Size

Best Case: $\Omega(\mathbf{1})$

Average Case: $T(n) \in \Omega(1) \wedge T(n) \in O(1)) \rightarrow \boldsymbol{\Theta}(1)$

Worst Case: $O(1)$

```
public int size() {
    return size; O(1) Operation
}
```

Is Empty

Best Case: $\Omega(\mathbf{1})$

Average Case: $T(n) \in \Omega(1) \wedge T(n) \in O(1)) \rightarrow \boldsymbol{\Theta}(1)$

Worst Case: $O(1)$

```
boolean isEmpty()
{
    return size == 0; O(1) Operation
}
```

Insert

Best Case: $\Omega(\mathbf{1})$ when no restructuring needs to occur/heap is empty

Average Case: Since $\Omega(1) \supset \Omega(\log_2 n)$, $(T(n) \in \Omega(\log_2 n) \wedge T(n) \in O(\log_2 n)) \rightarrow \boldsymbol{\Theta}(\log_2 n)$

Worst Case: $O(n), O(\log_2 n)$ *amoritized*

```
public void insert(K key, V value)
{
    if (size == capacity)
    {
        dynamicallyResize(); O(n) Operation
    }
    priorityHeap[size++] = new EntryNode<K,V>(key, value); O(1) Operation
    upheap(size - 1); O(log2 n) Operation
}
```

Max

Best Case: $\Omega(\mathbf{1})$

Average Case: $T(n) \in \Omega(1) \wedge T(n) \in O(1)) \rightarrow \boldsymbol{\Theta}(1)$

Worst Case: $O(1)$

```
public Entry<K,V> max()
{
    return priorityHeap[0]; O(1) Operation
}
```

Remove Max

Best Case: $\Omega(\mathbf{1})$ when the root is the only element/the max heap property is already satisfied

Average Case: Since $\Omega(1) \supset \Omega(\log_2 n)$, $(T(n) \in \Omega(\log_2 n) \wedge T(n) \in O(\log_2 n)) \rightarrow \boldsymbol{\Theta}(\log_2 n)$

Worst Case: $O(\log_2 n)$

```
public Entry<K,V> removeMax()
{
    Entry<K,V> val = max(); O(1) Operation
    swap(0, size - 1); O(1) Operation
    priorityHeap[size - 1] = null; O(1) Operation
    size--; O(1) Operation
    downheap(0); O(log2 n) Operation
    return val; O(1) Operation
}
```

Update

Best Case: $\Omega(\mathbf{1})$ when the root is the only element

Average Case: Since $\Omega(1) \supset \Omega(\log_2 n)$, $(T(n) \in \Omega(\log_2 n) \wedge T(n) \in O(\log_2 n)) \rightarrow \boldsymbol{\Theta}(\log_2 n)$

Worst Case: $O(\log_2 n)$ amortized

```
public Entry<K, V> update(K oldKey, K newKey) throws IllegalArgumentException {
       Entry<K,V> entry = priorityHeap.max(); O(1) Operation
       priorityHeap.removeMax(); O(log2 n) Operation
       priorityHeap.insert(newKey, entry.getValue()); O(log2 n) Operation
       return entry; O(1) Operation
    }
```