SOLUTION OF TASK:-

```
import java.util.*;


public class PowerOfTwoMaxHeap {
  private int x;
  private List<Integer> data;


  public PowerOfTwoMaxHeap(int x) {
    if (x <= 0) {
      throw new IllegalArgumentException("x must be greater than 0");
    }
    this.x = x;
    this.data = new ArrayList<Integer>();
  }


  public void insert(int value) {
    data.add(value);
    int index = data.size() - 1;
    int parentIndex = (index - 1) / 2;
    while (parentIndex >= 0 && data.get(parentIndex) < value) {
      Collections.swap(data, parentIndex, index);
      index = parentIndex;
    parentIndex = (index - 1) / 2;
  }
}

public int popMax() {
  if (data.size() == 0) {
    throw new NoSuchElementException("heap is empty");
  }
  int max = data.get(0);
  Collections.swap(data, 0, data.size() - 1);
  data.remove(data.size() - 1);

  int index = 0;
  while (index < data.size()) {
    int leftChildIndex = 2 * index + 1;
    int rightChildIndex = 2 * index + 2;
    int leftChild = Integer.MIN_VALUE;
    int rightChild = Integer.MIN_VALUE;
    if (leftChildIndex < data.size()) {
      leftChild = data.get(leftChildIndex);
    }
    if (rightChildIndex < data.size()) {
      rightChild = data.get(rightChildIndex);
    }
```

```
      if (Math.max(leftChild, rightChild) <= data.get(index)) {
        break;
      } else if (leftChild >= rightChild) {
        Collections.swap(data, index, leftChildIndex);
        index = leftChildIndex;
      } else {
        Collections.swap(data, index, rightChildIndex);
        index = rightChildIndex;
      }
    }

    return max;
  }

}
```

Explanation:

```
public class PowerOfTwoMaxHeap {
    private int[] heap;
    private int maxSize;
    private int size;
    private int exponent;

    public PowerOfTwoMaxHeap(int exponent) {
```

```
    this.exponent = exponent;
    this.maxSize = (int) Math.pow(2, exponent);
    this.heap = new int[maxSize];
    this.size = 0;
}

public void insert(int value) {
    if (size == maxSize) {
        throw new HeapFullException();
    }

    heap[size] = value;
    size++;

    int current = size - 1;

    while (current > 0 && heap[current] > heap[getParent(current)]) {
        swap(current, getParent(current));
        current = getParent(current);
    }
}

public int popMax() {
    if (size == 0) {
        throw new HeapEmptyException();
    }

    int max = heap[0];
    heap[0] = heap[size - 1];
    size--;
    maxHeapify(0);
    return max;
}

private void maxHeapify(int index) {
    int left = getLeftChild(index);
    int right = getRightChild(index);
    int largest = index;

    if (left < size && heap[left] > heap[index]) {
        largest = left;
    }

    if (right < size && heap[right] > heap[largest]) {
        largest = right;
    }

    if (largest != index) {
}

        swap(index, largest);
        maxHeapify(largest);
    }
}

private int getParent(int index) {
    return (index - 1) / 2;
}

private int getLeftChild(int index) {
    return 2 * index + 1;
}

private int getRightChild(int index) {
    return 2 * index + 2;
}

private void swap(int a, int b) {
    int temp = heap[a];
    heap[a] = heap[b];
    heap[b] = temp;
}

private class HeapFullException extends RuntimeException {}
```

```
    private class HeapEmptyException extends RuntimeException {}
}
```

```
public class PowerOfTwoMaxHeap {
    private int x;
    private List<Integer> heap;

    public PowerOfTwoMaxHeap(int x) {
        this.x = x;
        this.heap = new ArrayList<>();
    }

    public void insert(int val) {
        heap.add(val);
        int curr = heap.size() - 1;
        int parent = (curr - 1) / 2;
    while (parent >= 0 && heap.get(parent) < heap.get(curr)) {
        Collections.swap(heap, parent, curr);
        curr = parent;
        parent = (curr - 1) / 2;
    }
}

public int popMax() {
    if (heap.size() == 0) {
        throw new NoSuchElementException();
    }
    int max = heap.get(0);
    Collections.swap(heap, 0, heap.size() - 1);
    heap.remove(heap.size() - 1);
    int curr = 0;
    while (curr < heap.size()) {
        int leftChild = 2 * curr + 1;
        int rightChild = 2 * curr + 2;
        if (leftChild >= heap.size() && rightChild >= heap.size()) {
            break;
        }
        if (rightChild >= heap.size()) {
            if (heap.get(curr) < heap.get(leftChild)) {
                Collections.swap(heap, curr, leftChild);
                }

                break;

            }

            int maxChild = heap.get(leftChild) > heap.get(rightChild) ? leftChild :
rightChild;

            if (heap.get(curr) < heap.get(maxChild)) {

                Collections.swap(heap, curr, maxChild);

                curr = maxChild;

            } else {

                break;

            }

        }

        return max;

    }

}
```

Time complexity: O(log n) for insert and popMax

Space complexity: O(n)

Note: if you need to support arbitrary number of children (i.e. not just 2^x) then you can use an array to represent the heap and the heap property will be that for any node at index i, its children are at indices 2i+1 and 2i+2, and its parent is at index (i-1)/2.