UNIVERSITY IBN TOFAIL

Algorithms II

Problem Set II

Exercise 1:

Write an algorithm that calculates the combination C_N^P. Use a function "Fact" that calculates and returns the factorial of an integer.
Translate this algorithm into Python.

Correction

Algorithm

Require: Two integers n and p $(0 \leq p \leq n)$
Ensure: Combination $C(n, p)$
  Define function Fact(x):
  $result \leftarrow 1$
  for $i \leftarrow 1$ to x **do**
    $result \leftarrow result \times i$
  end for
  return $result$
  Compute $numerator \leftarrow \text{Fact}(n)$
  Compute $denominator \leftarrow \text{Fact}(p) \times \text{Fact}(n-p)$
  Output $C(n, p) \leftarrow \dfrac{numerator}{denominator}$

Python Implementation

```python
def fact(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

def cnp(n, p):
    if p < 0 or p > n:
        return 0
    return fact(n) // (fact(p) * fact(n - p))

# Example usage
n = int(input("Enter n: "))
p = int(input("Enter p: "))
print(f"C({n}, {p}) = {cnp(n, p)}")
```

Listing 1: Python Code for Exercise 1

Exercise 2:

Write an algorithm that determines the first 80 prime numbers ≥ 2. Use a function "prime" that determines whether its argument is prime or not.
Translate this algorithm into Python.

Correction

Algorithm

Require: None
Ensure: List of first 80 primes 2
Define function is_prime(x):
if $x < 2$ **then**
return False
end if
for $i \leftarrow 2$ to $\sqrt{x}$ **do**
if $x \mod i = 0$ **then**
return False
end if
end for
return True
Initialize $primes \leftarrow [\,]$
Initialize $num \leftarrow 2$
while len($primes$) < 80 **do**
if is_prime(num) **then**
Append num to $primes$
end if
$num \leftarrow num + 1$
end while
Output $primes$

Python Implementation

```python
def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(x**0.5) + 1):
        if x % i == 0:
            return False
    return True

primes = []
num = 2
while len(primes) < 80:
    if is_prime(num):
        primes.append(num)
    num += 1
print(primes)
```

Listing 2: Python Code for Exercise 2

Exercise 3:

Write an algorithm that reads a sequence of pairs (x, y) ending with $(0, 0)$ and displays each time the greatest common divisor of the two numbers. Use a function "GCD" that returns the greatest common divisor of its two parameters.
Translate this algorithm into Python.

Correction

Algorithm

Require: Pairs (x, y) ending with $(0, 0)$
Ensure: Display $\gcd(x, y)$ for each pair
  Define function $\gcd(a, b)$:
  while $b \neq 0$ **do**
    $a, b \leftarrow b, a \mod b$
  end while
  return a
  Read pairs (x, y):
  while $(x, y) \neq (0, 0)$ **do**
    Output $\gcd(x, y)$
    Read next (x, y)
  end while

Python Implementation

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

while True:
    x, y = map(int, input("Enter x, y: ").split())
    if x == 0 and y == 0:
        break
    print(f"GCD({x}, {y}) = {gcd(x, y)}")
```

Listing 3: Python Code for Exercise 3

Exercise 4:

Write a function "LCM" that determines the least common multiple of two strictly positive integers n and m.
Translate this algorithm into Python.

Correction

Algorithm

Require: Two positive integers n and m
Ensure: $\text{lcm}(n, m)$
Define function $\gcd(a, b)$:
while $b \neq 0$ **do**
$a, b \leftarrow b, a \mod b$
end while
return a
Compute $\text{lcm}(n, m) \leftarrow \dfrac{n \times m}{\gcd(n, m)}$
Output $\text{lcm}(n, m)$

Python Implementation

```
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def lcm(n, m):
    return abs(n * m) // gcd(n, m)

n, m = map(int, input("Enter n, m: ").split())
print(f"LCM({n}, {m}) = {lcm(n, m)}")
```

Listing 4: Python Code for Exercise 4

Exercise 5:

Write an algorithm that gives the list of all perfect numbers between 6 and 10000. Use a function "perfect" that determines if its argument is perfect or not. A number is said to be perfect if it is equal to the sum of all its strict divisors (For example, 28 is perfect: $28 = 1 + 2 + 4 + 7 + 14$).
Translate this algorithm into Python.

Correction

Algorithm

Require: None
Ensure: List of perfect numbers between 6 and 10,000
  Define function is_perfect(x):
  $sum\_divisors \leftarrow 0$
  for $i \leftarrow 1$ to $x - 1$ **do**
    if $x \mod i = 0$ **then**
      $sum\_divisors \leftarrow sum\_divisors + i$
    end if
  end for
  return $sum\_divisors = x$
  Initialize $perfects \leftarrow [\,]$
  for $x \leftarrow 6$ to 10000 **do**
    if is_perfect(x) **then**
      Append x to $perfects$
    end if
  end for
  Output $perfects$

Python Implementation

```python
def is_perfect(x):
    return sum(i for i in range(1, x) if x % i == 0) == x

perfect_numbers = [x for x in range(6, 10001) if is_perfect(x)]
print(perfect_numbers)
```

Listing 5: Python Code for Exercise 5

Exercise 6:

Consider a set $H_a = \{n \in \mathbb{N}, |, 2^n > a\}$ where $a \in \mathbb{N}$

1. Write the algorithm for the function MinSet that determines the minimum of the set H_a: Function MinSet(a: integer): integer

2. Use the result of the MinSet function to write the algorithm for the function DecimalBinary that converts an integer from decimal base to binary base (the result returned is stored in an array):

$$\text{Function DecimalBinary}(a: \text{ integer}): \text{ integer}[]$$

Correction

Algorithm

Require: Integer a
Ensure: Minimum n such that $2^n > a$, and binary representation of a

Define function MinSet(a):
$n \leftarrow 0$
while $2^n \leq a$ **do**
  $n \leftarrow n + 1$
end while
return n
Define function DecimalBinary(a):
$n \leftarrow \text{MinSet}(a)$
Initialize array $binary[0..n-1]$
for $i \leftarrow n-1$ down to 0 **do**
  $binary[i] \leftarrow a \mod 2$
  $a \leftarrow a//2$
end for
return $binary$

Python Implementation

```python
def min_set(a):
    n = 0
    while 2 ** n <= a:
        n += 1
    return n

def decimal_binary(a):
    n = min_set(a)
    binary = [0] * n
    for i in range(n - 1, -1, -1):
        binary[i] = a % 2
        a //= 2
    return binary

a = int(input("Enter a: "))
print(f"Binary of {a}: {decimal_binary(a)}")
```

Listing 6: Python Code for Exercise 6

Exercise 7:

Write an algorithm that asks the user to enter an integer N ($N \geq 2$), then calculates and displays all terms of the Fibonacci sequence less than or equal to N. The Fibonacci sequence is defined as follows:

$$\begin{cases} U_0 = 0 \\ U_1 = 1 \\ U_{n+2} = U_{n+1} + U_n \end{cases}$$

1. Use an iterative function "fibIter".
2. Use a recursive function "fibRec".

Translate this algorithm into Python.

Correction

Algorithm

Require: Integer $N \geq 2$
Ensure: Fibonacci sequence N
Define function fibIter(N):
$a \leftarrow 0, b \leftarrow 1$
Output a, b
while $b + a \leq N$ **do**
  $a, b \leftarrow b, a + b$
  Output b
end while
Define function fibRec(n):
if $n \leq 1$ **then**
  return n
end if
return fibRec$(n-1)$ + fibRec$(n-2)$

Python Implementation

```python
def fib_iter(n):
    a, b = 0, 1
    print(a, end=' ')
    while b <= n:
        print(b, end=' ')
        a, b = b, a + b

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

N = int(input("Enter N: "))
print("Iterative Fibonacci:")
fib_iter(N)
print("\nRecursive Fibonacci (first 10 terms):")
for i in range(10):
    print(fib_rec(i), end=' ')
```

Listing 7: Python Code for Exercise 7

Exercise 8:

Write a recursive function to calculate the sum of the digits of a positive integer n. Example: If $n = 834$, the sum of the digits of n is 15 $(= 8 + 3 + 4)$.

Correction

Algorithm

Require: Positive integer n
Ensure: Sum of digits of n

Define function digitSum(n):
if $n = 0$ **then**
return 0
end if
return $n \mod 10 + \text{digitSum}(n//10)$

Python Implementation

```python
def digit_sum(n):
    if n == 0:
        return 0
    return n % 10 + digit_sum(n // 10)

n = int(input("Enter n: "))
print(f"Sum of digits: {digit_sum(n)}")
```

Listing 8: Python Code for Exercise 8

Exercise 9:

Let n be a strictly positive integer. Write a recursive function digit(n, k) that returns the k-th digit of n from the right. Examples:

- The 3rd digit from the right of 5739 is 7.
- The 5th digit from the right of 81467 is 8.

Translate this algorithm into Python.

Correction

Algorithm

Require: Positive integers n and k
Ensure: k-th digit of n from the right
  Define function digit(n, k):
  if $k = 1$ **then**
    return $n \mod 10$
  end if
  return digit$(n//10, k-1)$

Python Implementation

```python
def digit(n, k):
    if k == 1:
        return n % 10
    return digit(n // 10, k - 1)

n = int(input("Enter n: "))
k = int(input("Enter k: "))
print(f"{k}-th digit from right: {digit(n, k)}")
```

Listing 9: Python Code for Exercise 9

Exercise 10:

Write a recursive function that calculates:

1. The sum of the first n natural numbers (starting from 1).
2. The sum of two natural numbers a and b.
3. The product of two natural numbers a and b.
4. The power of a raised to b (a being a real number, and b a natural number).
5. The quotient of two natural numbers a divided by b ($b \neq 0$).
6. The remainder of division of a by b (a and b are two natural numbers, and $b \neq 0$).

Translate this algorithm into Python.

Correction

Algorithm

Require: Natural numbers a, b
Ensure: Results of recursive operations
Define function sum(n):
if $n = 0$ **then**
return 0
end if
return $n + \text{sum}(n-1)$
Define function sum_ab(a, b):
if $b = 0$ **then**
return a
end if
return sum_ab($a+1, b-1$)
Define function product(a, b):
if $b = 0$ **then**
return 0
end if
return $a + \text{product}(a, b-1)$
Define function power(a, b):
if $b = 0$ **then**
return 1
end if
return $a * \text{power}(a, b-1)$
Define function quotient(a, b):
if $a < b$ **then**
return 0
end if
return $1 + \text{quotient}(a-b, b)$
Define function remainder(a, b):
if $a < b$ **then**
return a
end if
return remainder($a-b, b$)

Python Implementation

Correction

```python
def sum_n(n):
    return 0 if n == 0 else n + sum_n(n - 1)

def sum_ab(a, b):
    return a if b == 0 else sum_ab(a + 1, b - 1)

def product(a, b):
    return 0 if b == 0 else a + product(a, b - 1)

def power(a, b):
    return 1 if b == 0 else a * power(a, b - 1)

def quotient(a, b):
    return 0 if a < b else 1 + quotient(a - b, b)

def remainder(a, b):
    return a if a < b else remainder(a - b, b)

# Example usage
print("Sum of first 5 natural numbers:", sum_n(5))
print("Sum of 3 + 4:", sum_ab(3, 4))
print("Product of 3 * 4:", product(3, 4))
print("3^4:", power(3, 4))
print("Quotient of 10 / 3:", quotient(10, 3))
print("Remainder of 10 % 3:", remainder(10, 3))
```

Listing 10: Python Code for Exercise 10

Exercise 11:

Write the algorithm for the following function MajoritySquares:

$$\text{Function MajoritySquares}(T : \text{ integer}[1..N]): \text{ boolean}$$

The function returns TRUE if the number of perfect squares in array T is the majority. Translate this algorithm into Python.

Correction

Algorithm

Require: Array T of size N
Ensure: Boolean indicating if perfect squares are majority
Define function is_perfect(x):
$root \leftarrow \lfloor\sqrt{x}\rfloor$
return $root \times root = x$
Count perfect squares $count \leftarrow 0$
for $x \in T$ **do**
  if is_perfect(x) **then**
    $count \leftarrow count + 1$
  end if
end for
Return $count > N/2$

Python Implementation

```python
import math

def is_perfect(x):
    root = int(math.sqrt(x))
    return root * root == x

def majority_squares(T):
    count = sum(1 for x in T if is_perfect(x))
    return count > len(T) / 2

T = list(map(int, input("Enter array elements: ").split()))
print("Majority of perfect squares?", majority_squares(T))
```

Listing 11: Python Code for Exercise 11

Exercise 12:

Write an algorithm that asks the user to enter the size and elements of an integer array T. This algorithm uses two procedures "displayMax" and "displayMin" to determine and display the maximum and minimum of the array elements.
Translate this algorithm into Python.

Correction

Algorithm

Require: Array T of integers
Ensure: Display maximum and minimum

Read array size n
Read array elements $T[0..n-1]$
Initialize $max\_val \leftarrow T[0]$
Initialize $min\_val \leftarrow T[0]$
for $i \leftarrow 1$ to $n-1$ **do**
  if $T[i] > max\_val$ **then**
    $max\_val \leftarrow T[i]$
  end if
  if $T[i] < min\_val$ **then**
    $min\_val \leftarrow T[i]$
  end if
end for
Output max_val and min_val

Python Implementation

```python
def display_max_min(T):
    max_val = min_val = T[0]
    for num in T[1:]:
        if num > max_val:
            max_val = num
        if num < min_val:
            min_val = num
    print(f"Maximum: {max_val}, Minimum: {min_val}")

n = int(input("Enter array size: "))
T = list(map(int, input("Enter array elements: ").split()))
display_max_min(T)
```

Listing 12: Python Code for Exercise 12

Exercise 13:

Write an algorithm that asks the user to enter the size and elements of an integer array T, then asks the user to enter an integer a. The objective is to verify the existence of the number a in T.
Use a function "verifyExistence" with two arguments (T and a) that returns the index of the first occurrence of a in T and -1 if a does not exist in T.
Translate this algorithm into Python.

Correction

Algorithm

Require: Array T and integer a
Ensure: Index of first occurrence of a or -1
  Read array size n
  Read array elements $T[0..n-1]$
  Read target a
  Initialize $index \leftarrow -1$
  for $i \leftarrow 0$ to $n-1$ **do**
    if $T[i] = a$ **then**
      $index \leftarrow i$
      Break loop
    end if
  end for
  Output $index$

Python Implementation

```python
def verify_existence(T, a):
    for i, val in enumerate(T):
        if val == a:
            return i
    return -1

n = int(input("Enter array size: "))
T = list(map(int, input("Enter array elements: ").split()))
a = int(input("Enter value to search: "))
print(f"Index of {a}: {verify_existence(T, a)}")
```

Listing 13: Python Code for Exercise 13

Exercise 14:

Write a recursive procedure that rearranges the elements of an array in reverse order.

Procedure ReverseOrderRecurs(T: integer[1..N], i: integer)

Correction

Algorithm

Require: Array T and indices i (start), j (end)
Ensure: Reverse array in-place
Define function reverse_array(T, i, j):
if $i < j$ **then**
Swap $T[i]$ and $T[j]$
reverse_array$(T, i + 1, j - 1)$
end if

Python Implementation

```python
def reverse_array(T, i, j):
    if i < j:
        T[i], T[j] = T[j], T[i]
        reverse_array(T, i + 1, j - 1)

T = list(map(int, input("Enter array elements: ").split()))
reverse_array(T, 0, len(T) - 1)
print("Reversed array:", T)
```

Listing 14: Python Code for Exercise 14

Exercise 15:

Let T be an array of integers of size n ($n \leq 100$). Write recursive functions to perform the following operations:

1. Sum: which returns the sum of the elements of array T.
2. Product: which returns the product of the elements of array T.
3. Average: which returns the average of the elements of array T.
4. SearchElt: which returns the index of the element containing a given value.
5. SearchSeq: which returns the index of an element containing a given value in a vector T sorted in ascending order.
6. NumOcc: which returns the number of occurrences of a given value in T.
7. IsSorted: which indicates whether the array is sorted in ascending order or not.

Correction

Algorithm

Require: Array T of size n
Ensure: Recursive sum, product, average, search, etc.
  Define function sum_array(T, n):
  if $n == 0$ **then**
    return $T[0]$
  end if
  return $T[n] + \text{sum\_array}(T, n-1)$
  Define function product_array(T, n):
  if $n == 0$ **then**
    return $T[0]$
  end if
  return $T[n] * \text{product\_array}(T, n-1)$
  Define function search_elt(T, a, n):
  if $n == 0$ **then**
    return -1
  end if
  if $T[n] == a$ **then**
    return n
  end if
  return $\text{search\_elt}(T, a, n-1)$
  Define function is_sorted(T, n):
  if $n == 0$ **then**
    return True
  end if
  if $T[n] < T[n-1]$ **then**
    return False
  end if
  return $\text{is\_sorted}(T, n-1)$

Python Implementation

Correction

```python
def sum_array(T, n):
    return T[0] if n == 0 else T[n] + sum_array(T, n - 1)

def product_array(T, n):
    return T[0] if n == 0 else T[n] * product_array(T, n - 1)

def search_elt(T, a, n):
    if n < 0:
        return -1
    if T[n] == a:
        return n
    return search_elt(T, a, n - 1)

def is_sorted(T, n):
    if n == 0:
        return True
    if T[n] < T[n - 1]:
        return False
    return is_sorted(T, n - 1)

T = list(map(int, input("Enter array elements: ").split()))
n = len(T) - 1
print("Sum:", sum_array(T, n))
print("Product:", product_array(T, n))
print("Is sorted?", is_sorted(T, n))
print("Index of 5:", search_elt(T, 5, n))
```

Listing 15: Python Code for Exercise 15