NORTHERN ARIZONA UNIVERSITY
School of Informatics, Computing, and Cyber Systems

**CS 249 Data Structures**
**L15 Worksheet**
**(Heaps)**

# Team

Record below the names of students who worked together on this worksheet.

| **Student name** | **Student ID** |
|---|---|
| Noelia Canela | 6179389 |
| Ethan Ferguson | 6172256 |
| | |

# General Instructions

**Step 1.** Answer the questions below to strengthen your understanding of heap related concepts.

**Step 4.** Generate a PDF with your solution and submit it as the solution for this assignment.

---

1. Consider the following binary tree. Is the tree a heap? If so, define whether it is a max or min heap. If not, identify the first violation of the min or max-heap property.

```
[10, 15, 30, 40, 50, 100, 40, 80, 70, 60]
```

```
- min-heap
```

2. Insert the following elements one by one into an initially empty max-heap. Show the heap after each insertion. Elements: `20, 15, 30, 40, 10, 8, 25`

```
- 20 = heap: 20
- 15 = heap: [20, 15] -> [20, 15]
- 30 = heap: [20, 15, 30] -> [30, 15, 20]
- 40 = heap: [30, 15, 20, 40] -> [40, 30, 20, 15]
- 10 = heap: [40, 30, 20, 15, 10] -> [40, 30, 20, 15, 10, 8]
- 8 = heap: [40, 30, 20, 15, 10, 8] -> [40, 30, 20, 15, 10, 8]
- 25 = heap: [40, 30, 20, 15, 10, 8, 25] -> [40, 30, 25, 15, 10, 8, 20]
= [40, 30, 25, 15, 10, 8, 20]
```

3. Insert the following elements into an initially empty min-heap. Show the array representation of the heap after all insertions. Elements: `35, 12, 10, 40, 32, 30, 50`

```
- 35 = heap: [35]
- 12 = heap: [35, 12] -> [12, 35]
- 10 = heap: [12, 35, 10] -> [10, 35, 12]
```

- 40 = heap: [10, 35, 12, 40] -> [10, 35, 12, 40]
- 32 = heap: [10, 35, 12, 40, 32] -> [10, 32, 12, 40, 35]
- 30 = heap: [10, 32, 12, 40, 35, 30] -> [10, 32, 12, 40, 35, 30]
- 50 = heap: [10, 32, 12, 40, 35, 30, 50] -> [10, 32, 12, 40, 35, 30, 50]
= [10, 32, 12, 40, 35, 30, 50]

4. Given the unsorted array [20, 30, 15, 10, 40, 50, 45], turn it into a max-heap using the downheapify method. Show each step of the process.

- [20, 30, 50, 10, 40, 15, 45]
- [20, 40, 50, 10, 30, 15, 45]
- [50, 40, 20, 10, 30, 15, 45]
- [50, 40, 45, 10, 30, 15, 20]

5. Starting with the following max-heap array [100, 90, 70, 60, 50, 40, 20, 10], extract the maximum element and then re-heapify the remaining elements. Show the heap tree representation and the array before and after heapifying.

- [90, 10, 70, 60, 50, 40, 20]
- [90, 60, 70, 10, 50, 40, 20]
Final max heap array after re heapifying
= [90, 60, 70, 10, 50, 40, 20] Final
max heap array
= [90, 60, 70, 10, 50, 40, 20]

6. Sort the following array [30, 20, 10, 40, 50, 70, 60] using heap sort. Build a min-heap with the elements from the array, then repeatedly remove the root adding it back to the array. Show the minheap and the sorted array for each step as elements are removed and added back to the sorted array.

- [30, 20, 10, 40, 50, 70, 60]
Remains the same
Heapify at value 20
= remains the same
Heapify at value 30
= [10, 20, 30, 40, 50, 70, 60]
= [10, 20, 30, 40, 50, 60, 70]

7. Given a min-heap array [5, 10, 15, 20, 25, 30, 35], write out the steps needed to find the 3rd smallest element.

- smallest element: [10, 20, 15, 35, 25, 30]
- 2nd smallest element: [15, 20, 30, 35, 25]
- 3rd smallest element: new root (15_ is the 3rd smallest

8. For the array representation of a max-heap `[90, 80, 70, 50, 40, 30, 20, 10]`, identify the element at index 3 and its parent, left child, and right child using index calculations.

```
- element at index 3: value 50
- parent: 80 (index 1)
- left child: 10 (index 7)
- right child: none because its out of bounds
```

9. Given the two min-heaps `[10, 25, 20]` and `[30, 5, 15]`, merge them into a single min-heap by intercalating the elements from both arrays and using down-heapify to maintain the heap property after merging.
   To down-heapify, you start at the root of a subtree and compare the root with their children. If a heapproperty violation is identified, the children is swapped with the root. Since leaf nodes do not have children, they are considered down-heapified and can be ignored.
   Therefore, start at the first non-leaf node at index `n / 2 - 1` and heapify each subtree from that index to zero, working up in the root. For example, since the merged array is size 6, the first non-leaf node is at index `6 / 2 - 1 - 2`. Therefore, we need to down-heapify indexes 2, 1, and 0.
   Show the merged array, the tree representation after each heapify iteration (i.e., before heapifying and after heapifying each relevant subtree), and the resulting min-heap array representation.

```
- [10, 25, 20, 30, 5, 15]
Heapify at 2: [10, 25, 15, 30, 5, 20]
At index 1: [10, 5, 15, 30, 25, 20]
At index 0: [5, 10, 15, 30, 25, 20]
Final heap: [5, 10, 15, 30, 25, 20]
```

10. Starting with the min-heap `[5, 10, 15, 20, 25, 30, 35]`, convert it into a max-heap. To achieve that, remove the root (which is the smallest element in the current array), add it as the last element in the max-heap array, and re-heapify the min-heap so that the minimum element is again at the root. Repeat this process until all the elements are added to the max-heap. Show the initial min-heap, the min-heap tree representation after each removal (before and after heapifying), and the sorted array after each insertion.

```
- [10, 25, 15, 20, 25, 30, 35, 5]
Heapify → [15, 25, 20, 10, 30, 35, 25, 5]

- [20, 30, 25, 5, 25, 35, 15]
Heapify → [25, 30, 20, 5, 35, 15, 25]

- Remove 15, add it to the max-heap → [20, 30, 25, 5, 35, 25]
Heapify → [30, 25, 25, 5, 35, 20]

- Remove 20, add it to the max-heap → [25, 35, 30, 5, 25]
Heapify → [35, 25, 30, 5, 25]

- Remove 25, add it to the max-heap → [30, 35, 5, 25]
Heapify → [35, 30, 25, 5]
```

- Remove 30, add it to the max-heap → [35, 25, 5]
Heapify → [35, 25, 5]

- Remove 35, add it to the max-heap → [25, 5]
Heapify → [25, 5]

- Remove 25, add it to the max-heap → [5]

= Array: [35, 30, 25, 20, 15, 10, 5]