Generalizing Counting Sort

Demo

We want counting sort to work for non-unique and/or non-consecutive keys!

1. Count the number of occurrences for each key option.
2. Compute the starting indices for each key option from the counts array.
3. Move through the original items in order. For each `[item, key]` do:
   a. Get the correct index for result by checking the index array for `key`
   b. Copy `item` into the result using this index
   c. Increment the index array for `key`
4. Copy items back to initial array (if needed)

When poll is active, respond at **PollEv.com/kevinl**

What is the runtime for counting sort on N items with an alphabet with radix (size) R? Treat R as a variable, not a constant.

- $\Theta(N)$
- $\Theta(R)$
- $\Theta(N + R)$
- $\Theta(NR)$
- Not sure

Counting Sort

Runtime and memory use of **Θ(N + R)**! N = # of items, R = radix of alphabet

We are able to beat comparison sort by **avoiding binary compares**.

If N >= R, we expect reasonable performance. If N is much bigger than R, then R can become negligible.

Empirical experiments are needed to compare to Quicksort on practical inputs.

Input is restricted to alphabetic (finite radix) keys → **we can't sort items with non-alphabetic keys, like Strings!**

Q LSD Radix Sort

Why is it important for the correctness of LSD radix sort that counting sort is stable? Give an example of what could go wrong if it were not stable.

Index	Key	Name
0	22	Stitch
1	12	Gantu
2	31	Nani
3	23	Lilo
4	11	David

→

Index	Key	Name
0	**3**1	Nani
1	**1**1	David
2	**2**2	Stitch
3	**1**2	Gantu
4	**2**3	Lilo

→

Index	Key	Name
0	**1**1	David
1	**1**2	Gantu
2	**2**2	Stitch
3	**2**3	Lilo
4	**3**1	Nani

LSD Radix Sort Summary

Use counting sort on each index, right to left. **Now we can sort non-alphabetic keys** that consist of alphabetic keys!

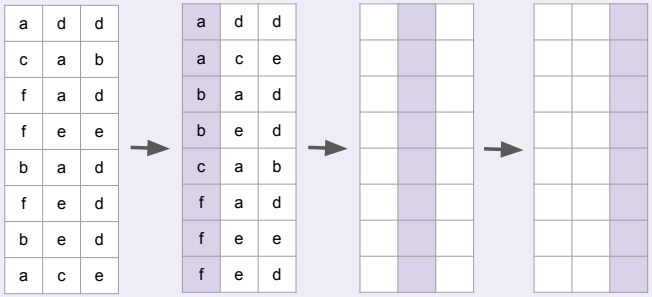
Runtime: Θ(WN + WR), Memory use: Θ(N + R) N = # of items, R = radix, W = width

If R is very small compared to N and W we can think of it as negligible.

It's annoying that the runtime depends on the length of the longest key → 🤔

Q MSD Radix Sort

Suppose we sort each digit index, left to right. Will we arrive at the correct result? Why?



Q MSD Radix Sort Runtime

What is the best case runtime of MSD sort? (in terms of N, W, R)?

What type of input leads to this best case?

What is the worst case runtime of MSD sort? (in terms of N, W, R)?

What type of input leads to this worst case?

N = # of items, R = radix, W = width

Analysis of MSD Radix Sort

Runtime - Best case: Θ(N + R), Worst case: Θ(WN + WR)
Memory usage - Θ(N + WR)

Think about the runtime of MSD radix sort by considering the **number of characters that must be examined**.

Long strings are rarely random in practice → may need specialized algorithms

Random (sublinear)	Non-random with duplicates (nearly linear)	Worst case (linear)
1EIO402	are	1DNB377
1HYL490	by	1DNB377
1RQZ572	sea	1DNB377
2HXE734	seashells	1DNB377
2IYE230	seashells	1DNB377
2XQR846	sells	1DNB377
3CDB573	sells	1DNB377
3CVP720	she	1DNB377
3IGJ319	she	1DNB377
3KNA382	shells	1DNB377
3TAV879	shore	1DNB377
4CQP781	surely	1DNB377
4QGI284	the	1DNB377
4YHV229	the	1DNB377

From Algorithms, 4th edition by Sedgewick and Wayne