James Zafiri

04/30/23

CSC6013

**Module 07: Programming Assignment 07**

**Problem 1.a):**

Write a recursive algorithm to determine the mean of the numbers in a non-empty array using the strategy that is illustrated by the following two examples with an array of 5 numbers (notice the recursive call with one less entry in the array) and an “array” of 1 number (think base case of the recursive algorithm):

mean({40,50,60,70,80}) = 4/5 \* mean({40, 50, 60, 70}) + 1/5 \* 80

mean({40}) = 40

Your function should have two parameters – the array of numbers and an integer indicating how many of the array values should be included in the calculations (note this is not the subscript of the last element of the array) – and should return the calculated value of the mean using a return statement.

For example, given five numbers stored in array A (in slots A[0]..A[4]), the function call

Mean(A, 5)

would cause a return of

4/5\*Mean(A, 4) + 1/5\*A[4].

Of course, the parameter in your code for the definition of the function would be n; the 4s and 5s would not show up explicitly in the code.

Problem 1.a) Code

def mean(A, n):

# Input: An array of numbers and an integer that specifies how many numbers to include in calculation

# Output: The calculated mean of numbers in an array that was given

# base case; if n is 1 then it returns the one number that's in the array

if n == 1:

return A[0]

# calculation based off of example using recursion to return the mean

else:

return (((n-1) / n) \* mean(A, n-1)) + ((1/n) \* A[n-1])

A1 = [12, 23, 37, 45, 63, 82, 47, 75, 91, 88, 102]

A2 = [-1.7, 6.5, 8.2, 0.0, 4.7, 6.3, 9.5, 12.2, 37.9, 53.2]

print(mean(A1, 11))

print(mean(A2, 10))

Problem 1.b)

Run your code on the problem instances:

i) A1 = <12, 23, 37, 45, 63, 82, 47, 75, 91, 88, 102> 11 entries

ii) A2 = <-1.7, 6.5, 8.2, 0.0, 4.7, 6.3, 9.5, 12.2, 37.9, 53.2> 10 entries

Problem 1.b) Output

60.45454545454545

13.68

**Problem 2.a):**

Write a recursive algorithm to determine the location in a sorted array where a specified searchkey is found. Unlike the algorithm in the class notes, this algorithm works with an array that is sorted into DESCENDING order and the code should print out all the subscripts in the array that were examined during the search. The sequence of elements that are examined is known as the “probe sequence”. Do not print out the “active” portion of the array, just the subscript of the one “middle” element that is compared to the searchkey.

Your function should have the same four input parameters as the code in the class notes – array, left subscript, right subscript, searchkey – and should return the subscript where the searchkey is found (or the value None) using a return statement.

For example, if you were searching for 27 in the array A = [50, 41, 27, 20, 17, 12, 5], the probe sequence printed by the algorithm would be: 3, 1, 2 because you would look at A[3]=20, then A[1]=41, then A[2]=27, and then you would stop because the searchkey was found.

Problem 2.a) Code

# importing math module to use floor function

import math

def BinarySearch(A, start, end, k):

# Input: An array in descending order, start & end positions, and key "k" to look for

# Output: Index i such that A[i] = k or None if a match was not found

# middle of the array

m = math.floor((end + start) / 2)

# return None if item is not found

if start > end:

return None

# to print out the "middle" subscript of array

print(m)

# if middle is what we are looking for, return it

if A[m] == k:

return m

# if middle is less than it, search again from beginning to one less index of that middle (left)

elif A[m] < k:

return BinarySearch(A, start, m-1, k)

# search to the right, one more index of that middle to end of array

else:

return BinarySearch(A, m+1, end, k)

A = [100, 87, 85, 80, 72, 67, 55, 50, 48, 42, 40, 31, 25, 22, 18]

BinarySearch(A, 0, 14, 87)

print("")

BinarySearch(A, 0, 14, 48)

print("")

BinarySearch(A, 0, 14, 33)

print("")

BinarySearch(A, 0, 14, 10)

Problem 2.b)

Run your code on the problem instances:

In array [100, 87, 85, 80, 72, 67, 55, 50, 48, 42, 40, 31, 25, 22, 18]

search for 87, search for 48, search for 33, search for 10

Problem 2.b) Output

7

3

1

7

11

9

8

7

11

9

10

7

11

13

14

**Problem 3.a):**

In his famous math book Elements, Euclid included an algorithm (that he might or might not have invented) to find the GCD of two positive integers. The basic idea is as follows:

GCD(a, b) = {a if b = 0 , GCD(b, a mod b) otherwise

For example, GCD (144, 42) = GCD (42, 18) = GCD (18, 6) = GCD (6, 0) = 6

Code this algorithm in Python. Use the appropriate Python operator to implement the modular arithmetic operation.

Your function should have two parameters – the two integer whose GCD is being determined – and should return the calculated value of the GCD using a return statement.

Before each recursive call, have your algorithm print out the two integers that were passed in as parameters. Have the driver/main program print out the result of the initial function call (that is, the final result).

Problem 3.a) Code

def gcd(a, b):

# Input: Two integers whose GCD is being determined

# Output: Calculated value of the GCD of the two numbers that were passed in

# the if and else statements below follow Euclid's algorithm that was given

if b == 0:

return a

else:

# to show the two integers being passed in before recursion

print(f"Integers being passed in are: {a} and {b}")

return gcd(b, a % b)

a = 2468

b = 1357

print(f"The GCD of {a} and {b} is:", gcd(a, b), "\n")

c = 111

d = 378

print(f"The GCD of {c} and {d} is:", gcd(c, d), "\n")

e = 123456789

f = 987654321

print(f"The GCD of {e} and {f} is:", gcd(e, f))

Problem 3.b)

Run your code on the problem instances:

i) GCD (2468, 1357)

ii)  GCD (111, 378)

iii)  GCD (123456789, 987654321)

Problem 3.b) Output

Integers being passed in are: 2468 and 1357

Integers being passed in are: 1357 and 1111

Integers being passed in are: 1111 and 246

Integers being passed in are: 246 and 127

Integers being passed in are: 127 and 119

Integers being passed in are: 119 and 8

Integers being passed in are: 8 and 7

Integers being passed in are: 7 and 1

The GCD of 2468 and 1357 is: 1

Integers being passed in are: 111 and 378

Integers being passed in are: 378 and 111

Integers being passed in are: 111 and 45

Integers being passed in are: 45 and 21

Integers being passed in are: 21 and 3

The GCD of 111 and 378 is: 3

Integers being passed in are: 123456789 and 987654321

Integers being passed in are: 987654321 and 123456789

Integers being passed in are: 123456789 and 9

The GCD of 123456789 and 987654321 is: 9