Grade: 8/9

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:

a. State are 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.

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.

☒ Dhruti Dilipbhai Patel
Hetal Hiteshkumar Patel
John Tu

Exercise 1: Solve and provide answer

☒

(A wiggle list must contain a sequence of differences between consecutive numbers where it alternates between positive and negative strictly. This algorithm shall be implemented as a stack.)

```
def wiggle_list(n):
    Let wiggle_stack = []
    Let temp_stack = []
    // First, ensure that the input list is a wiggle sequence, meaning it must alternate between
    // positive and negative. Also, the difference between two consecutive elements can't be zero.
    For i=1 to n do
        Let m = input(i) // Get the user input.
        If i > 1: // This condition only applies after the first element is inserted.
            Let temp_val = wiggle_stack.top() // Get the top-most element.
            temp_stack.push(m-temp_val) // Insert the difference into stack.
        wiggle_stack.push(m) // Insert the element at the front of queue.
    // Now verify if this is a valid wiggle sequence.
    // Get the length of the temporary stack first.
    While !(temp_stack.empty()):
        If temp_stack.top() == 0:
            Print "The difference between two successive numbers can't be zero."
            Return
        temp_stack.pop()
    Return wiggle_stack
```

(-0.5 points): incorrect algorithm; for example, it does not work for 5 4 3 2 1

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.**

Input: an array of n positive integers A[ ]
Output: an arrangement of the elements in the given array that yields the largest odd value

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: A[ ] = { 1, 23, 34, 56 }
Output = 5634231

Input: A[ ] = { 54, 546, 548, 60 }
Output = 6054854654

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.

A corner case can be one with an empty array A[] or one where all the elements in the array have same integers (i.e A={ 55, 555, 5, 5555 })

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?

The use case for this can be a student test grading system where the grades are given to the students on the basis of the marks obtained in various courses and then assigning the grades accordingly.

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?

X

It is similar to radix sort. That problem tends to have an efficient algorithm as the run time will be O(d*n) where d is how many digits to be sorted.

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.

X

We will arrange the given array of numbers such that the output will be the largest odd values. If the numbers are single digit between 0 and 9 then we can sort the numbers in descending order. For 2 or more digits we compare by concatenating the number (m+n compare to n+m)

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

X

```
string answer= ‘ ‘;           //store the final output in string because it will be easy to concat
string strArr = new String[A.length];         // create a string array
for(int i=0;i<A.length;i++)
convert the given integer array A to string array
Sort the string array in descending order and compare the two elements
Check (strArr[i] + strArr[j] >= strArr[j] + strArr[i])
Swap the values if false
print the string answer
```

Efficiency

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

(-0.5 points): incorrect baseline algorithm; it needs to return the largest odd value, not the largest value

X

The efficiency is O(nlogn). Because we use sort and 1 for loop.

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?

Exercise 3: 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.**

Input: A array of n integers
Output: Numbers that is duplicate/repeated in the array

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).

A=[2,3,4,2,5,6,4]
Output: [2,4]

A=[6,9,0,4,0]
Output: [0]

A=[1,1,1,1,1]
Output: [1]

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.

The corner case will be an empty array. For instance, if A=[], then Output: None.
Let's say, the array is empty then, the output will not return anything.

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?

The use cases for this problem will be to remove the data redundancy from a problem.

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?

Similar problem for this is cyclic sorting algorithm as it is useful when dealing with numbers in a given range and asking to find the duplicates ones. Yes, the problem has an efficient algorithm. And, has the time complexity for $\Theta(n)$.

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.

Using Brute-Force method for this problem

Input: A array of n integers
Output: Numbers that is duplicate/repeated in the array

1. Start
2. Input n elements of array with repeated elements in it
3. for loop i=0 to length<n-1
4. Another loop running in the array for j=0 to n-1
5. Check whether is i and j indices are same or not
6. a[i]==a[j]If yes the, add the element to a new array
7. End

Correct baseline algorithm

Pseudocode

Now, write clear pseudocode for your baseline algorithm.

```
1. array<int>arr //to store the elements
2. array<int> out //to store the repeated elements
2. for i=0, i<n-1, i++ //running the loop on array
3. for j=0;j<n-1;j++ //running another loop on array
4. if a[i]==a[j] //comparing the elements at the i, j indicies
5. out[]=a[i]
6. break //to break the second loop if get the repeated element
7. return out //the output array with repeated elements
```

Efficiency

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

The time complexity of the algorithm will be Θ(n^2) as we are using two nested loops.

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?