Sorting Algorithms

- Selection Sort
- Insertion Sort
- Bubble Sort
- Shaker Sort
- Shell Sort
- Merge Sort
- Heap Sort
- Quick Sort

- Bucket & Radix Sort
- Counting Sort

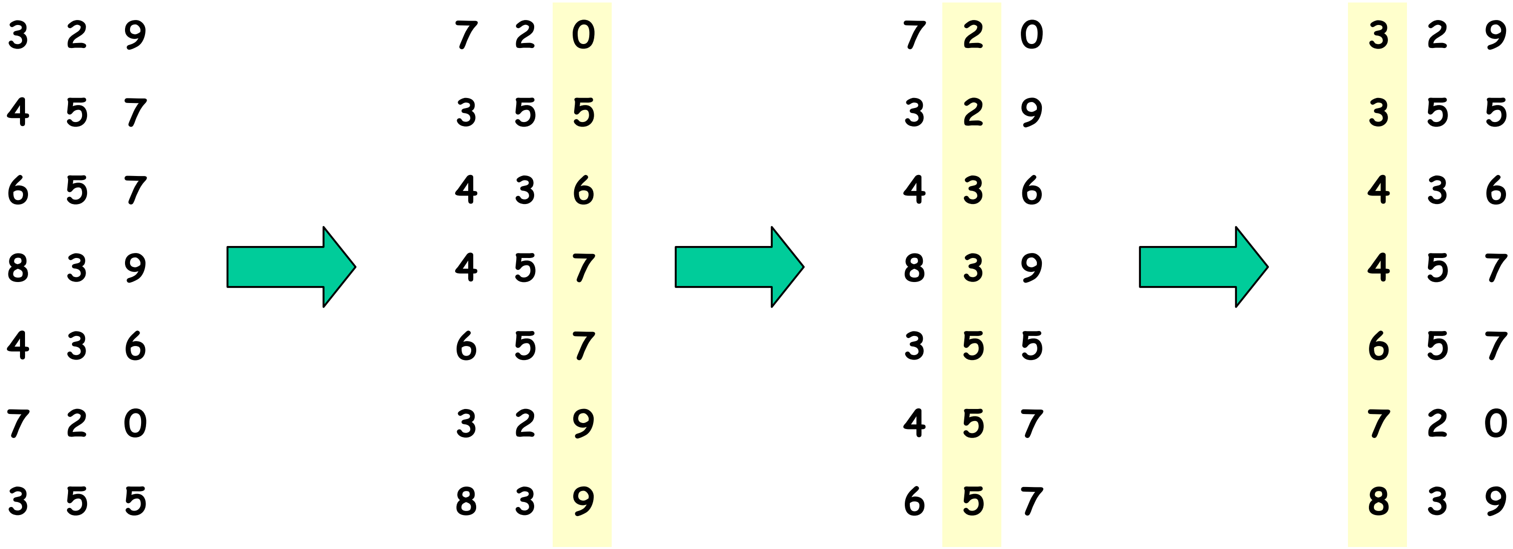
Bucket Sort

- N values in the range [a..a+m-1]
- For e.g., sort a list of 50 scores in the range [0..9].
- Algorithm
  - Make m buckets [a..a+m-1]
  - As you read elements throw into appropriate bucket
  - Output contents of buckets [0..m] in that order
- Time O(N+m)

Stable Sort

- A sort is stable if equal elements appear in the same order in both the input and the output.
- Which sorts are stable? Homework!

Radix Sort



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

Algorithm

```
for i = 1 to d do
        sort array A on digit i using a stable sort algorithm
```

Time Complexity: O((n+k)d)

Counting Sort

Initial Array

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

Counts

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

Cumulative Counts

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

Order Statistics

- Maximum, Minimum n-1 comparisons

7	3	1	9	4	8	2	5	0	6

- MinMax
  - 2(n-1) comparisons
  - 3n/2 comparisons
- Max and 2ndMax
  - (n-1) + (n-2) comparisons
  - ???

Upper & Lower Bounds

- Algorithm A solves problem P if it terminates & gives the correct output on every possible input.
- Algorithm A solving problem P has time complexity f(n) if it takes time at most f(n) for every input of length n.
- U(n) is an upper bound on the time complexity of P, if there exists an algorithm A that solves P and has time complexity U(n).
- L(n) is a lower bound on the time complexity of P, if there exists NO algorithm that solves P and has time complexity asymptotically less than L(n).

Upper & Lower Bounds for Maximum

- Naïve Algorithm A solves the Maximum problem, because it terminates in n iterations for every possible input of length n and outputs the correct maximum.
- Naïve Algorithm A has time complexity O(n).
- O(n) is an upper bound on the time complexity of the maximum problem.
- (n-1) is a lower bound on the time complexity of the maximum problem, because there exists NO algorithm that solves it with less than n-1 comparisons.
- WHY? In 1 comparison, at most 1 item is eliminated from being the maximum. How many to eliminate?
- Therefore, no matter how smart you are you cannot design an algorithm that solves the Maximum problem in less than n-1 comparisons on all inputs of length n.

Upper Bound on Sorting n items

- O(n log n) is the upper bound for sorting.
- WHY?
  - HeapSort
  - MergeSort
- What about QuickSort?
  - O(n^2) in the worst case!

Lower Bound for Sorting: Decision Tree Model

- The decision tree model models all comparison-based algorithms that solve the sorting problem. These algorithms perform no other "algebraic" operations on input values. They may perform data movements & other statements.
- Imagine a binary tree that models the algorithm, where
  - each node corresponds to a comparison
  - the edges to the children correspond to the two outcomes of the comparison: YES/NO
  - Leaves correspond to the output. WHAT IS THE OUTPUT?
- Decision tree for InsertionSort on 4 items?
- What can we say about such decision trees?
- Given an input, the algorithm follows a path from the root to a leaf.

Lower Bound for Sorting: Cont’d

- Leaves correspond to outputs.
- Paths correspond to a path followed on a specific input. Time complexity = height of decision tree.
- Different input orders must force different paths or else the output will end up being the same, giving rise to incorrect sorted orders.
- Therefore number of leaves is at least as large as the number of different input orders.
  - HOW MANY?
  - n!
- Height of the decision tree is at least log(n!). Hence lower bound is O(log(n!)) = O(n log n)