Formal Specification for Binary Heap

Allowed Operations

1. `insert(element)`

- **Description**: Adds a new element to the heap while maintaining the heap property.
- **Pre-conditions**:
  HeapHeap is a valid binary heap before insertion.
- **Post-conditions**:
  After insertion, the heap remains a complete binary tree, and the heap property is preserved.
- **Axiom**:
  If x is the newly inserted element:
  - For Max-Heap:$\forall\, y \in Heap,\ Parent(x) \geq x$

2. `extract()`

- **Description:** Removes and returns the root element(the maximum of the heap)
- **Pre-condition:** Heap is not empty.
- **Post-conditions:**
  - The root element was removed.
  - Adjust the binary heap so that it satisfies that any parent node is greater than or equal to its left and right child nodes
  - Return the heap property
- **Axiom:**

  $$extract(q) => size(q) - 1$$

3. `peak()`

- **Description:** Returns the root element without removing it.
- **Pre-conditions**:
  Heap is not empty.
- **Post-conditions**:
  The heap remains unchanged.

  Axiom:

  $peek(q) = max(Heap)$

4. `is_empty()`

- **Description:** Checks whether the heap is empty
- **Pre-conditions**:

  No
- **Post-condition:**

  returns TRUE if heap is empty,FALSE otherwise
- **Axiom:**

  $is\_empty(q) = (\text{size}(\text{Heap}) = 0)$

5. `heapify(index)`

- **Description:** Ensures that the subtree rooted at the given index satisfies the heap property.
- **Pre-conditions:**

  Heap is not empty
- **Post-condition:**

  Returns the propre heap
- **Axioms:**

  $Parent(x) \geq x$

Correction Test

1.Insert Test

insert element one by one into an initially empty heap. After each insertion ,verify the property of the heap for all nodes.

- Insert a single element into an empty heap.
- Insert multiple elements to ensure that the heap always meets the heap attributes.
- Insert repeated elements to verify the behaviour of the heap.
- Insert extreme values (such as minimum and maximum values) and check the boundary processing.

2.Extract Test

Extract the root repeatedly until the heap is empty. Check that the extracted elements are in decreasing order.

- Extract from a heap containing a single element.
- Extract from the heap of multiple elements until the heap is empty.
- When extracting, check whether the returned element is the current maximum value.
- Test the extraction behaviour in the case of duplicate elements.

3.Heapify Test

Create an invalid heap and apply `heapify()` .Verify that the heap property is restored.

- Call heapify on a randomly arranged array to ensure that the heap attributes are restored.

- Call heapify on a partially unordered heap to verify the result.

4.Peek Test

Call `peek` and ensure the returned value matches the maximum (or minimum) element without modifying the heap.

- Calling peek on the empty heap should throw an error or return an empty value.
- Call peek on the non-empty heap to verify whether the return value is the current maximum (or minimum value).
- Repeatedly call peek to ensure that the heap has not changed.

5.Edge Cases

- Calling `insert` ,`extract` or `peek` on the empty heap should throw an error or return an empty value.
- Insert the maximum value to ensure the correctness.
- Perform insertion, extraction and heap operations on large-scale data sets (such as million-level data) to test performance.