a) Pseudo code of replaceKey(e, k), state(), peekAt(n), and merge(otherAPQ) methods

```
K replaceKey(target, K key)
        Index ← -1;
        For (int i = 0; i< size; i++)
                If (heap[i] == target)
                        Index ← i;
        If (index == -1)
                Throw exception //never found target, hence index remaining -1

        K temp = target.key;   //store old key
        target.setKey(key);     //put new key
        upheap(index);
        downheap(index);       //changing key might violate heap order, so this reorders
        Return (old);

String state()
        If (min)
                Return "Min-heap"
        Else
                Return "Max-heap"

<K,V> peekAt(n)
        If (n < 1 or n > size)
                Throw exception         //peeking out of range\
        E<K,V> [] tempHeap ← this.heap;                 //copy heap into tmp array
        For (int i = 0; i < size-1; i++)   //form ordered copy by sorting
                Int chosen = i;
                For (int j = i; j < size; j++)
                        //sort depends on mode of heap
                        If ( (min == true and tempHeap[j] < tempHeap[i] )or (min == false and
        tempHeap[j] > tempHeap[i]))
                                Chosen = j;
                E temp = tempHeap[i];
                teamHeap[i]= tempHeap[chosen]
                tempHeap[chosen] = temp;
        Return (teampHeap[n-1] )                 //return from sorted at index

void merge(APQ other)
        For (int i = 0; i < other.size; i ++)
                this.insert( other.heap[i].getKey(), other.heap[i].getValue() )
                //insert should reorder everytime
```

b) Big-O of toggle(), remove(e), peekAt(n), and merge(otherAPQ) methods

toggle(): O (nlog(n))
Worst case, outer loop runs until root of tree reached (~n/2 iterations). Within loop, downheap is called on each index (downheap is O(log(n)) due to tree structure). So, worst case is ~ (n/2) * log(n) operations and overall is O(n log(n))

remove(e): O(n)
Within function, there is only one loop that searches entire array for index (n steps). If such element is found, it is removed (constant time) and then heap order is re establish via up and downheaps (each being log(n) due to tree structure). Since up and downheaps are not in loop, they are better case than the loop used to find index of element to remove, so overall remove(e) is O(n)

peekAt(n): O(n^2)
Function utilizes selection sort (2 nested for loop) in order to make a temporary sorted heap from which we may return the n-1 index. Since aside from sort, function consist only of constant time operation such as swapping and index accessing of array, the 2 nested for loops in sort make this function O(n^2).

merge(other AOQ): O(n log(n))
Merge speaks of looping over array of other apq (n steps). Within this loop, each element is inserted, but insertion makes call to reorder via upheap (which is log(n) due to structure being of tree). As result, within each iteration, upheap will be run in log(n) time (worst case), so function is O(n log(n))

c)

```
APQ.java:268: error: cannot select a static class from a parameterized type
    APQ<Integer, String>.Thing<Integer, String> toRemove = apq.heap[5];
                        ^
APQ.java:273: error: cannot select a static class from a parameterized type
    APQ<Integer, String>.Thing<Integer, String> toChange = apq.heap[3];
                        ^
Note: APQ.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
2 errors

C:\Users\tonyy\2025Summer\CSI2110\Assignments>javac APQ.java
Note: APQ.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

C:\Users\tonyy\2025Summer\CSI2110\Assignments>java APQ
After 5 inserts (min heap): [(1, one), (5, five), (15, fifteen), (10, ten), (20, twenty)]
After inserting 20 more to trigger resize: size=25
peekAt(1) (min): (1, one)
peekAt(10) (min): (104, num104)
Removed top: (1, one)
New top after removeTop: (5, five)
Removing element from middle: (100, num100)
Heap after removal: [(5, five), (10, ten), (15, fifteen), (102, num102), (20, twenty), (106, num106), (101, num101), (110, num110), (103, num103), (104, num104), (105, num105), (118, num118), (107, num107), (108, num108), (109, num109)
, (119, num119), (111, num111), (112, num112), (113, num113), (114, num114), (115, num115), (116, num116), (117, num117)]
Changing key/value of element: (102, num102)
Heap after key/value change: [(0, zero), (5, five), (15, fifteen), (10, ten), (20, twenty), (106, num106), (101, num101), (110, num110), (103, num103), (104, num104), (105, num105), (118, num118), (107, num107), (108, num108), (109, nu
m109), (119, num119), (111, num111), (112, num112), (113, num113), (114, num114), (115, num115), (116, num116), (117, num117)]
Heap mode before toggle: Min
Heap mode after toggle (max heap): Max
Heap after toggle: [(119, num119), (117, num117), (118, num118), (113, num113), (116, num116), (107, num107), (109, num109), (111, num111), (112, num112), (115, num115), (105, num105), (106, num106), (15, fifteen), (108, num108), (101,
 num101), (110, num110), (10, ten), (5, five), (103, num103), (114, num114), (104, num104), (20, twenty), (0, zero)]
peekAt(1) (max): (119, num119)
peekAt(5) (max): (115, num115)
Other APQ before merge: [(3, three), (7, seven)]
Heap after merge: [(119, num119), (117, num117), (118, num118), (113, num113), (116, num116), (107, num107), (109, num109), (111, num111), (112, num112), (115, num115), (105, num105), (106, num106), (15, fifteen), (108, num108), (101,
num101), (110, num110), (10, ten), (5, five), (103, num103), (114, num114), (104, num104), (20, twenty), (0, zero), (3, three), (7, seven)]
Removing all elements:
Removed: (119, num119)
Removed: (118, num118)
Removed: (117, num117)
Removed: (116, num116)
Removed: (115, num115)
Removed: (114, num114)
Removed: (113, num113)
Removed: (112, num112)
Removed: (111, num111)
Removed: (110, num110)
Removed: (109, num109)
Removed: (108, num108)
Removed: (107, num107)
Removed: (106, num106)
Removed: (105, num105)
Removed: (104, num104)
Removed: (103, num103)
Removed: (101, num101)
Removed: (20, twenty)
Removed: (15, fifteen)
Removed: (10, ten)
Removed: (7, seven)
Removed: (5, five)
Removed: (3, three)
Removed: (0, zero)
Heap empty? true
Exception on peekAt(1) empty heap: Index out of bounds
After inserting 3 elements post-empty: [(60, sixty), (40, forty), (50, fifty)]
Heap mode after toggle back to min: Min
Heap after toggle back: [(40, forty), (60, sixty), (50, fifty)]

C:\Users\tonyy\2025Summer\CSI2110\Assignments>
```