Programming Assignment

a) Pseudo code of your implementation of the flexible priority queue ADT using a parameterized heap that is implemented using extendable array. Note that Java code will not be considered as pseudo code. Your pseudo code must be on a higher and more abstract level.

```
Interface entry<K,V>
        K getKey();
        V getValue();


Class PriorityQueue implements entry:
pqArr (array used in priority queue)
comparator


Constructor (String comp):
comparator = comp


Algorithm remove():
Input : void
Output : return the entry
If is empty then:
        Return null
Else
        temp = entry of root
        Swapping first entry and last entry
        Decrement array size decremented by 1
        Downheap(); // if needed
```

Return temp

Algorithm insert(K key, V value):

Input: key and value which will be stored in the array

Output: void

Make array size +1 and insert in the last entry in the extended array

Upheap(); //If needed

Algorithm top():

Input: void

Output: Current root element (entry)

If it is empty then

Return null

Else

Return root (first) element

Algorithm toggle():

Input, output : void

If comparator is min

switchToMax()

else

switchToMin()

Algorithm switchToMin():

Input, output : void

If comparator is not min:

    for i =0 to size():

        upheap(i);

Algorithm switchToMax():

Input, output : void

If comparator is not max:

    for i =0 to size:

        upheap(i);

Algorithm state():

    return comparator

Algorithm isEmpty():

    If (this.size() is 0)

        return true;

    else

        return false;

Algorithm size():

    Return length of pqArr;

Algorithm compare(Entry e1, Entry e2):

Input: comparing 2 entry in array which called e1 and e2

Output: 1 or -1 depending on max and min heap

If comparator is “min” then

    If (e1.compareKey(e2))>=0)

        Return 1

    Else

        Return -1

Else

    If (e1.compareKey(e2))<=0)

        Return 1

    Else

        Return -1

// In Entry class

Algorithm compareKey(Entry ent):

Input: compared ent

Output: 1 or -1 depending on which one is bigger

If this.key>= ent.key then

    Return 1

Else

    Return -1

Algorithm upheap(int i):

input: element location which wants to do upheap

output: void

while (i>0):

int p = location of parent element in array

if (compare(pqArr[i] and pqArr[p]) >=0)

break;

swap(i,p)

let i is p (In this way, when i reaches 0 (which is root), while loop stops)

Algorithm downheap(int i):

Input: element location which wants to do downheap

Output: void

While (pqArr[i] has left child):

compare(pqArr[left] and pqArr[right]) and temp is which child of ever smaller (min case) or larger (max case).

If compare (pqArr[temp] and pqArr[i] >=0)

Break;

Swap (i, temp)

i is temp

For upheap and downheap, it will verify whether current state is min or max with compare function. In compare function, the keys are compared and if it's min case, if the first value is larger, returns positive number and if it's max case, if the first value is smaller, returns positive number. In this way we avoid duplication.

For extendable array, this is required for insert. Array is not extendable in java so simply create new array with size is incremented 1 more and copying the array and insert new value in the las element. For detailed code, please check actual Java code.

b) Tight big-Oh time complexities of toggle(), switchToMin(), and switchToMax() operations, together with your explanations.

---

toggle() is call function either switchToMin or switchToMax() and both function is almost the same except to know if it's max or min heap.

In pseudo code, I used upheap from root to the lasts node. Total calling time is total number of elements which is n. It is also same as size of pqArr (array). And when we check upheap time complexity, in the worst case, it will run log(n) times since it is binary tree.

Therefore, time complexity is **O(n*log(n))** for toggle(), switchToMin(), and switchToMax().

c) Well documented Java source code with 20 different but representative examples demonstrating the functionality of your implemented ADT. These examples should demonstrate all cases of your ADT functionality (e.g., all operations of your ADT, several cases requiring automatic array extension, sufficient number of down and up-heap operations).

Please check java code