Grade: 7.5/9

(+0.5 points): bonus for a 2-people team

Exercise 3 (for grade) ~ Wednesday, September 21, 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:

7. State are the input variables and what are the output variables
8. 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:

b. Answer the question: how is your improved algorithm different from your baseline; what did you change to make it faster?

c. 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.

A wiggle sequence is a sequence where the differences between successive numbers strictly alternate between positive and negative. The first difference (if one exists) may be either positive or negative. A sequence with one element and a sequence with two non-equal elements are trivially wiggle sequences. For example, [1, 7, 4, 9, 2, 5] is a wiggle sequence because the differences (6, -3, 5, -7, 3) alternate between positive and negative. In contrast, [1, 4, 7, 2, 5] and [1, 7, 4, 5, 5] are not wiggle sequences. The first is not because its first two differences are positive, and the second is not because its last difference is zero.
Design an algorithm that reads *n* numerical values from the user, one by one, and inserts these values as a wiggle sequence into a stack in O(*n*) time. You can use one additional stack or one additional queue, to store values temporarily but no other data structures. For stack and queues, only push/enqueue and pop/dequeue are allowed, and one cannot access any elements other than the top/front.
Note: You can assume that no two consecutive values are the same, but one can have identical values, just not consecutive ones.

2.

Given an array of positive integers, arrange them in a way that yields the largest ODD value.
Example 1:
If the given numbers are {1, 34, 3, 98, 9, 76, 45, 4}, then the arrangement 998764543431 (coming from 9 98 76 45 4 34 3 1) gives the largest odd value.
Example 2:
If the given numbers are {53, 516, 648, 60}, the arrangement 6486051653 gives the largest odd value.
Example 3:
If the given numbers are {54, 546, 548, 60}, then ~~arrangement 6054854654 gives the largest odd value.~~ The output is NONE
.
Your goal: baseline O(n!) time, improved to O(n log n).

3.

Design an algorithm that, given a set S of n integers, identifies all elements that repeat.
Your goal: baseline O(n^2) time, improved to either O(n log n) worst-case time or O(n) expected time.

Names

Write the names of all group members below.

❎Rosa Cho, Joseph Maa

Exercise 1: Solve and provide answer

❎ Joseph Maa, Rosa Cho

1. Input and Output variables.
    a. Input: sequence of numbers
    b. Output: a stack or a queue
2. Overview of the algorithm
    a. We realize that a given wiggle sequence contains smaller wiggle sequences within it. Moreover, a wiggle sequence can be flipped and maintain the wiggle invariance. Hence, if we run into an input that cannot currently be added to the wiggle sequence, we can invert the current stack until we can add it to the current sequence. Lastly, we should maintain that the stacks have to remain wiggle sequences. If we cannot add the current num to the stack, the input is invalid.
3. Pseudocode

```
def generate_wiggle_sequence(iterator) -> stack[int]:
    Stack1, stack2 = [], []
    M, n = -1, -1 # boolean flags for increasing or decreasing num expected for wiggle, 1 for
increasing number expected, 0 for decreasing number expected
    while iterator:
        num = *iterator
        if not stack1 or :
            stack1.push(num)
```

(-2 points): incomplete pseudocode

4. The Θ-notation time complexity of your baseline algorithm, with justification.
   This should run in O(n) time since it only needs one pass through the input.

Exercise 2: Solve and provide answer

☒ Rosa Cho, Joseph Maa

Understand

Input/Output

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

☒The Input is an array of positive integers. The Output is a sequence that has been sorted from largest odd value to smallest value a la 9_8_7_5_3…1 if the _ represents a following digit either in the hundreds/tens/ones place or, in some cases, no following digits at all..

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).

☒ Example 1: If the given input is {32, 59, 6, 1, 4}, then the arrangement 6594321 gives us the largest value in terms of first digit read, not the number as a whole.
Example 2: If the given input is {87, 9, 56, 73, 4, 22, 10}, then the arrangement 987735642210 gives us the largest value in terms of first digit read, not the number as a whole.

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.

☒Given that our inputs are supposed to be arrays with positive integers, I would not expect floats, negative numbers, irrational numbers, or anything that is contrary to what is given. However, if there are repeated integers or any number that exceeds 10^18.

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?

☒Since this is a comparison sort program, one thing that comes to mind would be a database that has a restricted mode of access as well as limited memory to use. So, in order to edit and sort that database, one would need to use this comparison sort algorithm.

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, inΘ-notation?

❎There was an algorithm that required one to rearrange an input sequence of {99,88,34, 68, 39} in a descending order but using a quick sort so that the run time was O(n*log n) and the output sequence was {99,88,68,39,34}.

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.

❎The naïve algorithm would use backtracking to generate all possible combinations of numbers in O(n!) time and obtaining the largest possible value from all possible combinations of numbers.

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

```
❎def find_largest_odd(arr: list[int]) -> int:
    “”””
    >>> find_largest_odd[1, 34, 3, 98, 9, 76, 45, 4]
    998764543431
    ”””””

    cur_max = 0
    Def helper(cur: int, remaining: list[int]) -> int:
        if not remaining:
            return cur
        res = 0
        for num in remaining:
            copy = remaining.copy()
            copy.remove(num)
            res = max(res, helper(cur * 10 * ceil(log_10(num)) + num)
        return res

    return helper(0, arr)
```

I do not understand how this algorithm works

Efficiency

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

☒This would be O(n!) since backtracking generally goes through all permutations of the array.

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?

☒We can improve the time efficiency of the algorithm by sorting the list using lexicographical sort.

Design Overview

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

☒We sort lexicographically in descending order. We then find the smallest instance of an odd number, then place that at the lowest position in the number. We remove the number from the sorted array. Then we concatenate the rest of the sorted array back to the smallest odd number.

Pseudocode

Now, write clear pseudocode for your improved algorithm.

```
☒def find_largest_odd(arr: list[int]) -> int:
        arr = [str(i) for i in arr]                    # cast to str to sort lexicographically
        arr.sort(inplace=True)                         # sort inplace
        smallest_odd = -1                              # smallest odd value is given designation of -1
        for num in arr[::-1]:                          # each element in given array with size num is examined
                if int(num) % 2 == 1:                  # if each value is an odd number
                        smallest_odd = num             #value to element's place in sequence
        assert smallest_odd != -1 "The array contains no odd numbers." #return message if none detected
        arr.remove(smallest_odd)                       #remove value set beforehand
        return int("".join(arr)+smallest_odd)          #return output number
```

Correct

Efficiency

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

☒This algorithm runs in O(nlog(n)) time since the sort is the slowest part of the algorithm.

Analysis

Did you meet your goal? Why or why not?

☒ Yes, we improved the runtime

Exercise 3: Solve and provide answer

☒ Rosa Cho, Joseph Maa

Understand

Input/Output

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

☒ The input is a set of integers with the output being a set that shows all (or any) repeated elements from that set.

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).

☒

Example 1: Input: arr = [5, 4, 7, 3, 9, 23, 7], n = 7 Output: 7	Example 2: Input: arr = [12, 75, 13, 7, 13, 12, 2, 10, 8], n = 9 Output: 13, 12

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.

☒One example of a corner case that can be applicable to this situation would be dealing with multiple digits of a very a value (i.e. 10^18). It would certainly take longer to process.

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?

☒Sometimes too much unusable/junk data can fill up much needed storage space so, even though this algorithm focuses on comparing a numerical sequence for repeated elements, a similar principle of detecting and deleting duplicate data is a good comparison.

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?

Another similar problem would be an algorithm that required one to use a merge sort because, instead of a given array, it was a given queue and repeated elements were placed at the front of the queue rather than outputted as their own individual numbers. The reason why repeated elements were placed at the front is because, given multiple inputs and multiple outputs, each output would then be compared and, whichever queue had the most repeated elements, would then be deleted.

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 use a dictionary to hold a count of all the values in the set, then we iterate through the dictionary again to get all values that have more than 1 instance in the set.

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

```
def find_multiple_instances(arr: list[int]) -> set:        #given a set
    counts = {}
    for num in arr:                                        #going through each element in size n array
        if not counts[num]:                                #if element is *not* a duplicate
            counts[num] = 1                                #count(i.e. number of elements of the same value) = 1
        else:
            counts[num] += 1                               #count(i.e. number of elements of the same value) += 1 if duplicates found
    res = set()
    for key, val in counts.items():                        #duplicates within the input set will be returned as an output
        if val > 1:
            res.add(key)
    return res
```

Correct

Efficiency

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

This algorithm runs in O(n) time since it iterates over all the values in the array to get the counts, then once more over the dictionary which is bounded above by n in order to add the keys with multiple instances.

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?

Design Overview

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

Pseudocode

Now, write clear pseudocode for your improved algorithm.

Efficiency

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

Analysis

Did you meet your goal? Why or why not?