Data Structure

Lab Session #09: Internal Sorting 2

U Kang
Seoul National University

Goals

- Implement “**Heap Sort**”
  - Complete “MinHeap” class in MinHeap.java
  - Complete “HeapSort” class in HeapSort.java

- Print the sample output corresponding to the sample input
  - Please carefully observe the I/O specification.

Notice

- After implementing "HeapSort", check if your program works well.
  - Check sample input and output files in the 'testcase' folder.
  - Test your program by using it.
- When you finish implementing the program, you can leave.
  - But, you need to stay for at least an hour.
- Check your attendance.

Build a project

- Download the project for this lab from eTL.
- Extract the project, and open it in IntelliJ.
  - See the slide of 1st lab session to check how to open the project in IntelliJ.

Heap Sort

- Given a series of integers, build min-heap.
- Extract values from the top until the tree empty.
- Refer to lec12 if you don't remember heap.

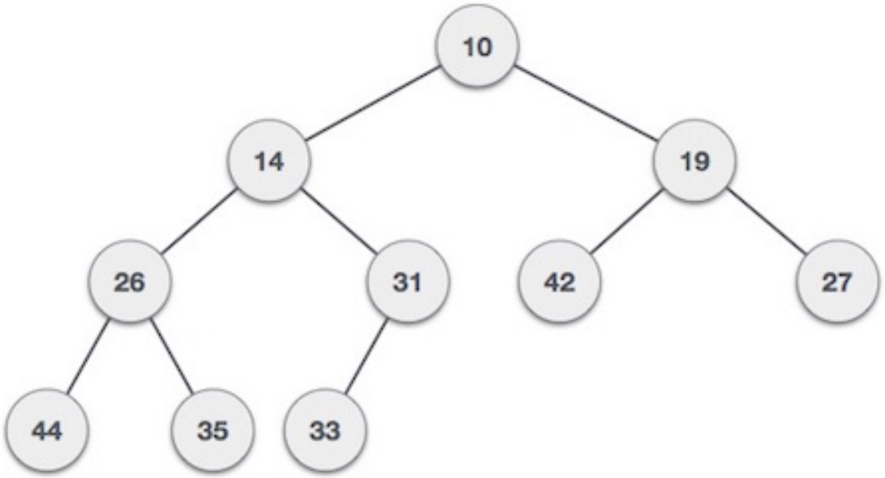

Figure. Min-heap

Function that you need to fill in

- In 'MinHeap':
  - insert
  - remove
  - find
  - siftdown
  - removeMin
- In 'HeapSort':
  - add
  - remove
  - sort

I/O Specification

- add

Input format	Output format
`add 3`	`(heap_state)`

Description
- **Add a value into the min-heap.** - **Value is an integer.**

Example Input	Example Output
`add 3`	`3`
`add 5`	`3 5`

I/O Specification

- remove

Input format	Output format
`remove`	`(heap_state)`
Description	
- Remove a value from the min-heap. - Value is an integer.	

Example Input	Example Output

`remove 15`	`1 1 9 3 10 15 20 5 (last state)`
	`1 1 5 3 10 9 20 (after remove 15)`

I/O Specification

- sort

Input format	Output format

`sort_a/sort_d`	`(sorted integers)`
	`(heap_state)`

Description
- Print out values in ascending(sort_a)/descending(sort_d) order using the min-heap. - (Sorted integers) is a sequence of integers sorted in ascending/ descending order. - After sorting, heapify all elements again and print heap_state.

Example Input	Example Output

`sort_a/sort_d`	`1 3 5 9 10 15` / `20 9 7 5 3 1 1`
	`1 3 9 5 10 15` / `1 3 1 5 9 7 20`

Sample Input

```
add     3
add     5
add     15
add     9
add     10
add     1
sort_a
add     20
add     1
remove  15
add     7
remove  10
sort_d
```

Sample Output

```
op        operand   heap_state
add       3         3
add       5         3 5
add       15        3 5 15
add       9         3 5 15 9
add       10        3 5 15 9 10
add       1         1 5 3 9 10 15
sort_a    1 3 5 9 10 15
                    1 3 9 5 10 15
add       20        1 3 9 5 10 15 20
add       1         1 1 9 3 10 15 20 5
remove    15        1 1 5 3 10 9 20
add       7         1 1 5 3 7 9 20
remove    10        1 1 5 3 7 9 20
sort_d    20 9 7 5 3 1 1
                    1 3 1 5 9 7 20
```

Heap_state may vary depending on how you heapify after sorting, but you have to make sure every heap_state follows heap property.

Questions?