Task04:

* **Priority Queues** (its primary use case)
* Operating System Process Scheduling
* Graph Algorithms (Dijkstra's, Prim's)
* Event-driven Simulations
* Load Balancing / Task Management
* Huffman Coding
* **In-place Sorting for Memory-Constrained Environments**
* Embedded Systems
* External Sorting (as part of hybrid approaches)
* **Guaranteed Performance** (predictable O(nlogn) time)
* Real-time Systems
* Mission-Critical Applications
* **Selection Problems** (finding k-th largest/smallest element or top K elements)
* **Operating System Utilities** (e.g., in memory management, garbage collection)

Task06:

* **Treat Array as Binary Tree:** Interpret the input array as a complete binary tree.
* **Build Max-Heap:** Convert the entire array into a Max-Heap. This is done by starting from the last non-leaf node and moving towards the root, applying a heapify operation on each node to ensure every parent is greater than its children.
* **Largest Element at Root:** After building the Max-Heap, the largest element in the entire array will always be at the root (index 0).
* **Swap and Shrink:** Swap this largest element (at index 0) with the last element of the unsorted portion of the array. Effectively, the swapped largest element is now in its correct sorted position at the end.
* **Re-heapify and Repeat:** Decrease the effective size of the heap by one (excluding the now sorted element). Then, perform a heapify operation on the new root (index 0) to restore the Max-Heap property for the remaining unsorted elements. Repeat steps 4 and 5 until the heap size is 1.