Homework #1

Wonjun Park
UTA ID: 1002237177
wxp7177@mavs.uta.edu

1. You have an array containing integers from 1 to 10 (not in order) but one number is missing (there is 9 numbers in the array).

a) write a pseudo code to find the missing number (15 points).

```
$missing\_number(array)$:
    $n = 10$
    $total = n \times (n+1)/2$

    $sum = 0$
    for $i$ in array:
        $sum += i$

    $missing\_number = total - sum$
    return $missing\_number$
```

b) what is the worst case run time complexity of your suggested solution (5 points)

My suggestion has a time complexity of $O(n)$, where n is the number of elements in the `array`. The algorithm requires a single loop to calculate the sum of the elements in the `array`.

2. You are given an array of integers:

a) write a pseudo code to find all pairs of numbers whose sum is equal to a particular number (15 points).

```
$find\_pairs(array, target)$:
    $array.sort()$

    $left = 0$
    $right = len(array) - 1$
    $pairs = []$

    while $left < right$ :
        $sum = array[left] + array[right]$

        if $sum == target$ :
            $pairs.append((array[left], array[right]))$
            $left += 1$
            $right -= 1$
        elif $sum < target$ :
            $left += 1$
        else:
            $right -= 1$

    return $pairs$
```

b) what is the worst case run time complexity of your suggested solution (5 points)

Assume that the sorting algorithm utilized in the `array.sort()` function utilized $O(n \log n)$ time complexity algorithms such as merge sort. The suggested solution has a time complexity of $O(n \log n + n) = O(n \log n)$, where n is the number of elements in the `array`.

3. You are given an array of integers:

a) write a pseudo code to remove duplicates from your array (15 points).

```
$remove\_duplicates(array)$:
    $n = len(array)$
    $present = [False] * n$
    $unique = []$

    for $num$ in $array$ :
        if not $present[num]$ :
            $present[num] = True$
            $unique.append(num)$

    return $unique$
```

b) what is the worst case run time complexity of your suggested solution (5 points)

The algorithm has a time complexity of $O(n)$, where n is the number of elements in the `array`. The algorithm requires a single loop to iterate through the elements in the `array` and check whether the element has already presented in the `unique` list with the `present` hash table.

4. you are given 2 sorted arrays:

a) write a pseudo code to find the median of the two sorted arrays (combined in to 1 array) (15 points).

```
$find\_median(array1, array2)$:
    $array = array1 + array2$
    $array.sort()$

    $n = len(array)$
    if $n\%2 == 0$ :
        $median = (array[n//2 - 1] + array[n//2])/2$
    else:
        $median = array[n//2]$

    return $median$
```

b) what is the worst case run time complexity of your suggested solution (5 points)

Assume that the sorting algorithm utilized in the `array.sort()` function utilized $O(n \log n)$ time complexity algorithms such as merge sort. The suggested solution has a time complexity of $O(n \log n)$, where n is the number of elements in the `array` due to the sorting process.

5. Answer the following questions:

a) When does the worst case of Quick sort happen and what is the worst case run time complexity in terms of big O? (5 points)

The worst case of Quick sort occurs when the pivot element is the smallest or largest element in the array. The worst case run time complexity of Quick sort is $O(n^2)$, where n is the number of elements in the array.

b) When does the best case of bubble sort happen and what is the best case run time complexity in terms of big O? (5 points)

The best case of Bubble sort occurs when the array has already been sorted. The best case run time complexity of Bubble sort is $O(n)$, where n is the number of elements in the array.

c) What is the runtime complexity of Insertion sort in all 3 cases? Explain the situation which results in best, average and worst case complexity. (10 points)

1) The best case run time complexity of Insertion sort is $O(n)$, where n is the number of elements in the array. The best case occurs when the array is already sorted. **2)** The average case run time complexity of Insertion sort is $O(n^2)$, where n is the number of elements in the array. The average case occurs when the array is randomly shuffled. **3)** The worst case run time complexity of Insertion sort is $O(n^2)$ as well, where n is the number of elements in the array. The worst case occurs when the array is sorted in reverse order.