



Radix Sort

- Unlike other sorting methods, radix sort considers the structure of the keys
- Assume keys are represented in a base M number system ($M = \text{radix}$), i.e., if $M = 2$, the keys are represented in binary

$$9 = \begin{array}{cccc} \text{weight} & 8 & 4 & 2 & 1 \\ \hline & 1 & 0 & 0 & 1 \\ \hline & 3 & 2 & 1 & 0 \\ \text{bit \#} & & & & \end{array} \quad (b = 4)$$

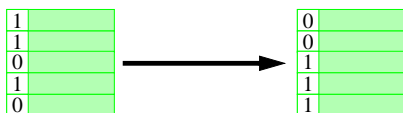
- Sorting is done by comparing bits in the same position
- Extension to keys that are alphanumeric strings



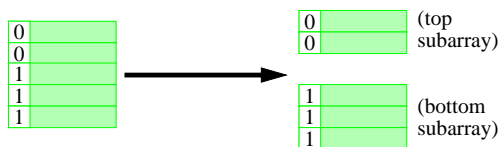
Radix Exchange Sort

Examine bits from *left to right*:

1. Sort array with respect to leftmost bit



2. Partition array



3. Recursion

- recursively sort top subarray, ignoring leftmost bit
- recursively sort bottom subarray, ignoring leftmost bit

Time: $O(b N)$



Radix Exchange Sort

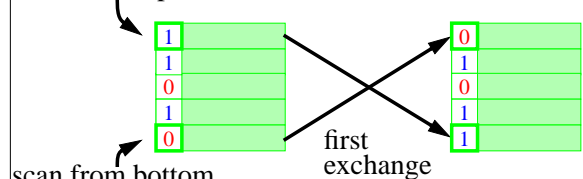
How do we do the sort from the previous page?
Same idea as partition in [Quicksort](#).

repeat

scan top-down to find key starting with 1;
scan bottom-up to find key starting with 0;
exchange keys;

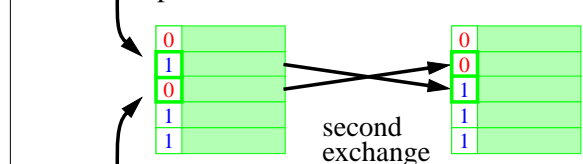
until scan indices cross;

scan from top



scan from bottom

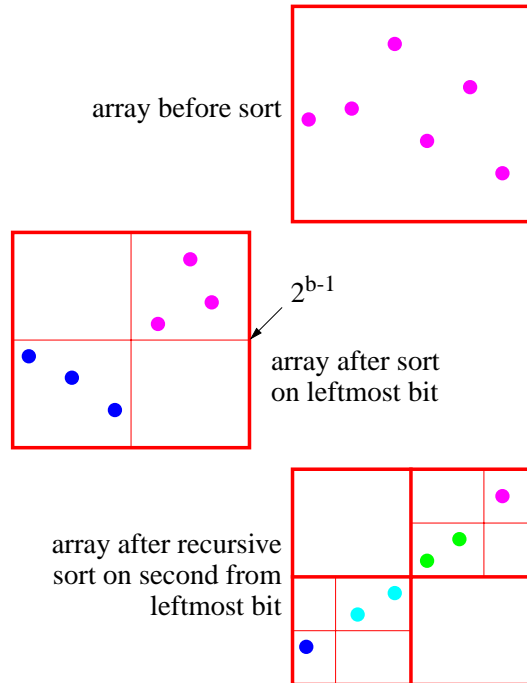
scan from top



scan from bottom



Radix Exchange Sort



Radix Exchange Sort vs. Quicksort

Similarities

- both partition array
- both recursively sort sub-arrays

Differences

- *Method of partitioning*

- radix exchange divides array based on greater than or less than 2^{b-1}
- quicksort partitions based on greater than or less than some element of the array

- *Time complexity*

- Radix exchange $O(bN)$
- Quicksort average case $O(N \log N)$
- Quicksort worst case $O(N^2)$

Straight Radix Sort

Examines bits from *right* to *left*

```

for k := 0 to b-1
    sort the array in a stable way,
    looking only at bit k

```

First,
sort
these

Next, sort
these digits

Last, sort these.

Figure 1 shows four 10x3 grids, labeled 1 through 4, representing the evolution of a 3x3 grid over 10 time steps. Each grid has three columns. The first column is always red. The second and third columns are blue in some cells. The number of blue cells increases from 1 in grid 1 to 10 in grid 4. The blue cells are arranged in a pattern that suggests a wave or front moving from left to right.

0	1	0
0	0	0
1	0	1
0	0	1
1	1	1
0	1	1
1	0	0
1	1	0

0	1	0
0	0	0
1	0	0
1	1	0
1	0	1
0	0	1
1	1	1
0	1	1

0	0	0
1	0	0
1	0	1
0	0	1
0	1	0
1	1	0
1	1	1
0	1	1

0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Note order of these bits after sort.

I forgot what it means to “sort in a stable way”!!!

In a stable sort, the initial relative order of equal keys is unchanged.

For example, observe the first step of the sort from the previous page:

The diagram illustrates the transformation of a 3D volume into a 2D feature map. On the left, a 3D volume of size 8x8x8 is shown, with a central 4x4x4 region highlighted in red. An arrow points to the right, where a 2D feature map of size 8x8 is shown. The feature map has a central 4x4 region highlighted in red, corresponding to the original 3D volume's central region.

Note that the relative order of those keys ending with 0 is unchanged, and the same is true for elements ending in 1

The Algorithm is Correct (right?)

- We show that any two keys are in the correct relative order at the end of the algorithm
- Given two keys, let k be the leftmost bit-position where they differ

0	1	0	1	1
---	---	---	---	---

0	1	1	0	1
---	---	---	---	---

 k

- At step k the two keys are put in the correct relative order
- Because of stability, the successive steps do not change the relative order of the two keys

dnc

155



For Instance,

Consider a sort on an array with these two keys:

0	1	0	1	1
---	---	---	---	---

0	1	1	0	1
---	---	---	---	---

 k

0	1	1	0	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1

It makes no difference what order they are in when the sort begins.

When the sort visits bit k , the keys are put in the correct relative order.

0	1	0	1	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1

Because the sort is stable, the order of the two keys will not be changed when bits $> k$ are compared.

0	1	0	1	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1
0	1	1	0	1

dnc

156



Radix sorting can be applied to decimal numbers

First, sort these digits

Next, sort these digits

Last, sort these.

0	3	2	0	3	1	0	1	5	0	1	5
2	2	4	0	3	2	0	1	6	0	1	6
0	1	6	2	5	2	1	2	3	0	3	1
0	1	5	1	2	3	2	2	4	0	3	2
0	3	1	2	2	4	0	3	1	1	2	3
1	6	9	0	1	5	0	3	2	1	6	9
1	2	3	0	1	6	2	5	2	2	2	4
2	5	2	1	6	9	1	6	9	2	5	2

Note order of these bits after sort.

Voila!

dnc

157



Straight Radix Sort Time Complexity

for $k := 0$ to $b-1$

sort the array in a *stable* way,
looking only at bit k

Suppose we can perform the stable sort above in $O(N)$ time. The total time complexity would be

$O(bN)$.

As you might have guessed, we can perform a stable sort based on the keys' k^{th} digit in $O(N)$ time.

The method, you ask? Why it's **Bucket Sort**, of course.



dnc

158



Bucket Sort

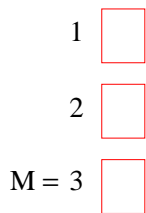
- N numbers
- Each number $\in \{1, 2, 3, \dots, M\}$
- Stable
- Time: $O(N + M)$

For example, $M = 3$ and our array is:



(note that there are two “2”s and two “1”s)

First, we create M “buckets”



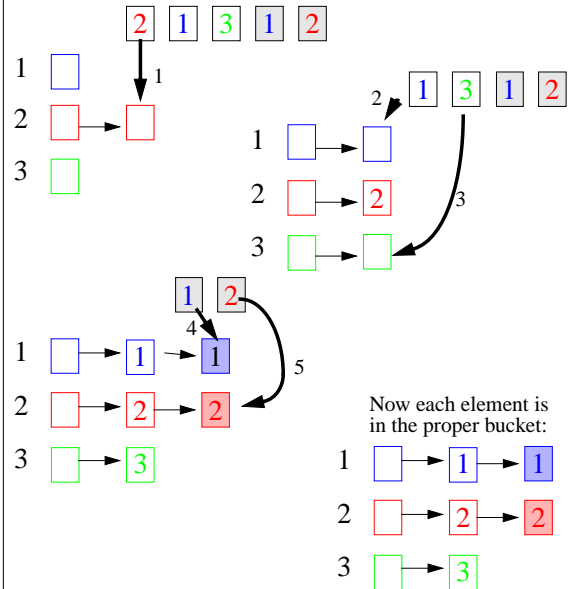
dnc

159



Bucket Sort

Each element of the array is put in one of the M “buckets”



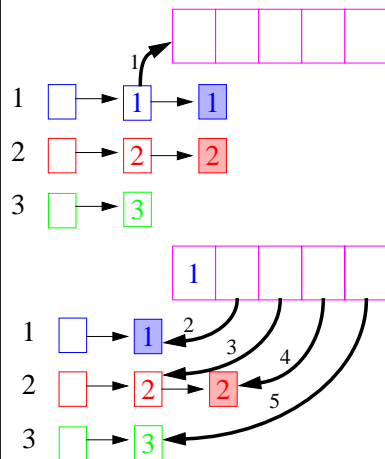
dnc

160



Bucket Sort

Now, pull the elements from the buckets into the array



At last, the sorted array (sorted in a stable way):



dnc

161

