Searching and Sorting

- Linear and Binary Search
- Order Statistics - MIN and MAX
- Comparison Based Sorting Algorithms
  - Selection Sort
  - Bubble Sort
  - Insertion Sort
- Lower Bound on Comparison based sorting
- Non-Comparison Based Sorting - Integers Sorting
  - Counting Sort
  - Radix Sort

IMDAD ULLAH KHAN

Lower Bound on Sorting Algorithms

Theorem: Any comparison-based algorithm requires $\Omega(n \log n)$ comparisons to sort n elements

Comparison Based Algorithm

- Only access to elements is via pairwise comparisons
- No direct manipulations allowed
- No decision based on values of elements or number of bits/digits

Linear Time Sorting Algorithms

Non-Comparison Based Algorithm

- Allowed to use value of the input, e.g. maximum of the array
- Works for integers only
- Runtime typically depends on value of the input (not size of input)

▷ Called *pseudo-polynomial* time algorithms

Counting Sort

- Suppose A has n distinct positive integers between 1 and K
- Counting Sort essentially populates an *'attendance register'* C
- Then outputs the elements present by scanning C in increasing order

initially empty

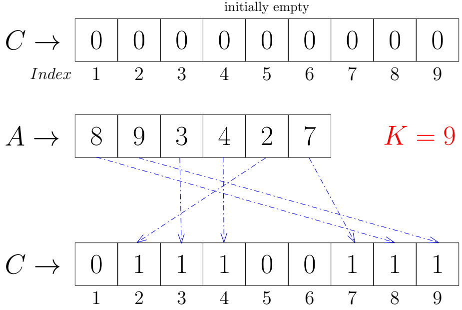
$C \rightarrow$	0	0	0	0	0	0	0	0	0
$Index$	1	2	3	4	5	6	7	8	9

$A \rightarrow$	8	9	3	4	2	7

$K = 9$

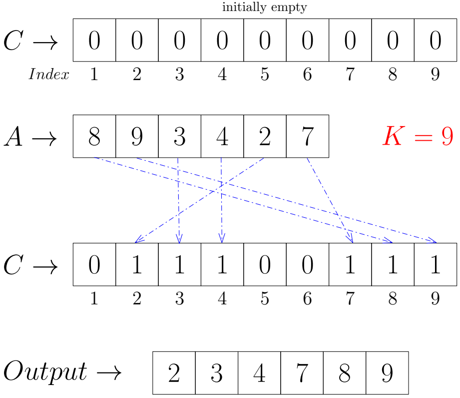
Counting Sort

- Suppose A has n distinct positive integers between 1 and K
- Counting Sort essentially populates an *'attendance register'* C
- Then outputs the elements present by scanning C in increasing order



Counting Sort

- Suppose A has n distinct positive integers between 1 and K
- Counting Sort essentially populates an *'attendance register'* C
- Then outputs the elements present by scanning C in increasing order



Counting Sort

- Suppose A has n distinct positive integers between 1 and K
- Counting Sort essentially populates an *'attendance register'* C
- Then outputs the elements present by scanning C in increasing order

---

Algorithm COUNTING-SORT(A, K)

```
C ← ZEROS(K)                    ▷ Initialize C to empty register of size K
for j ← 1 to n do
    C[A[j]] ← 1
for i ← 1 to K do
    if C[i] = 1 then
        PRINT(i)
```

- Runtime is $O(n+K)$, $O(n)$ for first loop, $O(K)$ for second loop
- The algorithm uses the value of K and runtime depends on it too

What if numbers are negative? How about repeated elements?

Counting Sort

Negative numbers are handled by shifting (and re-shifting the output)

For repeated elements, store frequencies in C and output each element the number of times it occurs

Algorithm $\textsc{counting-sort}(A, K)$

$C \leftarrow \textsc{zeros}(K)$ ▷ Initialize C to empty register of size K
for $j \leftarrow 1$ to n **do**
  $C[A[j]] \leftarrow C[A[j]] + 1$
for $i \leftarrow 1$ to K **do**
  for $j \leftarrow 1$ to $C[i]$ **do**
    $\textsc{print}(i)$

Runtime is still $O(n + K)$

Counting Sort: Stable Version

A sorting algorithm is called stable, if repeated elements are output in the original order

Let B be the array in which elements would be placed in sorted order

Algorithm $\textsc{stable-counting-sort}(A, B, K)$

$C \leftarrow \textsc{zeros}(K)$ ▷ Initialize C to empty register of size K
for $j \leftarrow 1$ to n **do**
  $C[A[j]] \leftarrow C[A[j]] + 1$
for $i \leftarrow 1$ to K **do**
  $C[i] \leftarrow C[i] + C[i-1]$ ▷ Modify C to keep cumulative counts
for $j \leftarrow n$ to 1 **do**
  $B[C[A[j]]] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort: Stable Version

$C =$

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
C	0	0	0	0	0	0	0	0	0	0	0	0	0

$A =$

8	9	3	4	2	7	3	13	7	13

$C =$

	1	2	3	4	5	6	7	8	9	10	11	12	13
C	0	1	2	1	0	0	2	1	1	0	0	0	2

$C =$

	1	2	3	4	5	6	7	8	9	10	11	12	13
C	0	1	3	4	4	4	6	7	8	0	0	0	10

Counting Sort: Stable Version

step 1

$A =$

8	9	3	4	2	7	3	13	7	13

$C =$

0	1	3	4	4	4	6	7	8	0	0	0	10
1	2	3	4	5	6	7	8	9	10	11	12	13

$B =$

										13
Index	1	2	3	4	5	6	7	8	9	10

$C =$

0	1	3	4	4	4	6	7	8	0	0	0	9
1	2	3	4	5	6	7	8	9	10	11	12	13

Counting Sort: Stable Version

step 2

$A =$

8	9	3	4	2	7	3	13	7	13

$C =$

0	1	3	4	4	4	6	7	8	0	0	0	9
1	2	3	4	5	6	7	8	9	10	11	12	13

$B =$

						7				13
$Index$	1	2	3	4	5	6	7	8	9	10

$C =$

0	1	3	4	4	4	5	7	8	0	0	0	9
1	2	3	4	5	6	7	8	9	10	11	12	13

Counting Sort: Stable Version

step 3

$A =$	8	9	3	4	2	7	3	13	7	13

$C =$	0	1	3	4	4	4	6	7	8	0	0	0	9
	1	2	3	4	5	6	7	8	9	10	11	12	13

$B =$						7			13	13
$Index$	1	2	3	4	5	6	7	8	9	10

$C =$	0	1	3	4	4	4	5	7	8	0	0	0	8
	1	2	3	4	5	6	7	8	9	10	11	12	13

Radix Sort

- Suppose A has n integers, each of d digits (same number of digits)
- From least significant digit to most significant digit
- Stable Sort the integers by the ith digits

$$A = \{385, 400, 1, 129, 10\}$$

3	8	5
4	0	0
0	0	1
1	2	9
0	1	0

Radix Sort

- Suppose A has n integers, each of d digits (same number of digits)
- From least significant digit to most significant digit
- Stable Sort the integers by the ith digits

$$A = \{385, 400, 1, 129, 10\}$$

3	8	5
4	0	0
0	0	1
1	2	9
0	1	0

Pass 1

4	0	0
0	1	0
0	0	1
3	8	5
1	2	9

Radix Sort

- Suppose A has n integers, each of d digits (same number of digits)
- From least significant digit to most significant digit
- Stable Sort the integers by the ith digits

$$A = \{385, 400, 1, 129, 10\}$$

3	8	5
4	0	0
0	0	1
1	2	9
0	1	0

Pass 1

4	0	0
0	1	0
0	0	1
3	8	5
1	2	9

Pass 2

4	0	0
0	0	1
0	1	0
1	2	9
3	8	5

Radix Sort

- Suppose A has n integers, each of d digits (same number of digits)
- From least significant digit to most significant digit
- Stable Sort the integers by the ith digits

$$A = \{385, 400, 1, 129, 10\}$$

3	8	5
4	0	0
0	0	1
1	2	9
0	1	0

Pass 1

4	0	0
0	1	0
0	0	1
3	8	5
1	2	9

Pass 2

4	0	0
0	0	1
0	1	0
1	2	9
3	8	5

Pass 3

0	0	1
0	1	0
1	2	9
3	8	5
4	0	0

Radix Sort

- Suppose A has n integers, each of d digits (same number of digits)
- From least significant digit to most significant digit
- Stable Sort the integers by the ith digits

Algorithm RADIX-SORT(A, d)

---

for $i \leftarrow 1$ to d **do**
  Use STABLE-COUNTING-SORT to sort A by digit i

- Let $[1 - k]$ be the range of values a *'digit'* can take
- Runtime is $O(d * (n + k))$

Is RADIX-SORT stable? Yes.

What if the unstable version of COUNTING-SORT was used instead?

Sorting Algorithms

Algorithm	Worst Case Runtime
Bubble Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Count Sort	$O(n+k)$
Radix Sort	$O(d*(n+k))$