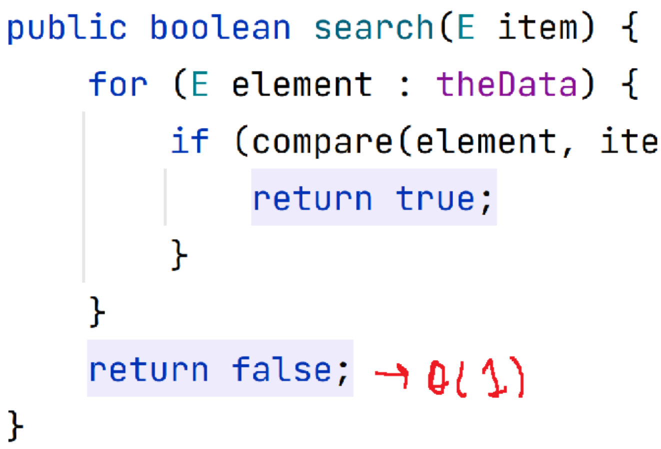
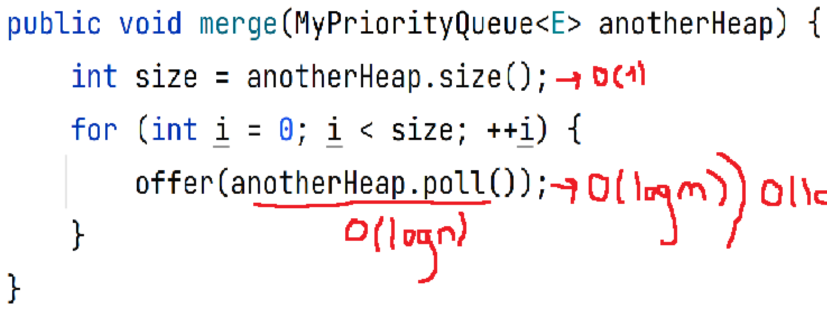
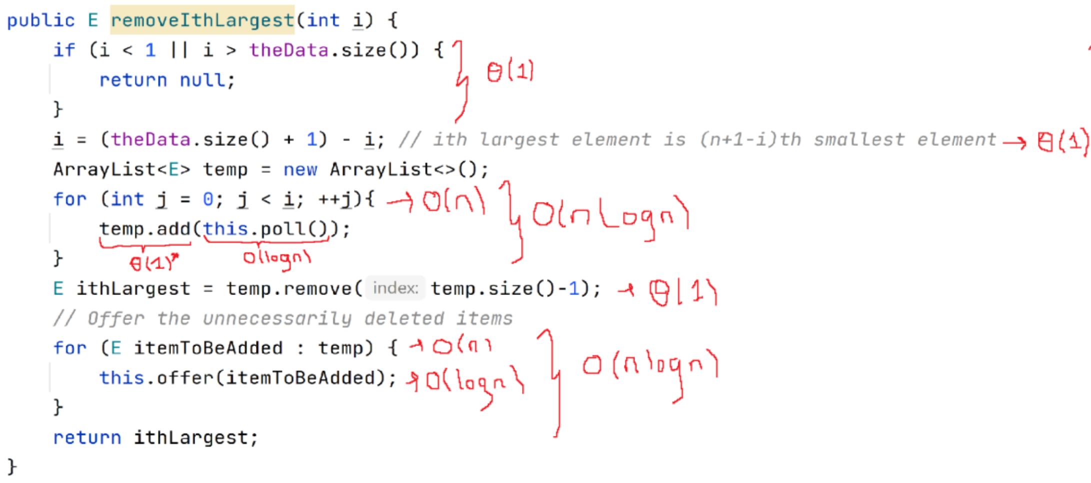


```
public boolean search(E item) {
    for (E element : theData) {
        if (compare(element, item) == 0) {   → θ(1) } θ(n)
            return true;
        }
    }
    return false;   → θ(1)
}
```



```
public void merge(MyPriorityQueue<E> anotherHeap) {
    int size = anotherHeap.size();   → θ(1)
    for (int i = 0; i < size; ++i) {
        offer(anotherHeap.poll());   → O(logm)) O(logmn) } O(nlogmn)
                  O(logn)
    }
}
```



```
public E removeIthLargest(int i) {
    if (i < 1 || i > theData.size()) {   } θ(1)
        return null;
    }
    i = (theData.size() + 1) - i; // ith largest element is (n+1-i)th smallest element   → θ(1)
    ArrayList<E> temp = new ArrayList<>();
    for (int j = 0; j < i; ++j){   → O(n) } O(nLogn)
        temp.add(this.poll());
          θ(1)     O(logn)
    }
    E ithLargest = temp.remove( index: temp.size()-1);   → θ(1)
    // Offer the unnecessarily deleted items
    for (E itemToBeAdded : temp) {   → O(n) } O(nlogn)
        this.offer(itemToBeAdded);   → O(logn)
    }
    return ithLargest;
}
```

} O(nlogn)