

Homework #5

(Introduction to Data Structures)

Due date: May 31, 2018

학번: 2014310407

이름: 이 준 혁(JunHyuk Lee)

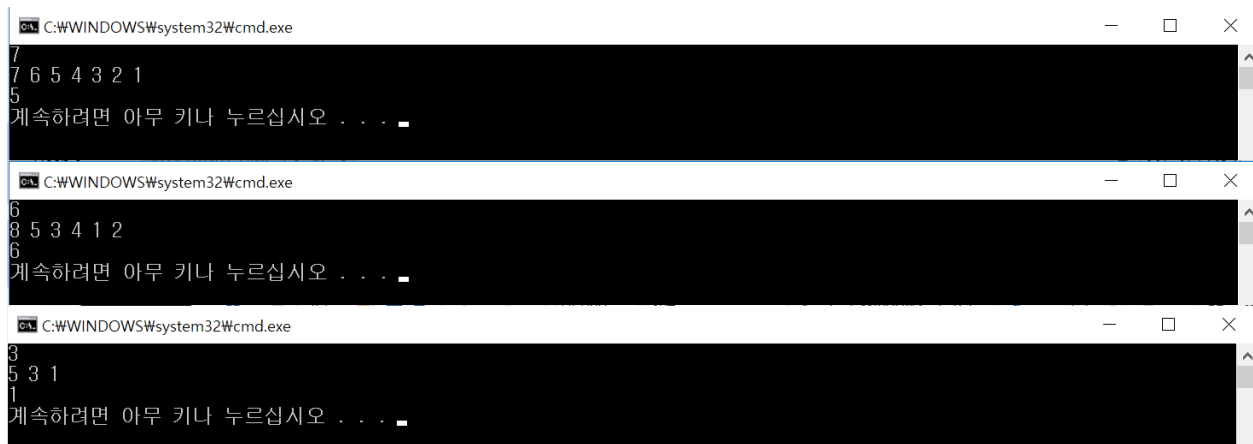
1. Number of Swap Operations

1.1. Solution

우선 입력을 모두 heap 에 삽입했다. 그리고 heap 의 가장 위의 item 을 매 번씩 지워 나갔다. 이 때, 가장 마지막 원소를 가장 위에 올린 후에, 아래 노드들과 비교해 아래 노드의 priority 가 더 높다면 부모와 자식을 바꾸는데, 이 횟수가 몇 번인지를 세는 것이 목표이므로, 자식의 priority 가 더 클 때마다 swap 을 나타내는 변수를 1 씩 증가시켰다.

이렇게 heap 에 원소가 하나가 남을 때까지 반복했고, swap 을 나타내는 변수를 반환했다.

1.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
7
7 6 5 4 3 2 1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
6
8 5 3 4 1 2
계속하려면 아무 키나 누르십시오 . . .

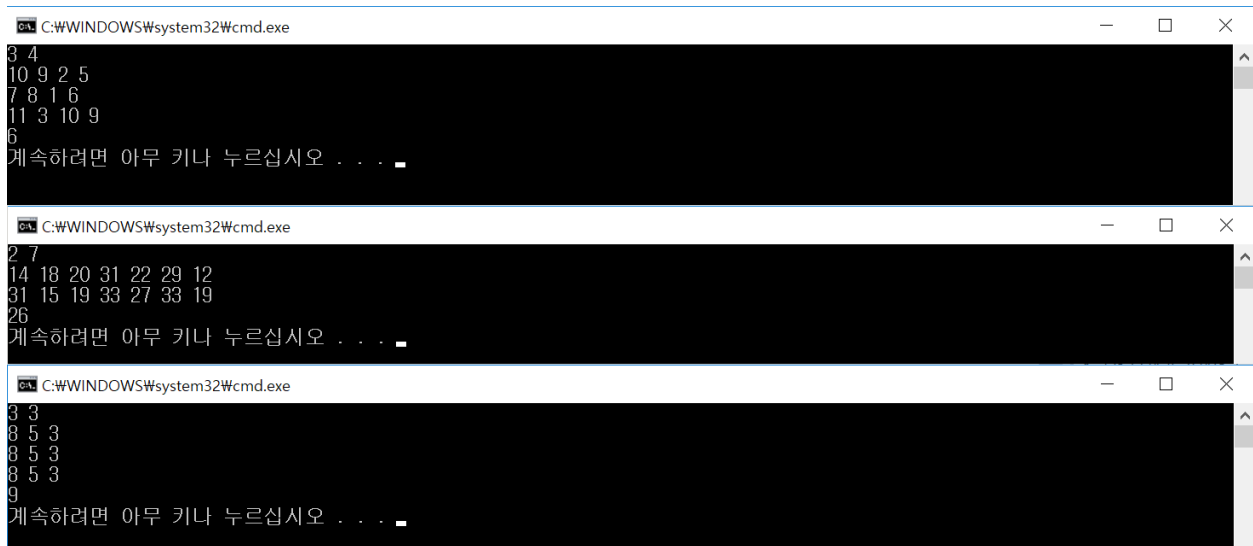
C:\WINDOWS\system32\cmd.exe
3
5 3 1
계속하려면 아무 키나 누르십시오 . . .
```

2. Finding Top k Smallest Elements

2.1. Solution

우선 입력 이차원 배열을 모두 heap 에 넣었다. 이 때 작은 수를 먼저 뽑아내야 했으므로, priority 를 data 값의 음수 값으로 설정했다. 이렇게 되면 가장 작은 수가 가장 위로 올라오는 모양이 된다. 즉, heap 에서 삭제를 진행하면 가장 작은 원소를 반환 받게 된다. k 번 만큼 deletion 을 진행해서 나온 결과값을 모두 더한 후, 그 값을 반환했다. 이는 k 개의 가장 작은 원소들의 합이다.

2.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
3 4
10 9 2 5
7 8 1 6
11 3 10 9
6
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
2 7
14 18 20 31 22 29 12
31 15 19 33 27 33 19
26
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
3 3
8 5 3
8 5 3
8 5 3
9
계속하려면 아무 키나 누르십시오 . . .
```

3. Median in BST

3.1. Solution

우선 BST의 노드가 총 몇 개 있는지 알기 위해 Binary tree의 노드의 개수를 recursion으로 구하는 함수를 만들었다. 그리고 BST의 모든 노드를 array 형식으로 저장하기 위해, inorder로 BST를 탐색하며 트리의 모든 노드를 배열에 저장했다. 이 때 inorder로 트리를 탐색하면 데이터가 정렬된다는 점을 이용했다.

이후 노드의 개수가 홀수이면 배열의 중앙에 있는 값을 반환했고, 짝수이면 배열의 중앙에 있는 두 값의 합을 반환했다.

3.2. Result (snapshot)



```
C:\WINDOWS\system32\cmd.exe
10
2 8 6 12 4 20 18 16 14 10
22
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
1
5
5
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
7
10 32 61 55 24 31 47
32
계속하려면 아무 키나 누르십시오 . . .
```

4. BSTs with same Structures

4.1. Solution

우선 만들어진 두 함수가 같은 구조인지 판별하기 위해 다음과 같은 함수 세 개를 사용했다.

1. 왼쪽 노드가 있는지 판별하는 함수

2. 오른쪽 노드가 있는지 판별하는 함수

3. 리프인지 판별하는 함수

위와 같은 세 개의 함수를 구현한 후 다음과 같은 과정으로 재귀적으로 함수를 구성했다.

0. 왼쪽 구조가 똑같은지를 나타내는 변수 `left` 와 오른쪽 구조가 똑같은지를 나타내는 변수 `right` 선언 후 `false` 로 초기화

1. 두 노드가 모두 리프인 경우, `true` 를 반환

2. 두 노드가 모두 왼쪽 자식이 있는 경우 왼쪽 자식에 대해 함수를 재귀적으로 호출 후 값을 `left` 에 저장

3. 둘 다 왼쪽 자식이 없는 경우 `left` 값을 `true` 로 저장

4. 두 노드가 모두 오른쪽 자식이 있는 경우 오른쪽 자식에 대해 함수를 재귀적으로 호출 후 값을 `right` 에 저장

5. 둘 다 왼쪽 자식이 없는 경우 `right` 값을 `true` 로 저장

만약 한 쪽만 왼쪽 자식이 있는 경우, 즉 구조가 다른 경우에는, 어느 경우에도 해당하지 않으므로, `left` 에 그대로 `false` 값이 남게 된다. 이렇게 위 경우에 해당하지 않는 경우에는 변수 값에 `false` 가 들어가 있게 된다. 두 변수 중 하나라도 `false` 이면 두 `tree` 의 구조가 다르다는 말이므로, `left && right` 를 반환했다. 즉, 두 변수가 모두 `true`, 즉 모든 구조가 같을 때만 `true` 를 반환하도록 한 것이다.

이를 두 트리에 대해 시행하면 두 트리가 같은 지 여부가 출력된다.

4.2. Result (snapshot)

```
C:\WINDOWS\system32\cmd.exe
5
1 4 3 2 5
1 2 3 4 5
0
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
7
1 3 5 7 9 11 13
2 3 4 5 6 7 8
1
계속하려면 아무 키나 누르십시오 . . .

C:\WINDOWS\system32\cmd.exe
5
5 4 6 2 7
5 6 4 7 2
1
계속하려면 아무 키나 누르십시오 . . .
```

5. Number of Identical Trees

5.1. Solution

이 함수는 수학적으로 접근했다.

- 어떤 노드가 왼쪽 자식이 있으면, 왼쪽 자식에서 나올 수 있는 identical tree 의 개수를 left, 그와 관계없이 왼쪽 자식의 노드의 수를 left_num 이라 한다.
- 오른쪽 노드에 대해서도 같은 방식으로 right 와 right_num 을 설정한다.
- 총 노드의 수를 num_root 라 한다. (이는 left_num + right_num + 1 이다)

이 경우, identical tree 의 개수는 left_num 개의 원소와 right_num 개의 원소를 정렬하는 수와 같은데, left_num, right_num 개의 원소 내부에서는 순서를 지키며 정렬해야 한다.

이 값은 $(num_root - 1)! / ((left_num)! * (right_num)!)$ 이다.

그리고 이 값에다 왼쪽 자식 내부에서 원소를 정렬하는 경우의 수는 left, 오른쪽에서는 right 이다.

따라서, $(num_root - 1)! / ((left_num)! * (right_num)!)$ * left * right 를 반환해서 재귀적으로 함수를 구성했다. 이 결과, 모든 identical tree 의 개수를 얻을 수 있었다

5.2. Result (snapshot)



The image shows three stacked Windows command prompt windows, each titled "C:\WINDOWS\system32\cmd.exe". Each window displays a sequence of numbers followed by a Korean prompt.

Top window:

```
4
10 7 9 20
3
계속하려면 아무 키나 누르십시오 . . .
```

Middle window:

```
7
4 2 6 1 3 5 7
80
계속하려면 아무 키나 누르십시오 . . .
```

Bottom window:

```
9
5 3 7 1 2 6 8 9 4
630
계속하려면 아무 키나 누르십시오 . . .
```