


Exercise 2 (for grade) ~ Wednesday, September 14, 2022 ~ CPSC 535 Fall 2022

Write one submission for your entire group, and write all group members' names on that submission. Turn in your submission before the end of class. The  symbol marks where you should write answers.

Recall that our recommended problem-solving process is:

1. **Understand** the problem definition. What is the input? What is the output?
2. **Baseline** algorithm for comparison
3. **Goal** setting: improve on the baseline how?
4. **Design** a more sophisticated algorithm
5. **Inspiration** (if necessary) from patterns, bottleneck in the baseline algorithm, other algorithms
6. **Analyze** your solution; goal met? Trade-offs?

Follow this process for each of the following computational problems. For each problem, your submission should include:

- a. State the input variables and what are the output variables
- b. Pseudocode for your baseline algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
- a. The Θ -notation time complexity of your baseline algorithm, with justification.

and if you manage to create an improved algorithm:

- c. Answer the question: how is your improved algorithm different from your baseline; what did you change to make it faster?
- d. Pseudocode for your improved algorithm, that needs to include the data type and an explanation for any variable other than input and output variables
- a. The Θ -notation time complexity of your improved algorithm, with justification.

Today's problems are:

1.

List the following functions according to their order of growth from the lowest to the highest:

2^{2n} , $0.001n^4 + 3n^3 + 1$, $\ln^2 n$, $\sqrt[3]{n}$

2.

Consider the problem of sorting an array of n floating point numbers which are in range from 0 to 1 and have at most two digits after the decimal point. The elements in the array are not necessarily uniformly distributed across the range $[0..1]$. Write a linear time $O(n)$ algorithm to sort these n values.

Example:

Input : arr[] = { 0.89, 0.7, 0.56, 0.5, 0.65, 0.6, 0.12, 0.34, 0, 0.34, 0.4 }

Output : 0 0.12 0.34 0.34 0.4 0.5 0.56 0.6 0.65 0.7 0.89

3. (*Forming the largest number.*)

Using Figure 8.3 (below) as a model, illustrate the operation of RADIX-SORT on the following list of English words: BAD, BUT, DAG, FOG, HOT, IAN, NOW, PUT, RIG, TUG, VIO, WAS, TOP, CAT, ARC, OUT.

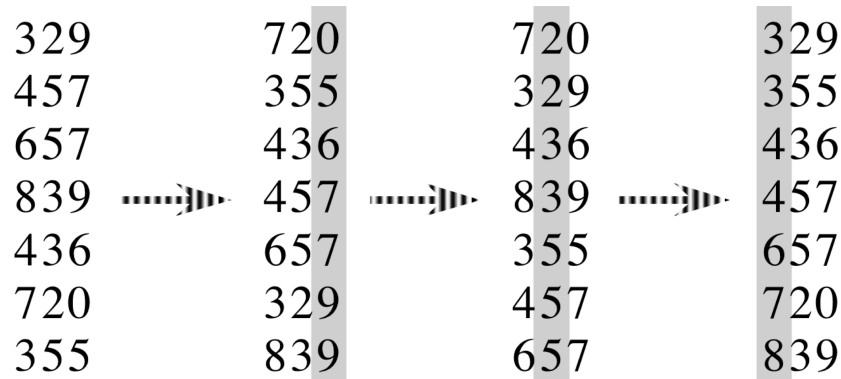


Figure 8.3

Names

Write the names of all group members below.

✘ Joseph Maa, Daniel Cao, Rosa Cho

Exercise 1: Solve and provide answer

✘ $\ln^2 n$ ($O(\log(n))$), $\sqrt[3]{n}$ ($O(n^{1/3})$), $0.001n^4 + 3n^3 + 1$ ($O(n^4)$), 2^{2n} ($O(4^n)$) Correct

In increasing order, the order of growth of these functions go from logarithmic time, to a smaller polynomial time, to a larger polynomial time, finally to exponential time.

Exercise 2: Solve and provide answer



Understand

Input/Output

Read and discuss the problem statement, then answer: **Describe the input and output definition in your own words.**

✗ The input would be an array of floating point numbers $[0, 1]$ with up to two decimals with n being the number of elements in the array.

The output would be the array of n floating point numbers $[0, 1]$ with up to two decimals after being sorted.

Examples

Write out a concrete input value for the problem, and the concrete correct output. Repeat this at least once (so there are at least two input/outputs).



Input: [0.1, 0.22, 0.32, 0.51, 0.62, 0.16, 0.24, 0.84, 0.01, 0.31, 0.24]

Output: [0.01, 0.1, 0.16, 0.22, 0.24, 0.24, 0.31, 0.32, 0.51, 0.62, 0.84]

Input: [0.32, 0.54, 0.87, 0.21, 0.12, 0.73, 0.37, 0.46, 0, 0.21, 0.22]

Output: [0, 0.12, 0.21, 0.21, 0.22, 0.32, 0.37, 0.46, 0.54, 0.73, 0.87]

Corner Cases

A corner case is a special case of input that is particularly unusual or challenging. Are there corner cases for this problem? If so, describe them.

✗ We have to be careful that 1 is inclusive in the end range, so I to make sure that the range included a value for 1. Also, if the sort algorithm used in bucket sort is not stable, then the algorithm will not result in the same final ordering of the initial input values provided. This could potentially be problematic in the case where there are multiple non-unique values in the array (i.e. $[0.8, 0.8, 0.8]$). If such values were mapped to individuals like in the example below such that the values were provided by a key: value pair (i.e. {“Kelly”: 0.8, “George”: 0.8, “Tom”: 0.8}), the resulting output of names may not be the same depending on whether the sort was stable.

Use Case

Think of a specific use case for an algorithm that solves this problem. What kind of feature could it support in practical real-world software?

✖ If users were told to input a cash value into an input box and the usernames and cash values they input were stored in a database, we could use a similar algorithm to sort the usernames by cash values provided in a relatively time-efficient way. Such a sorting algorithm could possibly improve run-time efficiency in this particular use case. However, testing should be done empirically in order to test on different actual inputs.

Similar Problems

Does this problem resemble any other problems you remember? If so, which problem? Does that problem have an efficient algorithm? If so, what is the run time of that algorithm, in Θ -notation?

✖ This reminded me of the Top K frequent elements problem. That problem is also solved using a bucket sort, running in linear time. $O(n)$. However, I solved the problem using a priority queue, which by comparison runs in $O(n \log(n))$ time, which was more inefficient.

```
class Solution {
public:
    //given a vector of size k
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map <int, int> f;

        //our vector is then initialized with a set of values given as unordered_map or key-value pairs
        for (int i = 0; i < nums.size(); ++i) {
            f[nums[i]]++;
        }

        priority_queue<pair<int, int>> p;
        vector<int> res;

        //the key-value pairs are then compared with each other before being placed into “buckets” or subdivisions of the
        vector
        for (auto it = f.begin(); it != f.end(); ++it) {
            p.push(make_pair(it->second, it->first));
        }

        //compared values are then placed into a new vector and their temporary bucket containers are “popped”
        for (int i = 0; i < k; ++i) {
            res.push_back(p.top().second);
            p.pop();
        }

        //a newly sorted vector is returned
        return res;
    }
};
```

Baseline

Design Overview

Design a naïve algorithm that solves the problem. This should be a simple algorithm that is easy to describe and understand. First, describe the basic idea of your algorithm in 1-2 sentences.

✘ We create n empty buckets or lists and insert each element of the array, `arr[i]`, into `bucket[n * arr[i]]`. We sort the individual buckets using insertion sort and concatenate all sorted buckets.

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

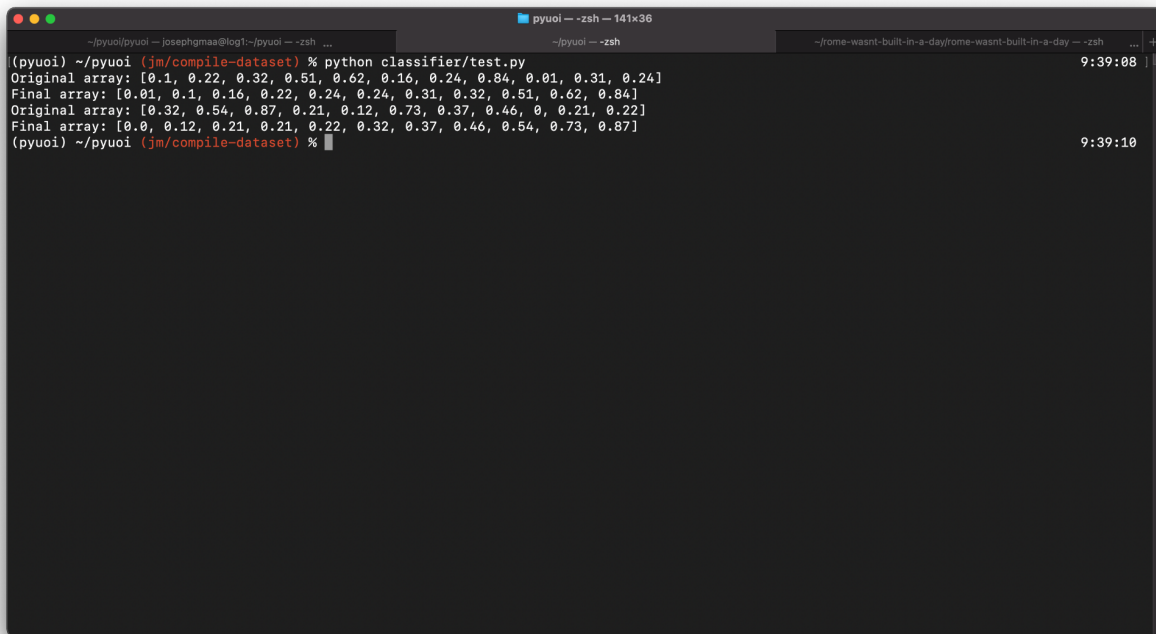
✘

```
def bucket_sort(arr: list[float]) -> list[float]:
    buckets = [0 for _ in range(101)]
    for num in arr:
        buckets[int(num*100)] += 1
    res = []
    for i, bucket in enumerate(buckets):
        if bucket:
            res.extend([i/100 for _ in range(bucket)])
    return res
```

Correct

```
def main():
    print(f"Original array: {[ 0.1, 0.22, 0.32, 0.51, 0.62, 0.16, 0.24, 0.84, 0.01,
0.31, 0.24 ]}")
    print(f"Final array: {bucket_sort([ 0.1, 0.22, 0.32, 0.51, 0.62, 0.16, 0.24, 0.84,
0.01, 0.31, 0.24 ])}")
    print(f"Original array: {[ 0.32, 0.54, 0.87, 0.21, 0.12, 0.73, 0.37, 0.46, 0, 0.21,
0.22 ]}")
    print(f"Final array: {bucket_sort([ 0.32, 0.54, 0.87, 0.21, 0.12, 0.73, 0.37, 0.46,
0, 0.21, 0.22 ])}")

if __name__ == "__main__":
    main()
```

A terminal window with a dark background and light text. It shows the execution of a Python script. The prompt is '(pyuoi) ~/pyuoi (jm/compile-dataset) %'. The command is 'python classifier/test.py'. The output shows two iterations of an array being sorted. In the first iteration, the original array is [0.1, 0.22, 0.32, 0.51, 0.62, 0.16, 0.24, 0.84, 0.01, 0.31, 0.24] and the final array is [0.01, 0.1, 0.16, 0.22, 0.24, 0.24, 0.31, 0.32, 0.51, 0.62, 0.84]. In the second iteration, the original array is [0.32, 0.54, 0.87, 0.21, 0.12, 0.73, 0.37, 0.46, 0, 0.21, 0.22] and the final array is [0.0, 0.12, 0.21, 0.21, 0.22, 0.32, 0.37, 0.46, 0.54, 0.73, 0.87]. The prompt returns to '(pyuoi) ~/pyuoi (jm/compile-dataset) %'. The terminal has three tabs at the top: '~/pyuoi/pyuoi -- josephgmaa@log1~/pyuoi -- ~zsh ...', '~/pyuoi -- ~zsh', and '~/rome-wasnt-built-in-a-day/rome-wasnt-built-in-a-day -- ~zsh ...'. The window title is 'pyuoi -- zsh -- 141x36'. The time '9:39:08' is shown in the top right corner of the terminal area, and '9:39:10' is shown at the bottom right.

//once each element within a bucket has been compared, the element will be determined:
// 1) if they are less or equal to their neighbor, placed in spot before for remain in the same place
// 2) if they are more than their neighbor, place in a spot after the neighbor until the process is re-iterated
and, when compared to a di they are determined to be more than a different neighbor
// All this is added and combined in a different array called res

Efficiency

What is the Θ -notation time complexity of your baseline algorithm? Justify your answer.

✗ The run-time complexity of this baseline algorithm is $O(n)$ since initialization of the buckets takes linear time, filling the buckets takes linear time, and filling in the resulting array also takes linear time. (In Python this might not be strictly true because we can't pre-initialize the res list, which might cause it to have to dynamically resize, but we can assume it to be true for languages with fixed-size arrays).

Improvement (Optional)

Now, steps 3-6 involving creating a better algorithm. This part is optional. Only attempt this part if you finished the earlier parts and still have time.

Goal

What aspect of your baseline algorithm do you want to improve? Time efficiency, or something else?

✗ n/a

Design Overview

Design a better algorithm for the same problem. First, describe the basic idea of your algorithm in 1-2 sentences.

☒ n/a

Pseudocode

Now, write clear pseudocode for your improved algorithm.

☒ n/a

Efficiency

What is the Θ -notation time complexity of your improved algorithm? Justify your answer.

☒ n/a

Analysis

Did you meet your goal? Why or why not?

☒ Yes, the algorithm runs as expected.

Using Figure 8.3 (below) as a model, illustrate the operation of RADIX-SORT on the following list of English words: BAD, BUT, DAG, FOG, HOT, IAN, NOW, PUT, RIG, TUG, VIO, WAS, TOP, CAT, ARC, OUT.

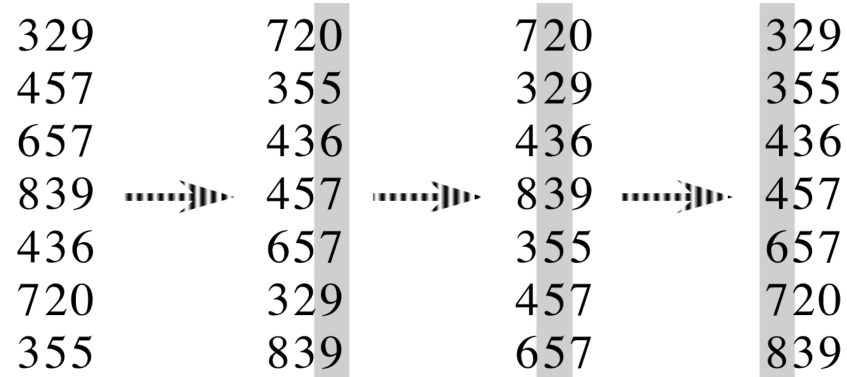
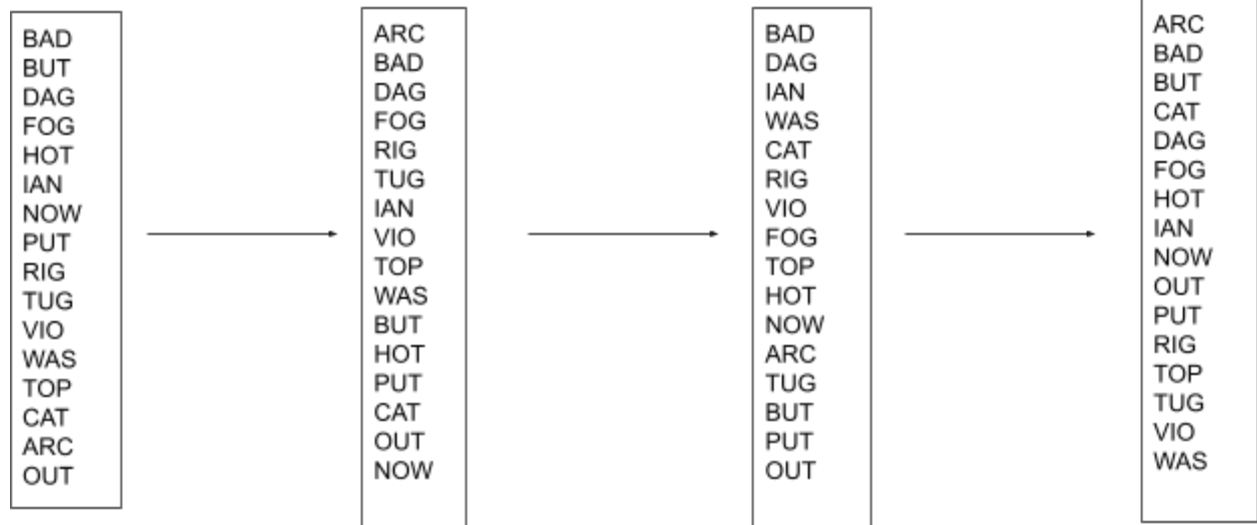


Figure 8.3

Exercise 3: Solve and provide answer



Input:



Output:

Correct

Code:

```
# (input), given an array of 16 elements, we are expected to sort them using the RADIX sort
>>> arr = ["BAD", "BUT", "DAG", "FOG", "HOT", "IAN", "NOW", "PUT", "RIG", "TUG",
"VIO", "WAS", "TOP", "CAT", "ARC", "OUT"]

//the third letter of each element is compared to their neighbors and sorted accordingly
>>> sorted(arr, key=lambda x: x[2])
['ARC', 'BAD', 'DAG', 'FOG', 'RIG', 'TUG', 'IAN', 'VIO', 'TOP', 'WAS', 'BUT', 'HOT', 'PUT', 'CAT',
'OUT', 'NOW']
//each element is then added to a new array
>>> arr2 = ['ARC', 'BAD', 'DAG', 'FOG', 'RIG', 'TUG', 'IAN', 'VIO', 'TOP', 'WAS', 'BUT', 'HOT',
'PUT', 'CAT', 'OUT', 'NOW']

//the second letter of each word is then compared to their neighbor and sorted accordingly
>>> sorted(arr2, key= lambda x: x[1])
['BAD', 'DAG', 'IAN', 'WAS', 'CAT', 'RIG', 'VIO', 'FOG', 'TOP', 'HOT', 'NOW', 'ARC', 'TUG',
'BUT', 'PUT', 'OUT']
//each element is then added to a new array
>>> arr3 = ['BAD', 'DAG', 'IAN', 'WAS', 'CAT', 'RIG', 'VIO', 'FOG', 'TOP', 'HOT', 'NOW', 'ARC',
'TUG', 'BUT', 'PUT', 'OUT']

# (output) the fully sorted array is returned as each word is finally sorted alphabetically
>>> sorted(arr3, key=lambda x: x[0])
['ARC', 'BAD', 'BUT', 'CAT', 'DAG', 'FOG', 'HOT', 'IAN', 'NOW', 'OUT', 'PUT', 'RIG', 'TOP',
'TUG', 'VIO', 'WAS']
```

*Python sort is guaranteed to be stable.