CSE331 Introduction to Algorithm
Lecture 16: Radix Sort and Bucket Sort

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Introduction

- This is the last lecture on sorting.
- In the previous lecture, I gave an $\Omega(n \log n)$ lower bound for comparison-based sorting algorithms.
- I also presented COUNTING SORT, which runs in $O(n+k)$ time when the numbers are integers between 0 and k. So this is linear time if $k = O(n)$.
- Today, I present two more algorithms that run in linear time in special cases.
- **Reference**: Section 8.3 and 8.4 of the textbook Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

Radix Sort

186		851		719		056
394		394		325		134
056		834		834		186
719	→	134	→	134	→	325
851		325		851		394
834		056		056		719
325		186		186		834
134		719		394		851

Radix Sort

Pseudocode

procedure RADIX-SORT(A, d)
  for $i \leftarrow 1, d$ **do**
    sort A according to digit i using a stable sort

Analysis

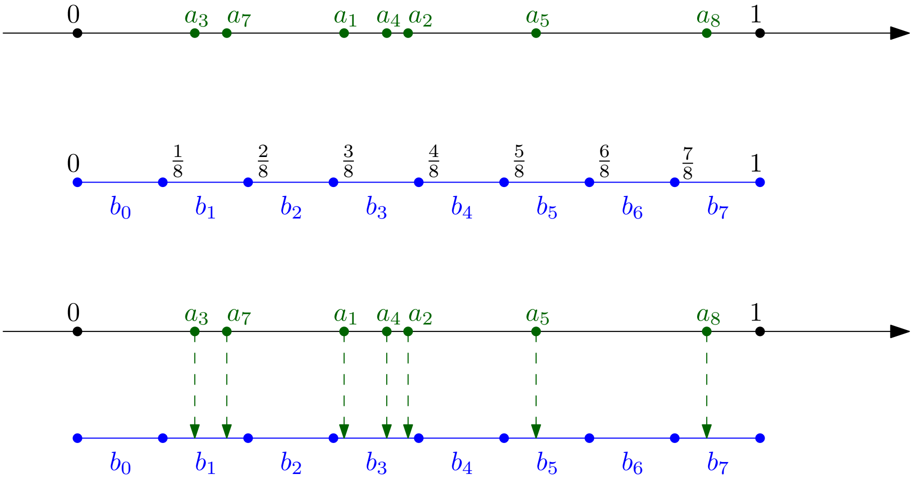
Suppose that the stable sort we use is COUNTING SORT. Then RADIX SORT runs in time $O(d(n+k))$ where d is the number of digits and k is the number of possible values for each digit.

- If the numbers are written in base 10, we have $k=9$, hence the running time is $O(d \times (n+9)) = O(dn)$.
- If the numbers are 64-bit integers, the running time is $O(64 \times (n+1)) = O(n)$.

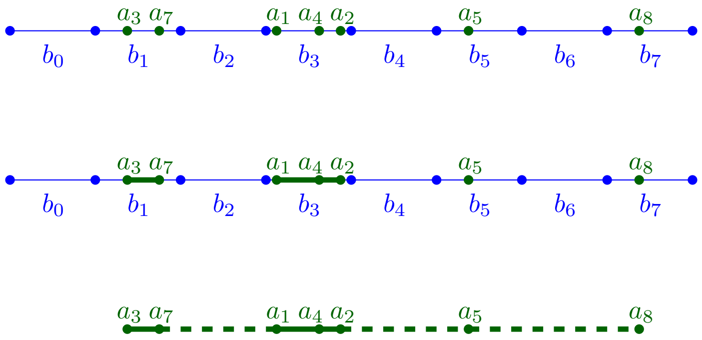
Radix Sort

- A good implementation of RADIX SORT may be faster than QUICKSORT for sorting arrays of integers.
- Drawbacks:
  - Only applies to integers, while QUICKSORT or MERGE SORT also work with floating point numbers.
  - Not in-place due to COUNTING SORT.
- The $O(n)$ running time for 64-bit integers does not contradict our $\Omega(n \log n)$ lower bound because RADIX SORT is not comparison based: It does not only compare input numbers, it also accesses their digits.
- Another reason: on large integers it does not run in linear time. For instance, for $\log n$-bit integers, it runs in $\Theta(n \log n)$ time.

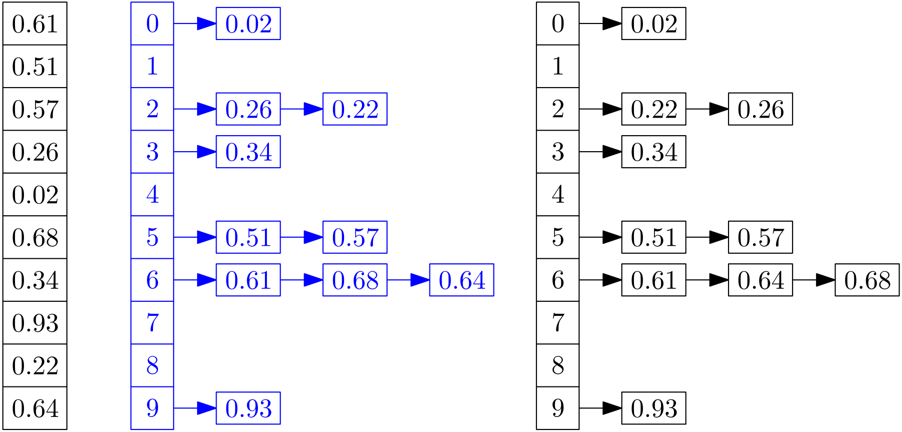
Bucket Sort


0
a_3 a_7 a_1 a_4 a_2 a_5 a_8
1
0
$\frac{1}{8}$ $\frac{2}{8}$ $\frac{3}{8}$ $\frac{4}{8}$ $\frac{5}{8}$ $\frac{6}{8}$ $\frac{7}{8}$
1
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
0
a_3 a_7 a_1 a_4 a_2 a_5 a_8
1
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7

Bucket Sort


a_3 a_7 a_1 a_4 a_2 a_5 a_8
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
a_3 a_7 a_1 a_4 a_2 a_5 a_8
b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7
a_3 a_7 a_1 a_4 a_2 a_5 a_8

Bucket Sort



Input Buckets Sorted buckets

Bucket Sort

- INPUT: $a_1, \dots, a_n \in [0, 1)$
  - It means $0 \leqslant a_i < 1$ for all i.
- We create n *buckets* $b_j = \left[\frac{j}{n}, \frac{j+1}{n}\right)$, $j = 0, \dots n-1$.
- Each a_i is placed in the bucket b_j such that $a_i \in b_j$.
- We sort each bucket separately using insertion sort.
- Finally we concatenate the sorted lists corresponding to buckets $b_0, \dots b_{n-1}$.

Bucket Sort

Pseudocode

procedure $\textsc{BucketSort}(A[1 \dots n])$
    $B[0 \dots n-1] \leftarrow$ new array of lists
    for $j \leftarrow 0, n-1$ **do**
        $B[j] \leftarrow$ empty list
    for $i \leftarrow 1, n$ **do**
        $j \leftarrow \lfloor n.A[i] \rfloor$
        insert $A[i]$ into $B[j]$
    for $j \leftarrow 0, n-1$ **do**
        $\textsc{InsertionSort}(B[j])$
    return $B[0].B[1] \dots B[n-1]$ $\triangleright$ concatenation of sorted lists

Bucket Sort

- At line 7, we insert $A[i]$ at the end of $B[j]$ if we need the algorithm to be stable. It can be done in $O(1)$ time by keeping a pointer to the tail of the list.
- If we don't count the calls to INSERTION SORT, it is clear that BUCKET SORT takes $\Theta(n)$ time.
- Worst case: If all $A[i]$ fall in the same bucket, $\Omega(n^2)$ time.
- Best case: If each bucket contains one input number, $O(n)$ time.
- Average case?
  - BUCKET SORT is deterministic, so we will determine the expected running time $\mathbb{E}[T(n)]$ over a distribution of input numbers.
  - We will assume that the input numbers are chosen *uniformly and independently at random* in $[0, 1)$.

Bucket Sort: Analysis

- Let n_j denote the number of input numbers a_i that fall in bucket b_j:

$$n_j = |\{a_i \mid a_i \in b_j\}|$$

Proposition

BUCKET SORT *runs in* $O(n + \sum_j n_j^2)$ *time.*

- By linearity of expectation, it implies that the expected running time is

$$\mathbb{E}[T(n)] = O\left(n + \sum_{j=0}^{n-1} \mathbb{E}[n_j^2]\right)$$

- So we need to determine $\mathbb{E}[n_j^2]$.

Bucket Sort: Analysis

- Let X_i be the following indicator random variable:

$$X_i = \begin{cases} 0 & \text{if } a_i \notin b_0 \\ 1 & \text{if } a_i \in b_0 \end{cases}$$

- Then $n_0 = \sum_{i=1}^{n} X_i$, and

$$n_0^2 = \left(\sum_{i=1}^{n} X_i\right) \cdot \left(\sum_{j=1}^{n} X_j\right) = \sum_{i=1}^{n} \sum_{j=1}^{n} X_i X_j = \sum_{i=1}^{n} X_i^2 + \sum_{\substack{1 \leqslant i \leqslant n \\ 1 \leqslant j \leqslant n \\ i \neq j}} X_i X_j$$

- By linearity of expectation, it follows that

$$\mathbb{E}\left[n_0^2\right] = \sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + \sum_{i \neq j} \mathbb{E}\left[X_i X_j\right]$$

Bucket Sort: Analysis

- X_i^2 only takes values 0 or 1.
- In other words, it is an indicator random variable, and therefore

$$\mathbb{E}\left[X_i^2\right] = \Pr[X_i^2 = 1].$$

- Since a_i is chosen uniformly at random in $[0, 1)$, it follows that

$$\mathbb{E}\left[X_i^2\right] = \Pr[X_i^2 = 1] = \frac{1}{n}.$$

Bucket Sort: Analysis

- Suppose $i \neq j$.
- $X_i X_j$ is also an indicator random variable, so

$$\mathbb{E}\big[X_i X_j\big] = \Pr[X_i X_j = 1].$$

- $\Pr[X_i X_j = 1]$ is the probability that a_i and a_j fall in b_0.
- It is $1/n^2$ because a_i and a_j are chosen independently, so

$$\mathbb{E}\big[X_i X_j\big] = \frac{1}{n^2}.$$

Bucket Sort: Analysis

- It follows that

$$\begin{aligned}
\mathbb{E}\left[n_0^2\right] &= \sum_{i=1}^{n} \mathbb{E}\left[X_i^2\right] + \sum_{i \neq j} \mathbb{E}\left[X_i X_j\right] \\
&= \sum_{i=1}^{n} \frac{1}{n} + \sum_{i \neq j} \frac{1}{n^2} \\
&= 1 + \frac{n(n-1)}{n^2} \\
&= 2 - \frac{1}{n}
\end{aligned}$$

- As n_0 plays no special role, we also have

$$\mathbb{E}\left[n_j^2\right] = 2 - \frac{1}{n} \quad \text{for all } 0 \leqslant j \leqslant n-1$$

Bucket Sort: Analysis

- We just proved $1 \leqslant \mathbb{E}\left[n_j^2\right] < 2$, so it follows from the analysis on Slide 12 that:

Theorem

The expected running time of BUCKET SORT *is* $\Theta(n)$ *when the* n *input numbers are chosen uniformly and independently at random in* $[0, 1)$.

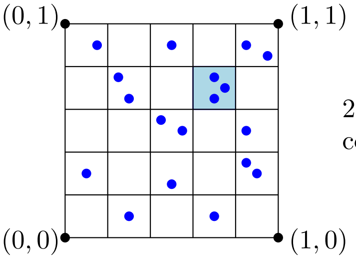
- So BUCKET SORT is very efficient if the input numbers are distributed uniformly at random.
- But in the worst case, for instance if the distribution is skewed and many input numbers fall in the same bucket, it runs in quadratic time.
- So it does not contradict our lower bound.

Bucket Sort: Analysis

- Here "expected" running time has a very different meaning from our analysis of QUICKSORT:
- QUICKSORT is very efficient on worst-case input. It can only be slow if we are extremely unlucky with the random choices of pivot. In practice it never happens.
- On the other hand BUCKET SORT performs poorly on worst case input. In practice it happens, because data is often skewed.

Bucket Sort: Concluding Remarks

- The approach of BUCKET SORT is called *bucketing*.



2D bucketing. The blue bucket contains 3 input points.

- It also applies to multidimensional data, using a uniform grid for instance.

Example

Fixed radius near-neighbor searching. See CSE520, Lecture 1.

Bucket Sort: Concluding Remarks

- Problem: How can we apply BUCKET SORT if the input numbers are not in the interval $[0, 1)$?