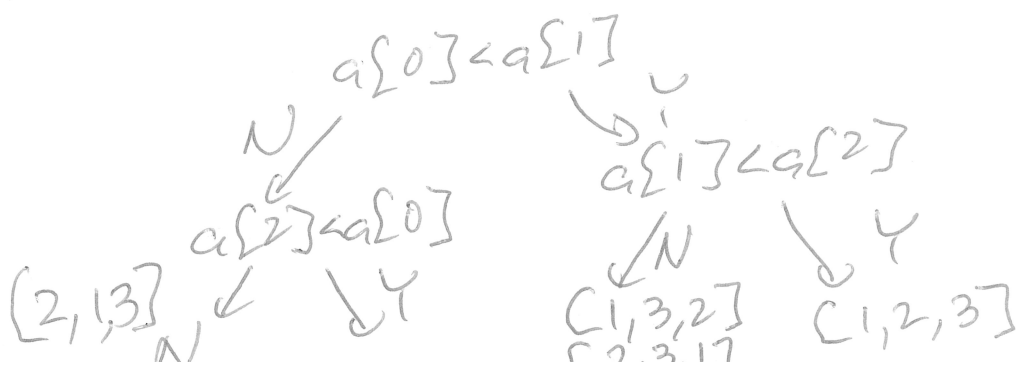
Input can come in $n!$ ways



$a[0] \ a[1] \ a[2]$

Program
 NN
 NY
 YN
 YY

Limit prog to do 2 comparisons w/ 6 unique inputs, there will be 2 diff inputs that go down the same decision tree.



If we are to distinguish btw X different inputs, we must have at least X different computation paths,

Comparison Based Sorting (sorting n items)

$$\# \text{ inputs} = n!$$

$$\# \text{ decision paths for } k \text{ comparisons} = 2^k$$

Stirling's Approx

$$2^k \geq n!$$

$$2^k \geq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$$

$$\log_2 2^k \geq \log_2 \sqrt{2\pi n} + \log_2 \left(\frac{n}{e}\right)^n$$

$$k \geq \underbrace{\text{small}}_{O(\log n)} + n \log_2 \left(\frac{n}{e}\right)$$

$$k \geq O(\log n) + n \left(\log_2 n - \log_2 e \right) \quad \text{Const}$$

$$k \geq O(\log n) + O(n \log n)$$

What's the fewest # of comp necessary to sort 8 #s?

$$8! = 40,320$$

$$2^k > 40,320$$

$$k > \log_2 40320$$

$$k \geq 16$$

$$2^{16} = 65536$$

$$2^{15} = 32768$$

$$15! \sim 3.2 \times 10^{10}$$

$$2^{137} \sim 4 \times 10^9$$

Radix Sort

Works on arrays of digits

236 821 437 329 188 616 331 427 ↑↑	⇒ last digit	821 331 236 616 437 427 188 329 ↑↑	⇒ 10 digit	616 821 427 329 331 236 437 188 ↑	⇒ 100 digit	188 236 329 331 427 437 616 821 Sorted!!!
		ones sort		tens sort		

Intermediate Sort to sort by digit is Counting Sort. Counting Sort is stable

arr 0 1 2 3 4 5 6 7 8 9 10
 arr 2, 6, 3, 0, 4, 2, 2, 6, 3, 4, 3

```
for (i=1; i<d; i++)
  f[i] += f[i-1]
```

0	1	2	3	4	5	6
0	0	0	0	0	0	0

freq

1	+	+	+	+
2	2	2	2	2
3	3	3	3	3

1	0	3	3	2	0	2
---	---	---	---	---	---	---

freq

0	1	2	3	4	5	6
1	1	4	7	9	9	11

Cumulative frequency

cpy

0	2	2	2	3	3	3	4	4	6	6
---	---	---	---	---	---	---	---	---	---	---

// go through array

```
for (i=n-1; i>=0; i--)
```

0	1	2	3	4	5	6
+	1	4	7	9	9	11
0	3	6	8	9	10	11
	2	8	7	9		
	1	4				

freq

$$cf[i] = \sum_{j=0}^i f[j]$$

Greedy Algorithms

In CS1, we did brute force, followed up w/ backtracking for both it's easy to prove correctness, but the run-time quickly get unmanageable!

Some problems where we can PROVE that it's not necessary to consider all options to find the best one.

An algorithm that correctly solves a problem by only considering the "best option in the moment" as opposed to all options is called a greedy algorithm.

Planting Trees

4 → # trees

2 3 4 3 → how long to grow

4	3	3	2
1	2	3	4

5 5 6 6 ⇒ >

Greedy choice
plant the trees
that take longest

← 1st exchange
arg if you're out
of order I
no ans
rows