SPQ ADT Initialization:

1. Initialize `heap` as an empty array with a default size.
2. Set `heapState` to Min (default heap state).
3. Set `size` to 0.
4. Define `max_size` as the initial capacity of the array.

Insert (key, value):

1. If the `size` of the heap equals `max_size`, call `extendArray` to increase capacity.
2. Add the (key, value) pair to the end of the heap.
3. Increment the `size` by 1.
4. Call `upheap` with the index of the newly inserted element to restore the heap property.

RemoveTop:

1. If the heap is empty, raise an exception.
2. Save the top element of the heap (element at index 0).
3. Replace the top element with the last element in the heap.
4. Decrease the `size` by 1.
5. Call `downheap` with index 0 to restore the heap property.
6. Return the saved top element.

Top:

1. If the heap is empty, raise an exception.
2. Return the element at index 0 of the heap.

Toggle:

1. Switch `heapState` between Min and Max.
2. Rebuild the heap using `heapify` to ensure it satisfies the property of the new state.

ReplaceValue (entry, newValue):

1. Search for the entry in the heap.
2. If the entry is not found, raise an exception.
3. Update the value of the entry to `newValue`.

4. Determine if the heap property is violated:
   - If the entry is smaller (or larger) than its parent in a Min-Heap (or Max-Heap), call `upheap`.
   - Otherwise, call `downheap`.
5. Return the old value of the entry.

Heapify:

1. For each non-leaf node starting from the last parent node to the root:
   - Call `downheap` on the current node to restore the heap property.

Upheap (index):

1. While the element at `index` has a parent:
   a. If the element violates the heap property with its parent, swap it with the parent.
   b. Update `index` to the parent index.
2. Stop when the heap property is restored or the element becomes the root.

Downheap (index):

1. While the element at `index` has at least one child:
   a. Determine the smaller (or larger) child based on the current `heapState`.
   b. If the element violates the heap property with the selected child, swap them.
   c. Update `index` to the index of the swapped child.
2. Stop when the heap property is restored or the element becomes a leaf.

ExtendArray:

1. Create a new array with larger capacity (e.g., double the current `max_size`).
2. Copy all elements from the current heap to the new array.
3. Replace the current heap with the new array.

LeftChildIndex (index):

1. Return the index of the left child (2 * index + 1).
2. If the index is out of bounds, return -1.

RightChildIndex (index):

1. Return the index of the right child (2 * index + 2).

2. If the index is out of bounds, return -1.

ParentIndex (index):

1. Return the index of the parent ((index - 1) / 2).
2. If the index is invalid (e.g., root node), return -1.

HasLeftChild (index):

1. Check if the left child index is within bounds.

HasRightChild (index):

1. Check if the right child index is within bounds.

Compare (index1, index2):

1. Compare the values at `index1` and `index2` based on `heapState`:
   - Min-Heap: Return true if the value at `index1` is smaller than at `index2`.
   - Max-Heap: Return true if the value at `index1` is larger than at `index2`.