Python for Computer Science and Data Science 2 (CSE 3652)

MINOR ASSIGNMENT-2: COMPUTER SCIENCE THINKING: RECURSION, SEARCHING, SORTING AND BIG O

1. Write a recursive function power(base, exponent) that, when called, returns *base*^{exponent} Example: power(3,4) = 3*3*3*3. Both base and exponent are user-defined and assume that the exponent is an integer greater than or equal to 1.

Ans:-

```python
def power(base, exponent):
    if exponent == 1:
        return base
    return base * power(base, exponent - 1)

base = int(input("Enter base: "))
exponent = int(input("Enter exponent: "))
if exponent < 1:
    print("Exponent must be greater than or equal to 1.")
else:
    print(f"{base}^{exponent} =", power(base, exponent))
```

Output:-

```
Enter base: 3
Enter exponent: 4
3^4 = 81
```

2. The greatest common divisor of integers x and y is the largest integer that evenly divides into both x and y. Write and test a recursive function gcd that returns the greatest common divisor of x and y. The gcd of x and y is defined recursively as follows: If y is equal to 0, then *gcd(x, y)* is x; otherwise, *gcd(x, y)* is *gcd(y, x%y)*?

Ans:-

```python
def gcd(x, y):
    if y == 0:
        return x
    return gcd(y, x % y)

x = int(input("Enter first number: "))
y = int(input("Enter second number: "))
print(f"GCD of {x} and {y} is", gcd(x, y))
```

Output:-

```
Enter first number: 36
Enter second number: 15
GCD of 36 and 15 is 3
```

3. Write a recursive function that takes a number n as an input parameter and prints n-digit strictly increasing numbers.

Ans:-

```python
def generate_numbers(n, start=1, num=""):
    if n == 0:
        print(num)
        return
    for i in range(start, 10):
        generate_numbers(n - 1, i + 1, num + str(i))

n = int(input("Enter the number of digits: "))
generate_numbers(n)
```

Output:-

```
Enter the number of digits: 8
12345678
12345679
12345689
12345789
12346789
12356789
12456789
13456789
23456789
```

4. Implement a recursive solution for computing the nth Fibonacci number. Then, analyze its time complexity. Propose a more efficient solution and compare the two approaches.

Ans:- Recursive Solution -

```
def fibonacci_recursive(n):
    if n <= 1:
        return n
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)

n = int(input("Enter n: "))
print(f"Fibonacci({n}) =", fibonacci_recursive(n))
```

Output:-

```
Enter n: 10
Fibonacci(10) = 55
```

Time Complexity Analysis:

a) Each call to fibonacci_recursive(n) makes two recursive calls.

b) This results in an exponential time complexity: O(2^n).

Iterative Approach -

```
def fibonacci_iterative(n):
    if n <= 1:
        return n
    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    return b

n = int(input("Enter n: "))
print(f"Fibonacci({n}) =", fibonacci_iterative(n))
```

Output:-

```
Enter n: 15
Fibonacci(15) = 610
```

Time Complexity Analysis: O(n), but without the overhead of recursion.

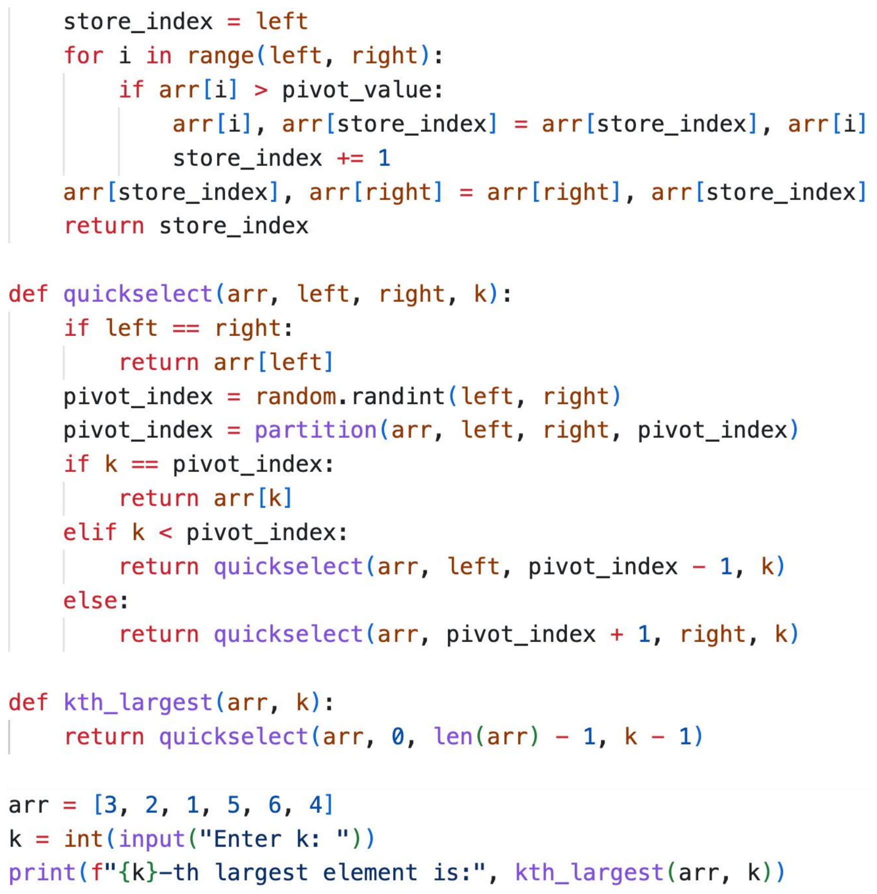
Comparison:

Approach	**Time Complexity**	**Space Complexity**	**Notes**
Recursive	O(2^n)	O(n) (due to call stack)	Very slow for large n
Iterative	O(n)	O(1)	Most efficient in both time & space

5. Given an array of N elements, not necessarily in ascending order, devised an algorithm to find the kth largest one. It should run in O(N) time on random inputs.

Ans:-

```
import random
def partition(arr, left, right, pivot_index):
    pivot_value = arr[pivot_index]
    arr[pivot_index], arr[right] = arr[right], arr[pivot_index]
```



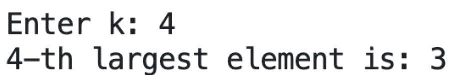
```
    store_index = left
    for i in range(left, right):
        if arr[i] > pivot_value:
            arr[i], arr[store_index] = arr[store_index], arr[i]
            store_index += 1
    arr[store_index], arr[right] = arr[right], arr[store_index]
    return store_index

def quickselect(arr, left, right, k):
    if left == right:
        return arr[left]
    pivot_index = random.randint(left, right)
    pivot_index = partition(arr, left, right, pivot_index)
    if k == pivot_index:
        return arr[k]
    elif k < pivot_index:
        return quickselect(arr, left, pivot_index - 1, k)
    else:
        return quickselect(arr, pivot_index + 1, right, k)

def kth_largest(arr, k):
    return quickselect(arr, 0, len(arr) - 1, k - 1)

arr = [3, 2, 1, 5, 6, 4]
k = int(input("Enter k: "))
print(f"{k}-th largest element is:", kth_largest(arr, k))
```

Output:-



```
Enter k: 4
4-th largest element is: 3
```

6. For each of the following code snippets, determine the time complexity in terms of Big O. Explain your answer.

a)

```
def example1(n):
    for i in range(n):
        for j in range(n):
            print(i, j)
```

Ans: - Time Complexity: O(n^2)

i. The outer loop runs n times.
ii. The inner loop runs n times for each iteration of the outer loop.
iii. Total iterations: $n \times n = n^2$.
iv. Big O Complexity: O(n^2) (Quadratic time).

b)

```
for i in range(n):
    print(i)
```

Ans: - Time Complexity: O(n)

i. The loop runs n times.
ii. Each iteration performs a constant-time operation (O(1)).
iii. Big O Complexity: O(n) (Linear time).

c)

```
def recursive_function(n):
    if n <= 1:
        return 1
            return recursive_function(n - 1) + recursive_function(n - 1)
```

Ans: - Time Complexity: O(2^n)

i. This is a tree recursion, where each call spawns two recursive calls.
ii. The number of recursive calls forms a binary tree of depth n.
iii. The total number of calls is approximately 2^n in the worst case.
iv. Big O Complexity: O(2^n) (Exponential time).

7. Write Given *N* points on a circle, centered at the origin, design an algorithm that determines whether there are two points that are antipodal, i.e., the line connecting the two points goes through the origin. Your algorithm should run in time proportional to *NlogN*.

Ans:-

```
import math
def are_antipodal(points):
    angles = sorted(math.atan2(y, x) for x, y in points)
    n = len(angles)
    for i in range(n):
        target = angles[i] + math.pi
        lo, hi = 0, n - 1

        while lo <= hi:
            mid = (lo + hi) // 2
            if abs(angles[mid] - target) < 1e-9:
                return True
            elif angles[mid] < target:
                lo = mid + 1
            else:
                hi = mid - 1
    return False

points = [(1, 0), (-1, 0), (0, 1), (0, -1)]
print(are_antipodal(points))
```

Output:-
```
True
```

8. The quicksort algorithm is a recursive sorting technique that follows these steps:

a) Partition Step: Choose the first element of the array as the pivot and determine its final position in the sorted array by ensuring all elements to its left are smaller and all elements to its right are larger.

b) Recursive Step: Recursively repeat the partitioning process on the subarrays created on either side of the pivot.

As an example, consider the array [37, 2, 6, 4, 89, 8, 10, 12, 68, 45] with 37 as the pivot. Using the partitioning logic, the pivot eventually moves to its correct position, resulting in two subarrays: [12, 2, 6, 4, 10, 8] and [89, 68, 45]. The algorithm continues recursively until the entire array is sorted.

Write a Python function quick sort that implements the quicksort algorithm. The function should include a helper function quick sort helper to handle recursion. The helper function must take a starting and ending index as arguments and sort the array in-place. Demonstrate the function by sorting the given array and printing the sorted output.

Ans:-

```
def partition(arr, low, high):
    pivot = arr[low]
    left = low + 1
    right = high
    while True:
        while left <= right and arr[left] <= pivot:
            left += 1
        while left <= right and arr[right] > pivot:
            right -= 1
        if left <= right:
            arr[left], arr[right] = arr[right], arr[left]
        else:
            break
    arr[low], arr[right] = arr[right], arr[low]
    return right

def quick_sort_helper(arr, low, high):
    if low < high:
        pivot_index = partition(arr, low, high)
        quick_sort_helper(arr, low, pivot_index - 1)
        quick_sort_helper(arr, pivot_index + 1, high)

def quick_sort(arr):
    quick_sort_helper(arr, 0, len(arr) - 1)

arr = [37, 2, 6, 4, 89, 8, 10, 12, 68, 45]
quick_sort(arr)
print("Sorted array:", arr)
```

Output:-
```
Sorted array: [2, 4, 6, 8, 10, 12, 37, 45, 68, 89]
```

9. You are given the following list of famous personalities with their net worth:
   - Elon Musk: $433.9 Billion
   - Jeff Bezos: $239.4 Billion
   - Mark Zuckerberg: $211.8 Billion
   - Larry Ellison: $204.6 Billion
   - Bernard Arnault & Family: $181.3 Billion
   - Larry Page: $161.4 Billion

Develop a program to sort the aforementioned details on the basis of net worth using

a. Selection sort
b. Bubble sort
c. Insertion sort.

The final sorted data should be the same for all cases. After you obtain the sorted data, present the result in the form of the following dictionary: {'name1':networth1, 'name2':networth2,...}.

Ans:-

```
people = [
    ("Elon Musk", 433.9),
    ("Jeff Bezos", 239.4),
    ("Mark Zuckerberg", 211.8),
    ("Larry Ellison", 204.6),
    ("Bernard Arnault & Family", 181.3),
    ("Larry Page", 161.4),
]

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j][1] < arr[min_idx][1]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(n - 1 - i):
            if arr[j][1] > arr[j + 1][1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j][1] > key[1]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

people_selection = people.copy()
people_bubble = people.copy()
people_insertion = people.copy()
selection_sort(people_selection)
bubble_sort(people_bubble)
insertion_sort(people_insertion)
sorted_dict = {name: networth for name, networth in people_selection}
print("Sorted Net Worths:", sorted_dict)
```

Output:-

```
Sorted Net Worths: {'Larry Page': 161.4, 'Bernard Arnault & Family': 181.3, 'Larry Ellison': 204.6, 'Mark Zuc
kerberg': 211.8, 'Jeff Bezos': 239.4, 'Elon Musk': 433.9}
```

10. Use Merge Sort to sort a list of strings alphabetically. Example:
Input: [’apple’, ’orange’, ’banana’, ’grape’]
Output: [’apple’, ’banana’, ’grape’, ’orange’]
Ans:-

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])
    return merge(left_half, right_half)

def merge(left, right):
    sorted_list = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            sorted_list.append(left[i])
            i += 1
        else:
            sorted_list.append(right[j])
            j += 1
    sorted_list.extend(left[i:])
    sorted_list.extend(right[j:])
    return sorted_list

words = ['apple', 'orange', 'banana', 'grape']
sorted_words = merge_sort(words)
print(sorted_words)
```

Output:-

```
['apple', 'banana', 'grape', 'orange']
```

11. What Without using the built-in sorted() function, write a Python program to merge two pre-sorted lists into a single sorted list using the logic of Merge Sort. Example:
Input: [1, 3, 5, 7] and [2, 4, 6, 8]
Output: [1, 2, 3, 4, 5, 6, 7, 8].
Ans:-

```
def merge_sorted_lists(list1, list2):
    merged_list = []
    i = j = 0
    while i < len(list1) and j < len(list2):
        if list1[i] < list2[j]:
            merged_list.append(list1[i])
            i += 1
        else:
            merged_list.append(list2[j])
            j += 1
    merged_list.extend(list1[i:])
    merged_list.extend(list2[j:])
    return merged_list

list1 = [1, 3, 5, 7]
list2 = [2, 4, 6, 8]
sorted_merged_list = merge_sorted_lists(list1, list2)
print(sorted_merged_list)
```

Output: -

```
[1, 2, 3, 4, 5, 6, 7, 8]
```