CMPSC 413 - Lab1

Algorithm Analysis

Exercise 1

Code and analyze (time complexity – algorithm analysis) to compute the greatest common divisor (GCD) of two numbers.

The time complexity of Euclid's GCD algorithm that I implemented is O(log(min(int1, int2))), where int1 and int2 are the two integers for which you are trying to find the greatest common divisor. This is because the algorithm repeatedly divides the larger number by the smaller number until the remainder is zero, and the number of times this division must be performed is logarithmic in the smaller of the two numbers.

```
import time

gcd = lambda int1, int2: int1 if int2 == 0 else gcd(int2, int1 % int2)

int1 : int
int2 : int

int1 = int(input('Enter first integer: '))
int2 = int(input('Enter second integer: '))

start_time = time.process_time()
greatest_common_divisor = gcd(int1, int2)
gcd_time = time.process_time() - start_time

print(greatest_common_divisor)
print(f'Time taken: {gcd_time}')
```

Output

```
Enter first integer: 42
Enter second integer: 12
6
Time taken: 3.500000000000031e-05
```

Exercise 2

Code and analyze (time complexity – algorithm analysis) to find maximum and minimum element from a list (or an array). Test it with 1000 random elements and 10,000 random elements. Also, calculate the actual CPU time it took find the minimum and maximum elements.

Time Complexity for Finding Minumum

The time complexity for finding the minimum element in an array using this algorithm is O(n), where n is the number of elements in the array. This is because you have to traverse the entire array once to identify the minimum element.

Time Complexity for Finding Maximum

Similar to finding the minimum element, the time complexity for finding the maximum element using this algorithm is also O(n), where n is the number of elements in the array. The algorithm iterates through the entire array once to determine the maximum element.

CPU Time to Find Minimum and maximum

The actual CPU time can vary based on factors like the processing power of your system and other running processes. The **time.process_time()** function is used in the code to measure the CPU time taken to execute the respective operations. In my execution of the code, 1000 elements took significantly less time than 10000 elements

```
import time
import random

def find_max_min(arr):
    max_element = float('-inf')
    min_element = float('inf')

    for num in arr:
        if num > max_element:
            max_element = num
        if num < min_element:
            min_element = num

    return max_element, min_element

def main():
    # Generate random arrays
    arr_1000 = [random.randint(1, 1000) for _ in range(1000)]
    arr_10000 = [random.randint(1, 10000) for _ in range(10000)]

    # Measure time for 1000 elements
    start_time = time.process_time()
    max_1000, min_1000 = find_max_min(arr_1000)
    print(f'For 1000 elements \n Max: {max_1000} \n Min: {min_1000}')
    time_1000 = time.process_time() - start_time
    print(f'Time taken for 1000 elemets: {time_1000}')

    # Measure time for 10000 elements
    start_time = time.process_time()
    max_10000, min_10000 = find_max_min(arr_10000)
    print(f'For 10000 elements \n Max: {max_10000} \n Min: {min_10000}')
    time_10000 = time.process_time() - start_time
    print(f'Time taken for 10000 elements: {time_10000}')
```

OUTPUT

```
For 1000 elements
 Max: 996
 Min: 1
Time taken for 1000 elemets: 5.2999999999997494e-05
For 10000 elements
```

```
 Max: 9999
 Min: 1
Time taken for 10000 elements: 0.00018699999999999967
```

Comparing the two values, the time taken to find the maximum and minimum elements increases as the number of elements in the array increases. In this case, the algorithm took longer to process the larger array (10000 elements) compared to the smaller array (1000 elements), which aligns with the expected behavior since the time complexity is linear (O(n)) and directly proportional to the number of elements in the array.

Exercise 3

Write a program to find maximum and minimum element in a list (or an array) using Divide and Conquer strategy and analyze the time complexity. Also, calculate the actual CPU time it took find the minimum and maximum elements.

Steps to perform Divide and Conquer

1. To use divide and conquer as an algorithm design technique, you must divide the problem into two smaller sub problems, solve each of them recursively, and then merge the two partial solutions into one solution to the full problem.
2. Whenever the merging takes less time than solving the two sub problems, we get an efficient algorithm. Divide-and conquer is a general algorithm design paradigm:
3. Divide: divide the input data S in two or more disjoint subsets S1, S2, ...
4. Recurrence: solve the sub problems recursively
5. Conquer: combine the solutions for S1, S2, ..., into a solution for S.

```python
import time
import random

def find_min_max(arr, low, high):
    if low == high:
        return arr[low], arr[high]

    if high - low == 1:
        if arr[low] < arr[high]:
            return arr[low], arr[high]
        else:
            return arr[high], arr[low]

    mid = (low + high) // 2

    min1, max1 = find_min_max(arr, low, mid)
    min2, max2 = find_min_max(arr, mid+1, high)

    return min(min1, min2), max(max1, max2)

if __name__ == '__main__':
    n = 1000
    arr = [random.randint(1, 1000) for i in range(n)]

    start = time.time()
    min_element, max_element = find_min_max(arr, 0, n-1)
```

```
    end = time.time()
    print("Minimum element:", min_element)
    print("Maximum element:", max_element)
    print("Time to find min and max in", n, "elements:", end - start)

    n = 10000
    arr = [random.randint(1, 1000) for i in range(n)]

    start = time.time()
    min_element, max_element = find_min_max(arr, 0, n-1)
    end = time.time()
    print("Minimum element:", min_element)
    print("Maximum element:", max_element)
    print("Time to find min and max in", n, "elements:", end - start)
```

OUTPUT

```
Minimum element: 1
Maximum element: 999
Time to find min and max in 1000 elements: 0.00022029876708984375
Minimum element: 1
Maximum element: 1000
Time to find min and max in 10000 elements: 0.0024640560150146484
```

The time complexity to find minimum element is O(n) where n is the number of elements in the list. This is because we need to look at each element once to find the minimum.

The time complexity to find maximum element is also O(n) for the same reason. We need to look at each element once.

The actual CPU time to find min and max grows linearly with the input size showing the linear time complexity.

In summary, the divide and conquer algorithm divides the problem into smaller subproblems, solves them recursively and combines the solutions to get the result. This leads to an overall linear time complexity for finding min and max.