프로그래밍언어론 HW1 – B935277

유송경

I. 과제1 : Binary Search

```
    def binary_search(left, right):
        while left <= right:
            mid = (left + right) // 2
            if n[mid] == ans:
                return mid
            elif n[mid] > ans:
                right = mid - 1
            elif n[mid] < ans:
                left = mid + 1
        return -1

    n=list(map(int,input("Input : ").split()))

    ans=input()

    l=len(n)

    n.sort()
    left = 0
    right = l-1
    if binary_search(left,right) == -1 :
        print("Output : None")
    else :
        num=binary_search(left,right)
        print("Output : "+str(num+1))
```

리스트에 숫자를 입력 받은 뒤 몇번째 순서에 그 수가 있는지 ”Binary Search”로 찾는 과제이다. 이진탐색이란 리스트의 숫자들이 정렬되어있다는 전제하에 진행 된다. 첫번째 수를 left, 마지막 수를 right로 설정한 뒤 가운데에 위치한 수(left+right/2)가 찾고자 하는 수보다 큰지, 작은지 비교한뒤 이를 계속 반복해 나간다. 있다면 몇번째 순서인지 , 없다면 None 을 출력하는 것으로 끝낸다.

II. 과제2 : Quick Sort

```
     def partition(arr,l,r):
        pivot=arr[l]
        i=l+1
        j =r
        while True:
            while i<=r and arr[i]<=pivot:
                i+=1
            while l<=j and arr[j]>pivot:
                j-=1
            if j>=i:
                arr[i],arr[j]=arr[j],arr[i]
            else:
                break
        arr[l],arr[j]=arr[j],arr[l]
        return j

    def quicksort(arr,l,r):
        if l<r:
            pivot=partition(arr,l,r)
            quicksort(arr,l,pivot-1)
            quicksort(arr,pivot+1,r)

    n=list(map(int,input("Input : ").split()))
    quicksort(n,0,len(n)-1)
    print("Output :", n)
```

리스트를 입력 받은뒤 Quick sort로 정렬하는 과제이다. pivot를 첫번째로 설정하거나, 가운데로 설정하거나 주로 두가지 방법이 있다. 나는 리스트의 첫번째를 pivot으로 설정하여 정렬하였다. pivot보다 작은지, 큰지를 따진뒤 크다면 뒤로 , 작다면 앞으로 설정하여 정렬을 하는 알고리즘이다.

III. 과제3 : Merge Sort

```
       def mergesort(arr):
        n = len(arr)
        if n <= 1:
            return

        m = n // 2
        g1 = arr[:m]
        g2 = arr[m:]
        mergesort(g1)
        mergesort(g2)

        l = 0
        r = 0
        k = 0
        while l < len(g1) and r < len(g2):
            if g1[l] < g2[r]:
                arr[k] = g1[l]
                l += 1
                k += 1

            else:
                arr[l] = g2[l]
                r += 1
                k += 1
        while l < len(g1):
            arr[k] = g1[l]
            l += 1
            k += 1

        while r < len(g2):
            arr[k] = g2[k]
            r += 1
            k += 1

    n=list(map(int,input("Input : ").split()))
    mergesort(n)
    print("Output :", n)
```

리스트를 입력받은뒤 merge sort하는 과제이다. 머지 정렬은 리스트를 하나의 숫자가 남을 때까지 나눈 뒤 병합을 하면서 정렬해 나가는 방식이다. 즉 머지정렬은 분해, 정복, 결합 세가지 단계로 이루어 진다고 표현할 수 있다. 추가적으로 리스트가 필요하다. 두개의 리스트를 처음부터 하나씩 고민하여 작은 값을 새로운 리스트(arr)로 옮긴다. 이 과정을 되풀이 한뒤 하나의 리스트가 끝나게 되면 끝이나게 된다.

IV. 과제4 : Tree Traverse

```
      class Node:
        def __init__(self,num):
            self.num=num
            self.left=None
            self.right=None
    class Bin_tree():
        def __init__(self):
            self.root=None

        def preorder(self,node):
            if node==None:
                return
            print(node.num)
            self.preorder(node.left)
            self.preorder(node.right)

        def inorder(self,node):
            if node==None:
```

```
                return
            else:
                self.inorder(node.left)
                print(node.num)
                self.inorder(node.right)

        def postorder(self,node):
            if node==None:
                return
            self.postorder(node.left)
            self.postorder(node.right)
            print(node.num)

    hw4=Bin_tree()
    n1=Node(15)
    n2=Node(1)
    n3=Node(37)
    n4=Node(61)
    n5=Node(26)
    n6=Node(59)
    n7=Node(48)

    hw4.root=n1
    n1.left=n2
    n1.right=n3
    n2.left=n4
    n2.right=n5
    n3.left=n6
    n3.right=n7

    print('Preorder Traverse\n')
    hw4.preorder(hw4.root)

    print('Inorder Traverse\n')
    hw4.inorder(hw4.root)

    print('Postorder Traverse\n')
    hw4.postorder(hw4.root)
```

클래스를 이용해서 트리를 만들었다. 모든 노드를 None으로 초기화 하였다. 차례대로 left,right를 설정해 주었고, 재귀를 이용해서 전위,중위,후위 순회를 하였다. 재귀도중 None인 노드를 만난다면 끝낸다.

V. 과제5 : Classroom Assignment

```
    def greedy(meeting):
        cnt=0
        l=0
        ans=[]
        for item in meeting:
            if item[1]>=l:
                l=item[2]
                ans.append(item[0])
                cnt+=1
        print(cnt)
        print(ans)

    num=int(input("Input :"))
    meeting=[]
    for i in range(num):
        num,start,end=map(int,input().split())
        meeting.append((num,start,end))

    meeting=sorted(meeting, key=lambda item:
    item[1])
    meeting=sorted(meeting, key=lambda item:
    item[2])
    greedy(meeting)
```

강의실 배정문제로, 우선 강의들의 시작 시간을 오름차순으로 정렬을 한뒤, 강의들의 종료 시간을 오름차순으로 정렬을 하였다. 그 후 시작 시간이 가장 빠른 것을 l, 시작시간이 l인 강의 중 종료시간이 가장 빠른 시간을 l로 다시 초기화 한다. 새로 초기화된 l로 시작시간이 l보다 커질 때까지 위 과정을 반복한다.

VI. 과제6 : Data Blocking

```
    import re
    def program(int):
        pattern = re.compile('(100{1,}1{1,}|01)
    {1,}')
        result = pattern.fullmatch(int)
        if result == None:
            return -1
        else:
            return 1

    num = int(input("Input :"))
    search = []
    for i in range(num):
        code = input()
        search.append(code)

    print("Output : ")

    for i in range(num):
        cmp = search[i]

        if program(cmp) == 1:
            print("DANGER")
        else:
            print("PASS")
```

암호화 되지 않은 데이터를 차단하는 프로그램을 만드는 과제였다. 주어진 패턴은 (100 1 —01) 로 정규표현식을 이용하여 (1001,11,—01)1, 이처럼 만들어 보았다. 100,1을 한번이상 반복한 수와 01 중 아무거나 한 번이상 반복되는 경우의 수를 구하게 하였다.