

## ha4\_1.py

### 1. Code:

```
ha4_1.py - C:\Users\wsomin\Desktop\ha4_1.py (3.10.7)
File Edit Format Run Options Window Help
# global variable to count the number of comparison
# for each finding algorithm
global comparisonAlgo1
global comparisonAlgo2
comparisonAlgo1 = 0
comparisonAlgo2 = 0

def MaxMin1(A):
    # use global variable to keep track of the count of comparison
    global comparisonAlgo1
    # initialize min and max as the first element of the array
    maxNum = A[0]
    minNum = A[0]

    # compare all element with current min, max
    # and update if the element is smaller than min or bigger than max
    for i in range(1, len(A)):
        comparisonAlgo1 += 2
        if maxNum < A[i]:
            maxNum = A[i]
        if minNum > A[i]:
            minNum = A[i]
    return minNum, maxNum

def MaxMin2(A, i, j):
    # use global variable to keep track of the count of comparison
    global comparisonAlgo2

    # when there is one element, max and min is both the element
    # no comparison needed
    if i == j:
        return A[i], A[i]
    # when there is two elements, max is bigger element and min is smaller element
    # one comparison needed
    elif (j - i) == 1:
        if A[i] < A[j]:
            comparisonAlgo2 += 1
            return A[i], A[j]
        else:
            comparisonAlgo2 += 1
            return A[j], A[i]
    # when there is two or more elements, divide the array in to two arrays
    # and get min and max for each subarray
    # then, compare two mins and maxes to get the min and max for this level
    # 2(n/2)+2 comparisons needed
    else:
        minLeft, maxLeft = MaxMin2(A, i, (i+j)//2)
        minRight, maxRight = MaxMin2(A, (i+j)//2+1, j)
        comparisonAlgo2 += 2
        return (min(minLeft, minRight), max(maxLeft, maxRight))

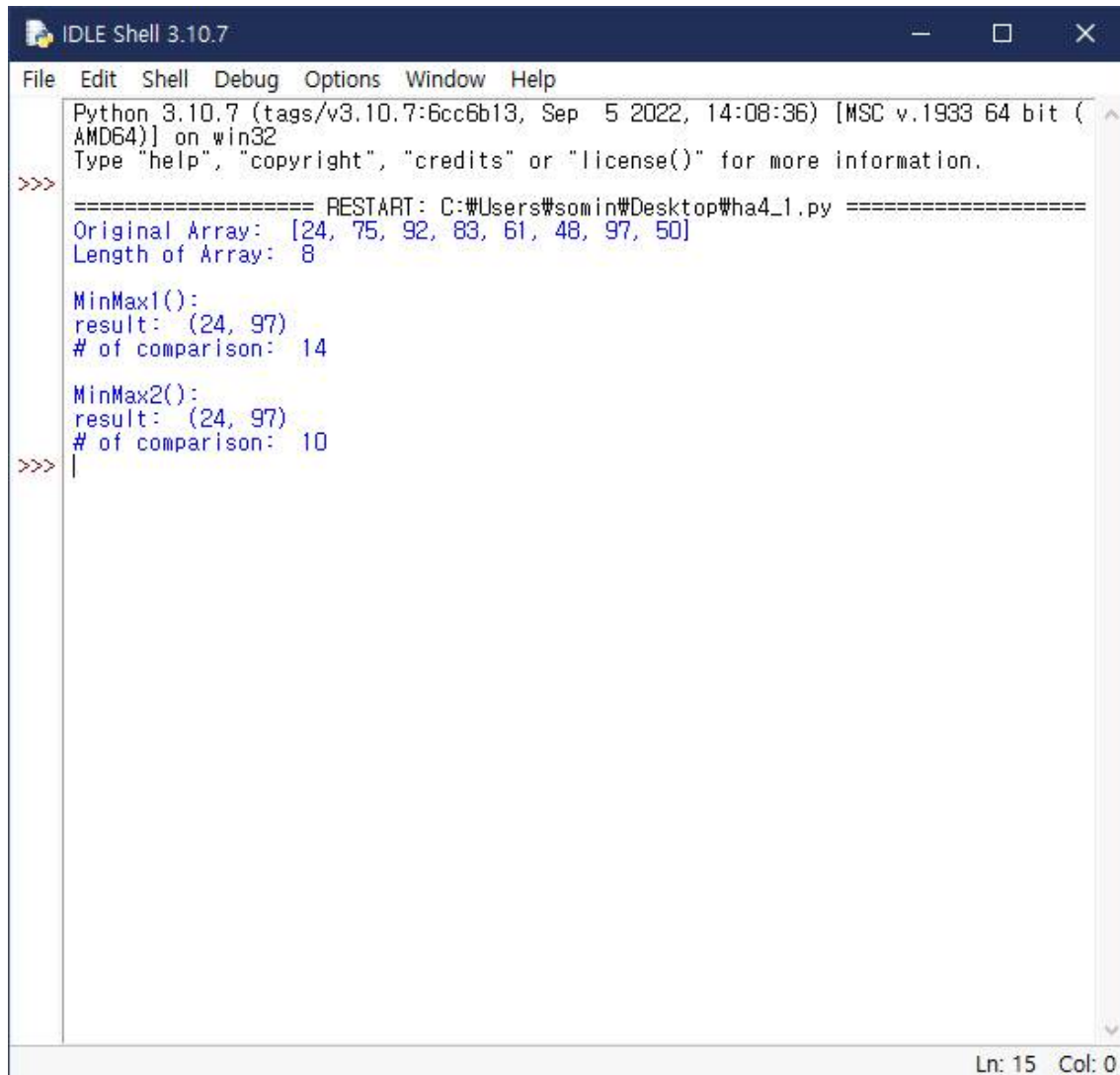
if __name__ == '__main__':
    A = [24, 75, 92, 83, 61, 48, 97, 50]

    print("Original Array: ", A)
    print("Length of Array: ", len(A))

    # First Algorithm
    # number of comparison: 2n-2
    print("#nMinMax1(): ")
    print("result: ", MaxMin1(A))
    print("# of comparison: ", comparisonAlgo1)
    # Second Algorithm
    # number of comparison: 3/2n-2
    print("#nMinMax2(): ")
    print("result: ", MaxMin2(A, 0, len(A)-1))
    print("# of comparison: ", comparisonAlgo2)
```

Ln: 12 Col: 17

## 2. Result:



```
Python 3.10.7 (tags/v3.10.7:6cc6b13, Sep 5 2022, 14:08:36) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\somin\Desktop\ha4_1.py =====
Original Array: [24, 75, 92, 83, 61, 48, 97, 50]
Length of Array: 8

MinMax1():
result: (24, 97)
# of comparison: 14

MinMax2():
result: (24, 97)
# of comparison: 10
>>> |
```

## 3. Time Complexity (by number of comparisons)

### 1) MaxMin1()

이 함수는 최댓값과 최솟값을 단순 linear 비교를 통해 구한다.  
따라서 max와 min의 초깃값으로 정한 0번 원소를 제외한 각 원소에 대해 두 번의 비교연산이 수행된다. 즉, 데이터의 크기가  $n$ 일 때  $2(n-1)$ 이다.  
이를 수식으로 나타내면  $T(n) = 2n-2$  이다.

### 2) MaxMin2()

이 함수는 최댓값과 최솟값을 divide and conquer 방식을 통해 구한다.  
데이터의 크기가  $n$ 일 때  $n/2$  크기의 subproblem 두 개를 수행하고 두 subarray의 min, max 값 중 최소와 최대를 결정하기 위해 추가로 2번의 비교 연산을 수행한다. 따라서 데이터의 크기가  $n$ 일 때  $T(n) = 2T(n/2)+2$ 이다. 이를 재귀적으로 풀어 점화식을 일반식으로 바꾸면  $T(n) = (3/2)*n-2$ 이다.

**4. Explain why one type performs efficiently than the other.**

MaxMin2()함수가 MaxMin1()함수보다 더 효율적 퍼포먼스를 수행하는 이유는 선형적으로 모든 원소를 탐색하며 비교하는 MaxMin1() 대신 MaxMin2()는 원소를 2개씩 비교하며 problem size를 재귀적으로 반으로 줄이기 때문이다. 따라서 MaxMin2()는 각 원소에 대해 MaxMin1()보다 적은 수의 비교 연산을 수행한다.