School of Computing
National University of Singapore
CS2010 Data Structures and Algorithms 2
Semester 2, AY 2015/16

Lab 2 – **kselect-heap**
Assigned – Feb 2nd (Tuesday), 2016
Lab – Feb 15 (Monday by 11.59pm), 2016
There is no lab on Feb 8 because of CNY holiday. Lab session on Feb 15 will be consultation if you need help, or would like to demo your lab code for verification.

Objective
Implement a new data structure, named kselect-heap, which has an access time to the k^{th} element in a collection of elements in **O(1)** time. The kselect-heap uses **two heaps (priority queues)**: a min-heap and a max-heap.

Preparation
There is a data structure called a *median-heap* that allows access to median of the collection of elements in O(1) time. The median of a list of (comparable) elements, is the middle element when the list is sorted from smallest to largest item (ascending order). If the list has an odd size, the median is the middle element in the list. If the list size is even, the median is often computed as the average of the two middle elements.

The following is a description of one way to implement a median-heap. You maintain two heaps: (1) a min-heap and (2) a max-heap. Both heaps start out empty.

Insert: (Step 1 – Insert) When a new element is added to the median-heap, check to see if the new element is smaller than the max-heap root. If it is smaller, add it to the max-heap. If not, add it to the min-heap.

(Step 2 – Heap balancing) After insertion, if the size between the two heaps differs by more than 1, remove the root from the larger-size heap and add it to the smaller-size heap. This will effectively balance the data.

Find Median: If the two heaps have equal size, return the average of the two heap's roots, otherwise, return the root of the larger heap. Note that this implementation does not delete any nodes from the heaps.

Implementing a median heap was originally going to be your lab 2, but then we found that it was already implemented on Stack Overflow! See here: http://stackoverflow.com/questions/15319561/how-to-implement-a-median-heap.

Your Task
Your task is to implement a kselect-heap that allows the kth element to be found in O(1) time. This is actually a variation on the median-heap, so the implement in the Stack Overflow site should help, however, unlike the median-heap, your help should able to delete the kth element. Your kselect-heap should provide the following operations:

kSelectHeap Operations

```
kselectHeap( int k)        // create a kselect-heap that finds the kth element
insert( element )          // insert a comparable element
element peek()             // returns the kth element (but do not delete the element)*
element delete()           // returns the kth element and removes it from the kselect-heap*
isEmpty()                  // is kselect-heap empty?
```

** if kth element doesn't exist (e.g. list has less than k elements), return a null (for delete - don't delete any item)*

Get started:

Download the file lab2.zip from IVLE into your working directory. Uncompress the file and you should find the following document and data files: **CS2010-Lab2.pdf**, **input1.txt, input2.txt, input3.txt** and **input4.txt**.

For this lab you are not given any Java source files.

Task:

You can implement this however you like, e.g. using the lecture note's max-heap implementation or the Java built in heap implementation (PriorityQueue), or your own heap implementation. You are also welcome to modify the Median Heap code on the stackoverflow page. To test your kselect-heap implementation, you need to write a main function that reads in a text file and performs inserts, peeks, and delete operations as follows:

The text files are encoded in the following manner:

```
[5, 9, 8, 11, 55, 66, 6, +, 3, 1, +, 0, 2, -, -, 4, 5, 6, +]
```

The first element in the text file is the k for the min-heap. In this case here, k=5. After you have read in the first integer (k), each integer after that should be inserted into the kselect-heap. If you encounter a + character, call the method `peek()` on the kselect-heap and print the result to the console. If you encounter a - character, call `delete()` on the heap and print the result to the console. For both `peek()` and `delete()` if the result is null, print the string "null".

Example output for the list above (k=**5** – **do not insert 5 into the median-heap**):

```
55 – peek     (55 is the 5th element after the insertion of 9,8,11,55,66,6)
9 – peek      (9 is the 5th element after additional insertion of 1, 3)
6 – delete    (6 is the 5th element after insertion of 0, 2; 6 is now deleted)
8 – delete    (8 is the 5th element after 6 was deleted; 8 is now deleted)
4 – peek      (4 is the 5th element after 4, 5, 6 were inserted)
```

Note: you don't have to print the text in (..)

Submission Instruction (IVLE submission) – Due Feb 15 (Monday by 11.59pm), 2016

Please submit your code to IVLE. You only need to zip up your source code.

1. Please put your Java codes in a folder with the source files and submit the entire folder zipped. Use the following convention to name your zipped folder: MatriculationNumber_yourName_Lab2. For example, if your matriculation number is A1234567B, and your name is Chow Yuen Fatt, for this assignment, your file name should be A1234567B_ChowYuenFatt_Lab2.zip. Points will be deducted if you don't follow the naming scheme.
2. It is up to you to test to make sure that your Java code in the zipped file is complete. It is recommended that you unzip your Java code in a different location and test it to make sure it runs.