## Problem 8:

8.a)

Place I updated in "Heap.java":

(Line 93~97, added:)

**void** Heap\_UpdateKey(**int** i, **int** k){

A.set(i, k);

shiftDown(i);

shiftUp(i);

}

8.b)

The asymptotic complexity is O(log n).

Actually, this method either need to run shiftDown or shiftUp, Since the key changed would either larger than its parent(thus need to shiftUp) or less than its child(thus need to shiftDown).

When one shift is executed, then another shift simply take constant time.

Therefore the time complexity is O(log n).

## Problem 9:

There're two methods here(see the full implementation in the ):

1. by changing shiftUp/shiftDown method -- therefore it's necessary to change (vector A) and (int heapsize) to be protected
2. by changing the IO value -- the heap save the inverse of one number, thus the order of the numbers simply reversed

## Problem 10:

10.a)

the type of Priority queue depends on the comparator method of its contents.

If it's declared as "PriorityQueue<Integer>", then by default it's a Min-Heap.

10.b)

see the file "HeapSort.java"

## Problem 11:

11.a) say n(A) is the number of elements in A.

if n(A) = n(B)-1, and b is the minimum in B. Then there're exactly n(A) elements smaller, and n(B)-1 = n(A) elements larger than b; therefore b is the median;

if n(A) = n(B), suppose a is the maximum in A and b is the minimum in B, then same as the arguing before, a and b are both the median and a <= b. Since we are taking the larger of the two median, b is the one to choose.

In conclusion, this method is correct.

11.b) You can refer to the file "MedianTester.java" for test, and file "DynamicMedian.java" for the implementation

11.c)

Problem of this method: then the median would at most depend on the numbers which can first fill the 10 position of A and B.

After the median is decided, then whatever number greater than this number would simply replace one number in B, while whatever number less than this number would replace one number in A, but the median stay unchanged.

Therefore it won't work.

Example proving the wrongness of this method:

insert these numbers:

1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22

The median is 12; but according to this method, the median is 11(only counts the first 21 elements).

11.d)

You may search for the tag "// \* Changed here \*" in the file "DynamicMedian2.java". These are all the changes I made based on "DynamicMedian.java".

References:

1. For Problem 9, I discussed with Ding Mingzhe, and he actually gave me the second solution (which is very brilliant)
2. For the use of Comparator:

http://www.cs.wcupa.edu/~rkline/DS/prioqueue.html