Radix sort

Radix Sort
Idea

- Treats each data to be sorted as a character string.

- It is not using comparison, i.e. no comparison between the data is needed.

- In each iteration:
  - Organize the data into groups according to the next character in each data.
  - The groups are then “concatenated” for next iteration.

Radix Sort Example

0123, 2154, 0222, 0004, 0283, 1560, 1061, 2150

Original integers

Radix Sort
Example

(156**0**, 215**0**) (106**1**) (022**2**) (012**3**, 028**3**) (215**4**, 000**4**)

Grouped by forth digit

Radix Sort Example

1560, 2150, 1061, 0222, 0123, 0283, 2154, 0004

Combined

Radix Sort Example

(00**0**4) (02**2**2, 01**2**3) (21**5**0, 21**5**4) (15**6**0, 10**6**1) (02**8**3)

Grouped by third digit

Radix Sort Example

0004, 0222, 0123, 2150, 2154, 1560, 1061, 0283

Combined

Radix Sort
Example

(0**0**04, 1**0**61) (0**1**23, 2**1**50, 2**1**54) (0**2**22, 0**2**83) (1**5**60)

Grouped by second digit

Radix Sort Example

0004, 1061, 0123, 2150, 2154, 0222, 0283, 1560

Combined

Radix Sort Example

(**0**004, **0**123, **0**222, **0**283) (**1**061, **1**560) (**2**150, **2**154)

Grouped by first digit

Radix Sort
Example

0004, 0123, 0222, 0283, 1061, 1560, 2150, 2154

Combined, sequence are sorted

Radix Sort Implementation

```
void radix_sort(vector<int>& arr)
{
    std::vector<int> vec = arr;
    int power_of_ten = 1;
    std::vector<std::vector<int>> buckets(10);
    for (int pow = 0; pow < 10; ++pow)
    {
        for (auto elem : vec)
        {
            buckets[elem / power_of_ten % 10].push_back(elem);
        }
        vec.clear();
        for (int i = 0; i < buckets.size(); ++i)
        {
            vec.insert(vec.end(), buckets[i].begin(), buckets[i].end());
            buckets[i].clear();
        }
        power_of_ten *= 10;
    }
    std::copy(vec.begin(), vec.end(), arr.begin());
}
```

Radix Sort Analysis

- For each iteration:
  - We go through each item once to place them into group.
  - Then go through them again to concatenate the groups.
  - Complexity is $O(n)$.

- Number of iterations is d, the maximum number of digits (or maximum number of characters).

- Complexity is thus $O(d \cdot n)$.