Sorting VI

- Comparator- vs. Non-Comparator- Based Sorting
- Binsort
- Bucket Sort
- Stable Sorting
- Lexicographic Sorting
- Radix Sort

Comparator- vs. Non-Comparator- based Sorting

- In a comparator-based sorting algorithm, _____

  _____.

  Examples:

- In a non-comparator-based sorting algorithm, _____

  _____.

- It has been proven that the best worst-case runtime possible for a comparator-based sorting algorithm is _____.

Binsort (Variation 1)

- May be useful when

- What you should know about your data:

Binsort (Variation 1) Example

Binsort (Variation 1) Space & Runtime

Binsort (Variation 2)

- May be useful when

- What you should know about your data:

Binsort (Variation 2) Example

- May be useful when

- What you should know about your data:

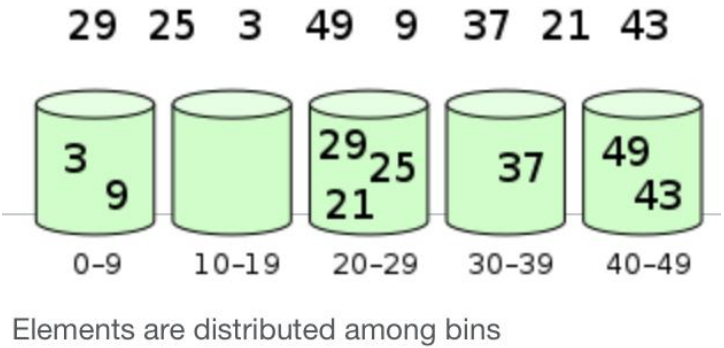
Binsort (Variation 2) Space & Runtime

Bucket Sort

- May be useful when

- What you should know about your data:

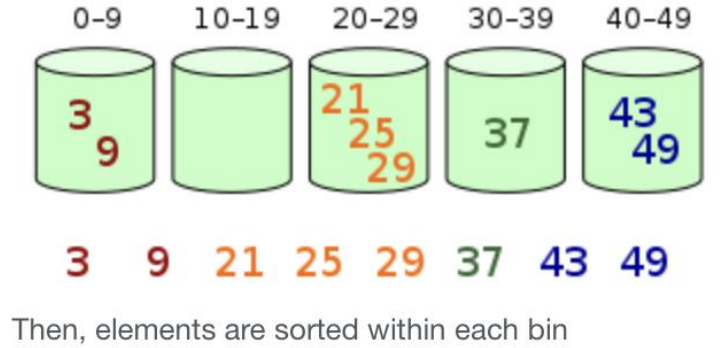
Bucket Sort



Function used in this case to sort items into buckets:

bucket index = floor(*key*/*M****k**)

where *M* = maximum key possible and *k* = number of buckets



Picture from [1]

Bucket Sort Runtime
(N elements, k buckets, Insertion Sort)

- Worst case?

  $O(N^2)$

- Best case?

  $O(N)$

- Average case?

  $O(N + N^2/k + k)$

Stable Sorting

If a sorting algorithm is stable, _____

_____.

Is Bubble Sort stable?

Is Selection Sort stable?

Determine if each of the following are stable or not.

- Bubble Sort
- Selection Sort
- Insertion Sort
- Shellsort
- Heapsort
- Mergesort
- Quicksort
- Binsort (Variation 1)
- Binsort (Variation 2)
- Bucket Sort

Lexicographic Sorting

A lexicographic sorting algorithm puts *d*-tuples into *lexicographic order*.

Lexicographic Sorting

A lexicographic sorting algorithm puts *d*-tuples into *lexicographic order*.

What's a *d*-tuple?

What is *lexicographic order*?

Are these in lexicographic order?

Example 1: Are the following 4-tuples in lexicographic order?

(4, 7, 9, 1)
(4, 7, 9, 2)
(5, 3, 2, 0)
(5, 5, 5, 5)

Example 2: Are the following 5-tuples in lexicographic order?

(0, 1, 2, 3, 4)
(1, 2, 3, 4, 5)
(1, 3, 4, 5, 5)
(1, 2, 3, 6, 6)

Examples of *d*-tuples…

Lexicographic Sort

Algorithm *lexicographicSort*(S)

Input sequence S of d-tuples
Output sequence S sorted in lexicographic order

for $i \leftarrow d$ **downto** 1
    stableSort(S, C_i)

Picture from [2]

Lexicographic Sorting: the RIGHT way and a WRONG way

The Right Way
(Least Significant First)

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)
- (3, **1**, 6) (3, **1**, 8) (8, **3**, 4) (3, **5**, 2)
- (**3**, 1, 6) (**3**, 1, 8) (**3**, 5, 2) (**8**, 3, 4)

The Wrong Way
(Most Significant First)

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (**3**, 5, 2) (**3**, 1, 8) (**3**, 1, 6) (**8**, 3, 4)
- (3, **1**, 8) (3, **1**, 6) (8, **3**, 4) (3, **5**, 2)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)

These are NOT in order!!!

Lexicographic Sorting: the RIGHT way and a WRONG way

The Right Way
(Stable Sorting)

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)
- (3, **1**, 6) (3, **1**, 8) (8, **3**, 4) (3, **5**, 2)
- (**3**, 1, 6) (**3**, 1, 8) (**3**, 5, 2) (**8**, 3, 4)

The Wrong Way
(Unstable Sorting)

- (3, 5, 2) (3, 1, 8) (8, 3, 4) (3, 1, 6)
- (3, 5, **2**) (8, 3, **4**) (3, 1, **6**) (3, 1, **8**)
- (3, **1**, 8) (3, **1**, 6) (8, **3**, 4) (3, **5**, 2)
- (**3**, 5, 2) (**3**, 1, 6) (**3**, 1, 8) (**8**, 3, 4)

These are NOT in order!!!

A couple things...

- Why is it important to use a stable sorting method when sorting each dimension?

A couple things...

- Why is it important to use a stable sorting method when sorting each dimension?

It’s important to maintain the relative order of equal elements so that the previously sorted dimensions remain sorted.

A couple things...

- Why is it important to use a stable sorting method when sorting each dimension?

It's important to maintain the relative order of equal elements so that the previously sorted dimensions remain sorted.

- Why is it important to sort from least significant to most significant dimension?

A couple things...

- Why is it important to use a stable sorting method when sorting each dimension?

It's important to maintain the relative order of equal elements so that the previously sorted dimensions remain sorted.

- Why is it important to sort from least significant to most significant dimension?

It has to do with the way lexicographic order is defined (from left to right). Therefore, to maintain that order, the sorting needs to happen from right to left so that later sorts don't "undo" previous sorts. (i.e. you have to start with the "least significant" item first.)

Runtime Analysis

Given **N** tuples of **d** dimensions and a stable sorting algorithm that sorts **N** elements in time **T(N)**, what is the runtime of a lexicographic sorting algorithm?

Radix Sort: Lexicographic Sorting with _____

```
Algorithm radixSort(S, N):
Input: A sequence S of d-tuples
Output: S, sorted in lexicographic order
for i from d downto 1:
    sort the tuples based on the i^th dimension
    using binsort
end for
```

Radix Sort for Binary Integers

- **Input**: A sequence of **N *b*-**bit integers (e.g. $x = x_{b-1}...x_1x_0$)
- **Overview**: Treat each integer like a tuple with _____ and sort with radix sort
- The **key range** (for buckets) is _____, so k = _____.
- **Analysis:** Runtime is _____
- **What does this mean?**

Radix Sort for Binary Integers

- **Input**: A sequence of **N** ***b***-bit integers (e.g. $x = x_{b-1}...x_1x_0$)
- **Overview**: Treat each integer like a tuple with **b** dimensions and sort with radix sort
- The **key range** (for buckets) is [0, 1], so **k = 2**
- **Analysis:** Runs in **O(b(N + 2)) = O(N)** if **b** is a constant
- **What does this mean?**

We can sort (for example) a sequence of 32-bit integers in LINEAR TIME!!!

References

[1] https://en.wikipedia.org/wiki/Bucket_sort
[2] Tamassia and Goodrich