[EEC3600-001] 수치해석		
소속: 전기전자공학부	학번: 12191529	이름: 장준영
Python Programming for Numerical Analysis		**HW number: #1**

1.1. (계산 복잡도) 아래의 두 알고리즘(함수)와 계산 복잡도(입력을 받아서 출력을 계산하는 데 필요한 사칙연산의 횟수)를 비교하시오.

a. Python 코드

```python
# Python Code 1
def sum_n_1(n):
    s = 0
    for i in range(1, n + 1):
        s = s + i
    return s
```

Python 코드 1

```python
# Python Code 2
def sum_n_2(n):
    return n*(n + 1) // 2
```

Python 코드 2

b. Python 출력

```python
print(sum_n_1(100))
print(sum_n_2(100))
```

Test Code

```
5050
5050
```

Result

c. 계산 복잡도 비교

A. Python 코드 1

: 반복문을 사용하여 1~n까지 더하는 방식이므로, 시간 복잡도는 $O(n)$.

B. Python 코드 2

: 공식을 이용한 방법이므로, 시간 복잡도는 $O(1)$. 즉, 두번째 방법이 더 효율적이다.

1.2. (최대값을 구하는 알고리즘) 리스트(list)를 입력으로 받아서 리스트의 components에서 최대값을 찾고, 그 인덱스(최대값의 위치)와 최대값을 출력으로 하는 파이썬 함수 findmax를 작성하시오.

a. Python 코드

```python
def findmax(lst):
    max_val = lst[0]                # list의 첫번째 값을 최대값으로 초기화
    max_idx = 0                     # 최대값의 index를 0으로 초기화
    for i in range(1, len(lst)):
        if lst[i] > max_val:        # 현재 값이 최대값보다 크면: 최대값 갱신 & 최대값의
            max_val = lst[i]
            max_idx = i
    return max_idx, max_val         # 최대값의 index, 최대값 반환

print(findmax([3, 1, 7, 5, 9, 2]))  # (4, 9)
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_2.py" "
(4, 9)
```

1.3. (Factorial) 연속한 숫자(자연수)의 곱을 구하는 알고리즘을 구현하는 파이썬 함수 fact를 작성하시오. 즉, fact(n) = $1 \times 2 \times 3 \times \ldots \times n!$.

a. Python 코드

```python
def fact(n):
    if n == 0 or n == 1:        # n = 0 또는 1 : 팩토리얼은 1 (종료 조건)
        return 1
    return n * fact(n - 1)      # 그렇지 않으면 : 재귀적으로 n * (n - 1)! 계산

print(fact(5))  # 120 = 5! = 5 * 4 * 3 * 2 * 1
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_3.py" "
120
```

1.4. (Greatest Common Divisor, GCD) 최대공약수를 구하는 알고리즘을 구현하는 파이썬 함수 gcd를 작성하시오. 즉, gcd(a,b) = 자연수 a와 b의 최대공약수.

a. Python 코드

```python
# Euclid Algorithm
# (a, b 의 최대공약수) = (b, a%b 의 최대공약수)

def gcd(a, b):
    # b가 0이 될 때까지 반복
    while b != 0:
        a, b = b, a % b  # a를 b로, b를 a를 b로 나눈 나머지로 갱신
    return a  # b가 0이 되면 a가 최대공약수

# 출력 해설
# gcd(48, 16) -> gcd(18, 48 % 18) -> gcd(18, 12)
# gcd(18, 12) -> gcd(12, 6)
# gcd(12, 6) -> gcd(6, 0)

print(gcd(48, 18))  # 6
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_4.py" "
6
```

1.5. (순차탐색) 주어진 리스트에 특정한 값이 있는지 찾아서 그 위치를 출력으로 돌려주는 알고리즘을 파이썬 함수로 구현하시오.

a. Python 코드

```python
# 리스트 lst에서 target 값을 찾는 순차 탐색 함수
def sequential_search(lst, target):
    for i, val in enumerate(lst):       # enumerate을 사용하여 인덱스(i), 값(val)을 동시에 반복
        if val == target:               # 값이 target과 같으면 인덱스 반환
            return i
    return -1                           # 끝까지 못 찾으면 -1 반환

# 출력 해설
# 리스트 [4, 2, 9, 7]에서 9(target)의 인덱스는 2
print(sequential_search([4, 2, 9, 7], 9))  # 2
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_5.py" "
2
```

1.6. (정렬) 주어진 리스트 안의 자료를 작은 수부터 큰 수 순서로 배열하여 정렬하는 알고리즘을 파이썬 프로그램으로 구현하시오.

a. Python 코드

```python
# 병합 정렬
# - 리스트를 반으로 계속 나눠서 길이가 1인 리스트로 만든 다음,
# - 두 리스트를 정렬된 상태로 합쳐나가면서 전체 정렬을 완성하는 알고리즘.

# 알고리즘 흐름 요약
# 1. 분할(Divide) : 리스트를 반으로 계속 나눈다.
# 2. 정복(Conquer) : 각 부분을 재귀적으로 정렬한다.
# 3. 병합(Merge) : 정렬된 두 부분 리스트를 합쳐서 정렬된 하나의 리스트로 만든다.

def merge_sort(lst):
    # 리스트의 길이가 1 이하 : 이미 정렬된 상태 -> 그대로 반환 (재귀의 종료 조건)
    if len(lst) <= 1:
        return lst

    mid = len(lst) // 2
    left = merge_sort(lst[:mid])
    right = merge_sort(lst[mid:])

    result = []
    i = j = 0

    # 두 리스트를 병합하면서 정렬
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    # 남은 원소들 추가
    result += left[i:]
    result += right[j:]

    return result

print(merge_sort([4, 2, 9, 7, 1]))  # [1, 2, 4, 7, 9]
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_6.py" "
[1, 2, 4, 7, 9]
```

1.7. (탐색) 자료가 (값의) 크기 순서대로 정렬된 리스트에서 특정한 값이 있는 위치를 돌려주고, 리스트에 그 값이 없으면 -1을 결과값으로 돌려주는 알고리즘을 파이썬 프로그램으로 구현하시오. 이때, binary search(이분 탐색)을 활용하시오. 즉, 리스트의 중간 위치를 찾고, 탐색의 대상이 되는 특정 값이 중간 위치의 값보다 큰 수인지 작은 수인지 판단하여, 후보 리스트 그룹을 1/2씩 줄여 나가면서 이 과정을 반복하시오.

a. Python 코드

```python
# 이진 탐색 함수 : 정렬된 리스트 lst에서 target의 인덱스를 찾는다.
def binary_search(lst, target):
    left, right = 0, len(lst) - 1   # 탐색 범위의 시작과 끝 설정.
    while left <= right:
        mid = (left + right) // 2   # 중간 인덱스 계산
        if lst[mid] == target:      # 중간 값이 목표값이면 : 인덱스 반환
            return mid
        elif lst[mid] < target:     # 중간 값보다 크면 : 오른쪽 반만 탐색
            left = mid + 1
        else:                       # 중간 값보다 작으면 : 왼쪽 반만 탐색
            right = mid - 1
    return -1                       # 찾지 못했을 경우 : -1 반환

sorted_list = [1, 3, 5, 7, 9]
print(binary_search(sorted_list, 5))  # 2
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_7.py" "
2
```

1.8. (문자열) 입력된 문자열이 '회문'이라면 true, 아니라면 false를 출력하라. 대소문자는 구분하지 않고, 입력된 문자열은 영문자와 숫자만으로 이루어져 있다.

a. Python 코드

```python
# 회문인지 확인하는 함수 (대소문자, 공백, 특수문자 무시)
def is_palindrome(s):
    # filter(함수, 반복가능한 객체) : 리스트, 문자열 등에서 "조건에 맞는 값만 걸려내는" 함수
    # 함수 : 각 요소에 적용될 조건 함수
    # 반복가능한 객체 : 리스트, 튜플, 문자열 등
    s = ''.join(filter(str.isalnum, s)).lower() # str.isalnum : 문자열이 '영문자, 숫자'로만 이루어져 있는지 확인해주는 문자열 메서드.

    # 문자열을 뒤집은 것과 원래 문자열이 같은지 비교
    return s == s[::-1]

print(is_palindrome("A man, a plan, a canal: Panama"))  # True
print(is_palindrome("race a car"))  # False
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_8.py" "
True
False
```

1.9. (리스트) 블록들의 높이를 입력받는다. 만약 비가 온다면, 블록들의 사이에 물이 얼마나 모일지 계산하는 프로그램을 구현하시오.

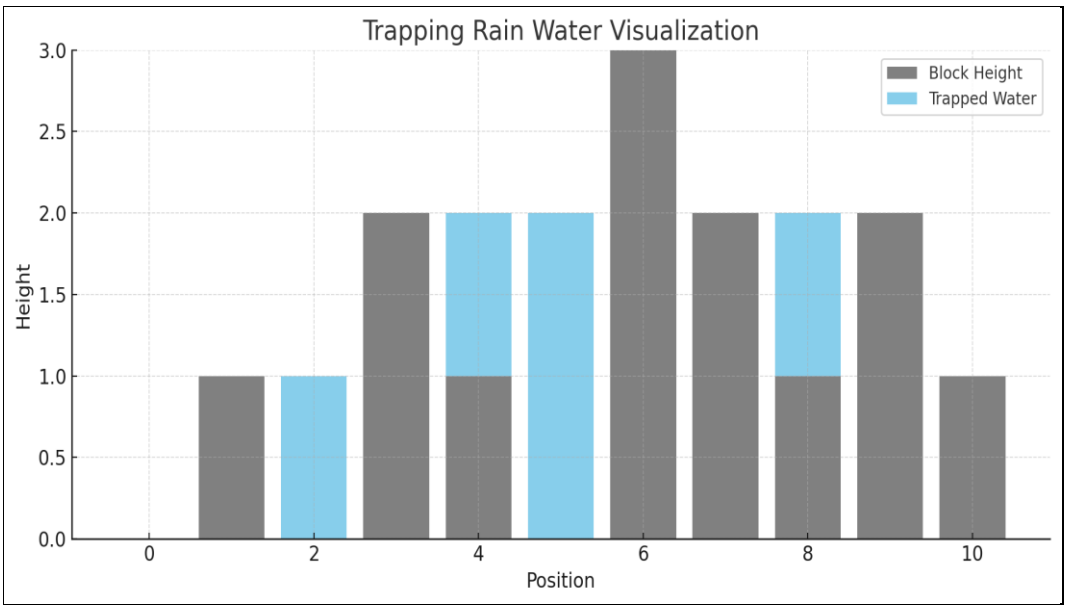
a. Python 코드

```python
# 빗물이 고일 수 있는 총량을 계산하는 함수
def trap(height):
    left, right = 0, len(height) - 1    # 양쪽 포인터 설정
    left_max = right_max = water = 0    # 왼/오 최대 높이와 물 저장량 초기화

    while left < right:
        # 더 낮은 쪽을 기준으로 물의 높이를 계산.
        if height[left] < height[right]:
            left_max = max(left_max, height[left])  # 왼쪽에서 가장 높은 벽 갱신
            water += left_max - height[left]        # 현재 위치에서 고일 수 있는 물 계산
            left += 1                               # 왼쪽 포인터 오른쪽으로 1칸 이동.
        else:
            right_max = max(right_max, height[right])   # 오른쪽에서 가장 높은 벽 갱신
            water += right_max - height[right]          # 현재 위치에서 고일 수 있는 물 계산
            right -= 1                                  # 오른쪽 포인터 왼쪽으로 1칸 이동.

    return water    # 총 고인 물의 양 반환

print(trap([0,1,0,2,1,0,3,2,1,2,1]))  # 5
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_9.py" "
5
```

c. 테스트 케이스 설명



1.10. (행렬 검색) 주어진 $m \times n$ 행렬에서 원하는 값을 찾아내는 함수를 만들어라. 행렬 안의 숫자들은 아래에서 위로, 오른쪽에서 왼쪽으로 내림차순으로 정렬되어 있다고 가정하고, 이러한 규칙성을 활용하여 검색 알고리즘의 효율성을 높이는 방법으로 프로그램(함수)을 구현하시오.

a. Python 코드

```python
# 정렬된 2차원 행렬에서 target 값을 찾는 함수
def search_matrix(matrix, target):
    # 빈 행렬일 경우 : False 반환
    if not matrix:
        return False

    # 시작 위치 : 맨 위 오른쪽
    row, col = 0, len(matrix[0]) - 1

    while row < len(matrix) and col >= 0:
        if matrix[row][col] == target:      # 값을 찾았을 때
            return True
        elif matrix[row][col] > target:     # 현재 값이 target보다 크면 : 왼쪽으로 이동
            col -= 1
        else:                               # 현재 값이 target보다 작으면 : 아래로 이동.
            row += 1

    return False

matrix = [
    [1, 4, 7, 11, 15],
    [2, 5, 8, 12, 19],
    [3, 6, 9, 16, 22],
    [10, 13, 14, 17, 24],
    [18, 21, 23, 26, 30]
]
print(search_matrix(matrix, 5))  # True : 4번만에 정답에 도달한다!
```

b. Python 출력

```
\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1\1_10.py" "
True
```

1.11. (numpy 모듈 사용 예제) 아래의 예제를 참고하여, 다음의 함수(function)를 설명하시오.

a. np.linalg.matrix_rank
   A. 입력: $A \in \mathbb{R}^{m \times n}$
   B. 출력: $rank(A) \in \mathbb{N}$
      - $rank(A) = np.linalg.matrix\_rank(A)$
      - 행렬 A의 랭크(rank), 즉 선형 독립인 행 또는 열의 최대 수.

b. np.trace
   A. 입력: $A \in \mathbb{R}^{n \times n}$ (정사각 행렬)
   B. 출력: $trace(A) \in \mathbb{R}$
      - $trace(A) = np.trace(A)$
      - 행렬 A의 주대각선 원소들의 합.

c. np.linalg.det
   A. 입력: $A \in \mathbb{R}^{n \times n}$ (정사각 행렬)
   B. 출력: $det(A) \in \mathbb{R}$
      - $det(A) = np.linalg.det(A)$
      - 행렬 A의 행렬식(determinant).

d. np.linalg.inv
   A. 입력: $A \in \mathbb{R}^{n \times n}$ (정사각 행렬, 가역행렬)
   B. 출력: $A_{inv} = \mathbb{R}^{n \times n}$
      - $A_{inv} = np.linalg.inv(A)$
      - 행렬 A의 역행렬.

e. np.linalg.matrix_power
   A. 입력: $A \in \mathbb{R}^{n \times n}, k \in \mathbb{Z}$
   B. 출력: $A_{pow} = \mathbb{R}^{n \times n}$
      - $A_{pow} = np.linalg.matrix\_power(A, k)$
      - 행렬 A를 정수 k 제곱한 결과.

f. Python 코드

```python
import numpy as np

A = np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])

print("Rank of A:", np.linalg.matrix_rank(A))
print("Trace of A:", np.trace(A))
print("Determinant of A:", np.linalg.det(A))
print("Inverse of A:\n", np.linalg.inv(A))
print("Matrix A raised to power 3:\n", np.linalg.matrix_power(A, 3))
```

g. Python 실행 결과

```
Rank of A: 3
Trace of A: 11
Determinant of A: -306.0
Inverse of A:
 [[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]
Matrix A raised to power 3:
 [[336 162 228]
 [406 162 469]
 [698 702 905]]

c:\Users\SAMSUNG\OneDrive\Desktop\대학\Solution 모음\25-1\수치해석\HW\HW_1>
```

1.12. (numpy 모듈 사용 예제) 아래의 코드를 실행한 후, 결과(출력)를 reporting하시오.

a. Python 코드

```python
import numpy as np
from numpy import linalg as LA
import warnings

# RuntimeWarning 무시 설정
warnings.filterwarnings("ignore", category=RuntimeWarning)

# 1. 벡터 a 생성
a = np.arange(9) - 4
print("Vector a:", a)

# 2. 행렬 b 생성 (3x3)
b = a.reshape((3, 3))
print("Matrix b:\n", b)

# 3. 벡터/행렬 norm 계산
print("\n--- Norms of vector a ---")
print("LA.norm(a):", LA.norm(a))
print("LA.norm(a, np.inf):", LA.norm(a, np.inf))
print("LA.norm(a, -np.inf):", LA.norm(a, -np.inf))
print("LA.norm(a, 1):", LA.norm(a, 1))
print("LA.norm(a, -1):", LA.norm(a, -1))
print("LA.norm(a, 2):", LA.norm(a, 2))
print("LA.norm(a, -2):", LA.norm(a, -2))
print("LA.norm(a, 3):", LA.norm(a, 3))
print("LA.norm(a, -3):", LA.norm(a, -3))

print("\n--- Norms of matrix b ---")
print("LA.norm(b):", LA.norm(b))
print("LA.norm(b, 'fro'):", LA.norm(b, 'fro'))
print("LA.norm(b, np.inf):", LA.norm(b, np.inf))
print("LA.norm(b, -np.inf):", LA.norm(b, -np.inf))
print("LA.norm(b, 1):", LA.norm(b, 1))
print("LA.norm(b, -1):", LA.norm(b, -1))
print("LA.norm(b, 2):", LA.norm(b, 2))
print("LA.norm(b, -2):", LA.norm(b, -2))
# 4. axis 기반 벡터 norm
c = np.array([[1, 2, 3], [-1, 1, 4]])
print("\n--- Norms with axis (matrix c) ---")
print("c:\n", c)
print("LA.norm(c, axis=0):", LA.norm(c, axis=0))
print("LA.norm(c, axis=1):", LA.norm(c, axis=1))
print("LA.norm(c, ord=1, axis=1):", LA.norm(c, ord=1, axis=1))

# 5. 다차원 행렬 norm
m = np.arange(8).reshape(2, 2, 2)
print("\n--- Norms of 3D matrix m ---")
print("m:\n", m)
print("LA.norm(m, axis=(1,2)):", LA.norm(m, axis=(1, 2)))
print("LA.norm(m[0, :, :]):", LA.norm(m[0, :, :]))
print("LA.norm(m[1, :, :]):", LA.norm(m[1, :, :]))
```

b. Python 출력

```
Vector a: [-4 -3 -2 -1  0  1  2  3  4]
Matrix b:
 [[-4 -3 -2]
 [-1  0  1]
 [ 2  3  4]]

--- Norms of vector a ---
LA.norm(a): 7.745966692414834
LA.norm(a, np.inf): 4.0
LA.norm(a, -np.inf): 0.0
LA.norm(a, 1): 20.0
LA.norm(a, -1): 0.0
LA.norm(a, 2): 7.745966692414834
LA.norm(a, -2): 0.0
LA.norm(a, 3): 5.848035476425731
LA.norm(a, -3): 0.0

--- Norms of matrix b ---
LA.norm(b): 7.745966692414834
LA.norm(b, 'fro'): 7.745966692414834
LA.norm(b, np.inf): 9.0
LA.norm(b, -np.inf): 2.0
LA.norm(b, 1): 7.0
LA.norm(b, -1): 6.0
LA.norm(b, 2): 7.348469228349534
LA.norm(b, -2): 1.8570331885190565e-16

--- Norms with axis (matrix c) ---
c:
 [[ 1  2  3]
 [-1  1  4]]
LA.norm(c, axis=0): [1.41421356 2.23606798 5.        ]
LA.norm(c, axis=1): [3.74165739 4.24264069]
LA.norm(c, ord=1, axis=1): [6. 6.]

--- Norms of 3D matrix m ---
m:
 [[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
LA.norm(m, axis=(1,2)): [ 3.74165739 11.22497216]
LA.norm(m[0, :, :]): 3.7416573867739413
LA.norm(m[1, :, :]): 11.224972160321824
```

c. 코드 설명 (a: vector, b: matrix)

LA.norm(a)	Euclidean Norm (2-norm)
LA.norm(a, np.inf)	Infinity Norm (max-norm)
LA.norm(a, -np.inf)	Min-norm
LA.norm(a, 1)	1-norm
LA.norm(a, -1)	정의상 가능한 값이 아님.
LA.norm(a, 2)	Euclidean Norm (= LA.norm(a))
LA.norm(a, -2)	역수 합의 역수 형태의 특수한 norm. 값은 정의 가능하지만, 값이 매우 작음.
LA.norm(a, 3)	$(\sum \lvert a_i \rvert^3)^{1/3}$
LA.norm(a, -3)	역수 형태의 특수한 p-norm. 수식상 가능하지만, 값이 매우 작게 나옴.
LA.norm(b)	Frobenius Norm
LA.norm(b, 'fro')	Frobenius Norm
LA.norm(b, np.inf)	행 단위 절댓값 합의 최댓값
LA.norm(b, -np.inf)	행 단위 절댓값 합의 최솟값
LA.norm(b, 1)	열 단위 절댓값 합의 최댓값
LA.norm(b, -1)	열 단위 절댓값 합의 최솟값
LA.norm(b, 2)	Spectral Norm (가장 큰 singular value)
LA.norm(b, -2)	가장 작은 Singular Value
LA.norm(c, axis = 0)	열 단위 2-norm
LA.norm(c, axis = 1)	행 단위 2-norm
LA.norm(c, ord = 1, axis = 1)	행 단위 1-norm
LA.norm(m, axis = (1, 2))	각 2×2 행렬의 Frobenius Norm
LA.norm(m[0,:,:]), LA.norm(b[1,:,:])	각 2×2 행렬의 Frobenius Norm