Simulation based practice problems on Heap Data Structure

Heap Operations Simulation

You are given an initially empty min-heap. Perform the following operations on the heap and show the state of the heap after each operation. Draw the heap as a binary tree after each step.

1. Insert the following elements in this order: 15, 10, 20, 8, 12, 16
2. Delete the minimum element
3. Insert 5
4. Delete the minimum element
5. Insert 18

For each operation:
a) Show the state of the heap as a binary tree
b) Explain the steps taken during the insert or delete operation
c) List the elements in the heap in array representation

Remember:
- In a min-heap, the parent node is always smaller than its children
- When inserting, add the new element at the next available position, then "bubble up" if necessary
- When deleting the minimum, replace the root with the last element, then "bubble down" if necessary

Heap Array Representation Simulation with Trace Table

In this exercise, you'll work with a max-heap represented as an array. Remember, for “0” based indexing, an element at index i:

- Its left child is at index 2i + 1
- Its right child is at index 2i + 2
- Its parent is at index (i - 1) // 2 (integer division)

Starting with an empty heap, perform the following operations and show the array representation after each step:

1. Insert the following elements in order: 42, 29, 18, 35, 26, 10, 44, 31, 15
2. Delete the maximum element (root)
3. Insert 50
4. Delete the maximum element (root)
5. Insert 12

For each operation: a) Update the trace table showing the complete array representation of the heap b) Explain the steps taken during the insert or delete operation c) Draw the corresponding binary tree representation (optional, for visualization)

Note:

- The array representation should show all elements, including empty spots if any
- For insertion, add the new element at the end of the array, then "bubble up" if necessary
- For deletion, replace the root with the last element, remove the last element, then "bubble down" if necessary

Use the following trace table on the next page to record the state of the heap after each operation:

Operation	**Index 0**	**Index 1**	**Index 2**	**Index 3**	**Index 4**	**Index 5**	**Index 6**	**Index 7**	**Index 8**
Initial	-	-	-	-	-	-	-	-	-
Insert 42									
Insert 29									
Insert 18									
Insert 35									
Insert 26									
Insert 10									
Insert 44									
Insert 31									
Insert 15									
Delete max									
Insert 50									
Delete max									
Insert 12									

Fill in the trace table as you perform each operation, showing how the array representation of the heap changes.

Example of a filled row:

Insert 42	42	-	-	-	-	-	-	-	-

This represents a heap with just the root element 42 inserted.

Transforming a Heap Array into a Sorted Array

In this exercise, you'll explore how a max-heap represented as an array can be transformed into a sorted array (in ascending order) using the heap properties. This process is the core of the heap sort algorithm.

Given the following max-heap represented as an array:

[90, 85, 75, 60, 70, 40, 65, 30, 50]

Perform the following steps:

1. Extract the maximum element (root) and place it at the end of the array.
2. Reduce the heap size by 1.
3. Heapify the root element to maintain the max-heap property.
4. Repeat steps 1-3 until the heap is empty.

Use the trace table below to show the state of the array after each extraction and heapify operation. The first row is filled out for you as an example.

Operation	Index 0	Index 1	Index 2	Index 3	Index 4	Index 5	Index 6	Index 7	Index 8	Extracted
Initial	90	85	75	60	70	40	65	30	50	-
1st Extract										
2nd Extract										
3rd Extract										
4th Extract										
5th Extract										
6th Extract										
7th Extract										
8th Extract										
Final										

For each extraction: a) Fill in the row in the trace table. b) Explain the steps taken to maintain the heap property after the extraction.

Questions:

1. How does the final state of the array compare to the initial state in terms of ordering?
2. Why does this process result in a sorted array?
3. How would the process and result differ if we started with a min-heap instead of a max-heap?