Homework 8 Answers

Question 1: Insert Values into MaxHeap in Two Orders

Code

```
MaxHeap<Integer> heap1 = new MaxHeap<>(50);
heap1.insert(30);
heap1.insert(20);
heap1.insert(80);
heap1.insert(40);
heap1.insert(50);
System.out.println("Heap 1: " + heap1.stringHeap());

MaxHeap<Integer> heap2 = new MaxHeap<>(50);
heap2.insert(80);
heap2.insert(50);
heap2.insert(40);
heap2.insert(30);
heap2.insert(20);
System.out.println("Heap 2: " + heap2.stringHeap());
```

Result

Heap 1 and Heap 2 generated different heaps due to the order of insertion.

Question 2: `deleteMax` Three Times

Code

```
MaxHeap<Integer> heap3 = new MaxHeap<>(50);
heap3.insert(10);
heap3.insert(40);
heap3.insert(50);
heap3.insert(5);
heap3.insert(45);
heap3.insert(30);
heap3.insert(20);
System.out.println("Initial Heap: " + heap3.stringHeap());
```

```
heap3.removeMax();
System.out.println("After 1st deleteMax: " + heap3.stringHeap());
heap3.removeMax();
System.out.println("After 2nd deleteMax: " + heap3.stringHeap());
heap3.removeMax();
System.out.println("After 3rd deleteMax: " + heap3.stringHeap());
```

Result

The heap and array representation after each `deleteMax` operation shows the heap property maintained.

Question 3: Indices for Leaves in Heap of 13 Nodes

Code

```
int[] leaves = getLeafIndices(13);
System.out.print("Leaf Indices: ");
for (int leaf : leaves) {
    System.out.print(leaf + " ");
}
System.out.println();
```

Result

Leaf indices: `6, 7, 8, 9, 10, 11, 12`.

Question 4: Parent and Child Indices for Index 22

Code

```
System.out.println("Parent Index: " + MaxHeap.parent(22));
System.out.println("Left Child Index: " + MaxHeap.leftChild(22));
System.out.println("Right Child Index: " + MaxHeap.rightChild(22));
```

Result

- Parent Index: `10`

- Left Child Index: `45`
- Right Child Index: `46`

Question 5: Check if Array Represents a MaxHeap

Code

```
int[] array = {100, 50, 80, 30, 20, 60, 70};
System.out.println("Is MaxHeap: " + isMaxHeap(array));
```

Method

```
public static boolean isMaxHeap(int[] array) {
    for (int i = 0; i < array.length / 2; i++) {
        int left = 2 * i + 1;
        int right = 2 * i + 2;
        if ((left < array.length && array[i] < array[left]) ||
            (right < array.length && array[i] < array[right])) {
            return false;
        }
    }
    return true;
}
```

Result

The array is a valid MaxHeap.

Question 6: BuildHeap Demonstration

Code

```
Integer[] buildArray = {100, 50, 7, 80, 5, 20, 30, 10, 70, 60};
MaxHeap<Integer> buildHeap = new MaxHeap<>(buildArray, buildArray.length, 50);
System.out.println("Heap after BuildHeap: " + buildHeap.stringHeap());
```

Result

The heap built demonstrates proper sift-down adjustments.

Question 7: MergeSort Execution

Steps

1. Split the array into halves recursively.
2. Merge sorted halves maintaining order.

Code

```
// Simulated steps of MergeSort
Array: [30, 50, 45, 40, 20, 10, 60, 35]
// Splitting:
[30, 50, 45, 40] and [20, 10, 60, 35]
[30, 50] [45, 40] [20, 10] [60, 35]
[30] [50] [45] [40] [20] [10] [60] [35]
// Merging:
[30, 50] [40, 45] [10, 20] [35, 60]
[30, 40, 45, 50] [10, 20, 35, 60]
[10, 20, 30, 35, 40, 45, 50, 60]
```

Question 8: Make MergeSort Unstable

Modify the merge step to prioritize the right array instead of the left when encountering equal keys.

Question 9: QuickSort Execution

Simulated steps for array `[30, 50, 45, 40, 20, 10, 60, 35]`:

1. Select pivot and partition.
2. Recursively sort partitions.

Question 10: Partition Instability Demonstration

Partitioning `[10, 20, 40a, 40b, 30]` shows original order of `40a` and `40b` is not preserved.

Question 11: Sorting Algorithm Properties

In-place and Stable

- **Insertion Sort**: In-place, Stable
- **Selection Sort**: In-place, Not Stable
- **Bubble Sort**: In-place, Stable
- **BST Sort**: Not In-place, Not Stable
- **Heap Sort**: In-place, Not Stable
- **MergeSort**: Not In-place, Stable
- **QuickSort**: In-place, Not Stable