Assignment 5

Short Answers

Answer the following questions with complete sentences in your own words. You are encouraged to conduct your own research online or through other methods before answering the questions. If you research online, please consult multiple sources before you write down your answers. You are expected to be able to explain your answers in detail. If you are unable to understand or answer some of the questions asked or the contents you have found, please provide the sources such as the website link or book, and mark your question in red.

1.Think about all the questions you meet before and summarize algorithms and approaches you encountered.
Assignment 1 mainly involved using different ways of indexing to solve the problems, as well as using sets to find unique elements. There was also one simple swapping algorithm and some ascii math for rotation.
Assignment 2 involved utilizing dictionaries, sets, and stacks in algorithms.
Assignment 3 involved a binary search algorithm, as well as some swapping using sets. Some problems like Q5 utilized set lookup for faster computation. There was also one question for creating a class with its functionalities defined according to the rules of the guessing game.
Assignment 4 involved different methods of tree and linked list traversal, including several approaches to DFS and binary search implementations for BSTs.
Assignment 5 involves different sorting algorithms like bubble sort, insertion sort, mergesort and selection sort, of which mergesort is the most effective. It also involved more recursive algorithms and some dynamic programming, which can be more time effective compared to recursion at the cost of more space.

Coding Practice

Write algorithms in Python to Complete following questions. Please write your own answers. You are highly encouraged to present more than one way to answer the questions. Please follow the syntax, and you need to comment on each piece of code you write to clarify your purpose. Also you need to clearly explain your logic of the code for each question. Clearly state your assumptions if you have any. You may discuss with others about the questions, but please write your own code. Explain your algorithm clearly in detail.

For all questions, list out your time complexity and space complexity.

1. Implement Bubble Sort.

Time complexity O(n^2), space complexity O(1)

```python
def bubbleSort(arr):
    for i in range(len(arr)):
        for j in range(len(arr)-1):
        # swap if current is larger than the next element
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

2. Implement Selection Sort.
   Time complexity O(n^2), space complexity O(1)

```python
def selectionSort(arr):
    for i in range(len(arr)):
        #index of minimum value
        minIndex = i
        for j in range(i+1,len(arr)):
            #update index if new min is found
            if arr[minIndex] > arr[j]:
                minIndex = j
        #swap min to the end
        arr[minIndex], arr[i] = arr[i],arr[minIndex]
    return arr
```

3. Implement Insertion Sort.

Time complexity O(n^2), space complexity O(1)

```python
def insertionSort(arr):
    for i in range(1,len(arr)):
        j = i-1
        value = arr[i]
        #shift all elements greater than the current value up by 1
        while j >= 0 and value < arr[j]:
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = value
        print(arr)
    return arr
```

4. implement Merge Sort.

Time complexity O(nlogn), space complexity O(n)

```python
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        #divide the array up until it's one element and sort those
        mergeSort(left)
        mergeSort(right)
        #i and j for the two halves
        i = j = 0
        #k for merged list
        k = 0
        #append the smaller value and increment the iterator
        while i < len(left) and j < len(right):
            if left[i] <= right[j]:
                arr[k] = left[i]
                i += 1
            else:
                arr[k] = right[j]
                j += 1
            k += 1
        # any remaining values in either left or right
        while i < len(left) or j < len(right):
            if i < len(left):
                arr[k] = left[i]
                k += 1
                i += 1
            if j < len(right):
                arr[k] = right[j]
                k += 1
                j += 1
    return arr
```

5.Implement Longest Common Subsequence in two ways: Dynamic Programming and Recursion.

```python
# recursive
def longestSubsequence(s, t):
    if not s or not t:
        return 0
    #if the first character is the same, add 1 to the count and recurse over [1:]
    elif s[0] == t[0]:
        return 1 + longestSubsequence(s[1:],t[1:])
    #if they're different, then compare the second character of s/t with t/s
    else:
        return max(longestSubsequence(s[1:], t), longestSubsequence(s,t[1:]))
```
Time complexity: O(2^n). Space complexity O(1)

```
# DP
def longestSubsequence1(s,t):
    m = len(s)
    n = len(t)
    #create a matrix size m+1 by n+1
    matrix = [[0 for i in range(n+1)] for j in range(m+1)]
    for row in range(1,m+1):
        for col in range(1,n+1):
            if s[row - 1] == t[col - 1]:
                matrix[row][col] = 1 + matrix[row - 1][col - 1]
            else:
                matrix[row][col] = max(matrix[row][col-1],matrix[row-1][col])
    return matrix[m][n]
```

Time complexity: O(n^2). Space complexity: O(n^2) because matrix is m x n

6. Implement find all combinations of a string.

```
def findCombos(s,index=0):
    if index == len(s) - 1:
        print("".join(s))
    for i in range(index, len(s)):
        # change string to character array
        letters = [ch for ch in s]
        # swap elements of the word
        letters[index], letters[i] = letters[i], letters[index]
        # recurse until index is the same as length
        findCombos(letters, index + 1)
```
Time complexity O(n!) because there are n! combinations of a string of length n

Space complexity: O(n!) to store all the combinations

7.You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top/

1. 1 step + 1 step
2. 2 steps

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step
2. 1 step + 2 steps
3. 2 steps + 1 step

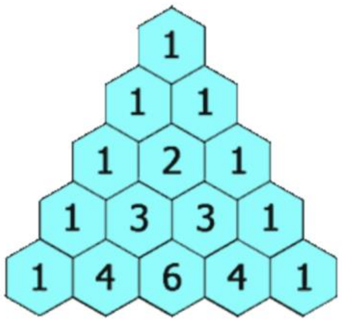
```
# the way to get to n steps is the way to get to n-1 steps + 1 step and n-2 steps
+ 2 steps
# f(n) = f(n-1) + f(n-2)
def stairs(n):
    #base cases
    if n == 0:
        return 0
    if n == 1:
        return 1
    #base array for 1 and 2 steps
    steps = [1,2]
    #DP for more than 2 steps maintaining constant space
    for i in range(2,n):
        temp = steps[1]
        steps[1] = steps[0] + steps[1]
        steps[0] = temp
    return steps[1]
```

Time complexity : O(n)

Space complexity: O(1)

8. Given an integer numRows, return the first numRows of Pascal’s triangle.

In Pascal’s triangle, each number is the sum if the two numbers directly above it as shown:



Example 1:

Input: numRows = 5

Output: [[1], [1,1], [1,2,1], [1,3,3,1], [1,4,6,4,1]]

Example 2:

Input: numRows = 1

Output: [[1]]

```
def pascal(n):
    #base case
    output = [[1]]
    for i in range(1,n):
        #fill triangle with 1s
        output.append([1]*(i+1))
        for j in range(1,i):
            # set each value as the sum of the numbers above it
            output[i][j] = output[i-1][j] + output[i-1][j-1]
    return output
```

Time complexity: O(n^2)
Space complexity O(n)

9. You are given an integer array nums. In one move, you can pick an index i where 0 <= i <= nums.length and increment nums[i] by 1.

Return the minimum number of moves to make every value in nums unique.

Example 1:

Input: nums = [1,2,2]

Output: 1

Explanation: After 1 move, the array could be [1,2,3]

Example 2:

Input: nums = [3,2,1,2,1,7]

Explanation: After 6 moves, the array could be [3,4,1,2,5,7].

```
def minMovesUnique(nums):
    output = 0
    value = -1
    #value represents the unique value at each position
    nums_sorted = sorted(nums)
    for i in nums_sorted:
    #unique value
        if value < i:
            value = i
    #if value is >= i then it's not unique, so increment and add to the output
        else:
            value += 1
            output += value - i
    return output
```
Time complexity O(nlogn) because of sorting

Space complexity: O(n) for storing the sorted array