Heap-based Priority Queue (OpenMP)

Presented by :-

Divyansh Maheshwari

Eish Malvi

Somya Mehta

Tanishq Jain

CS-309 Parallel Computing

Prof. CHANDRESH KUMAR SIR

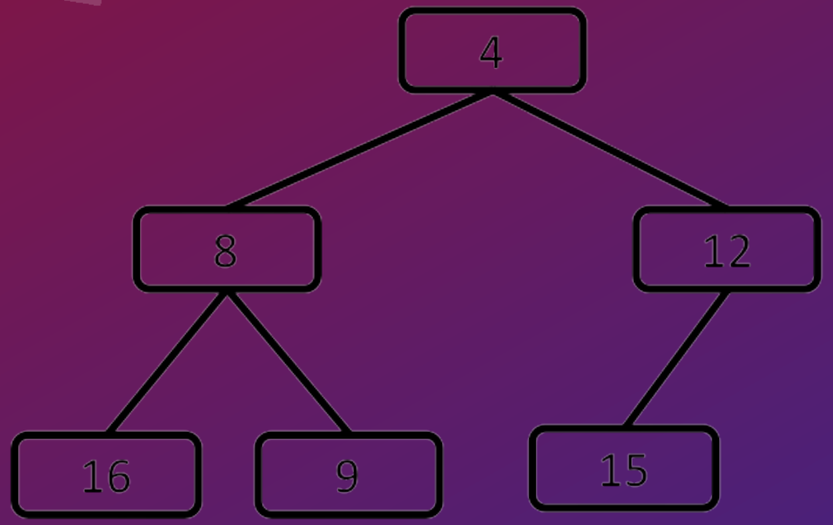
Mr. PANKAJ CHAUDHARY SIR

Priority Queues

- Priority queues are fundamental data structures.
- Insert and DeleteMin Operations.
- Graph problem: Dijkstra's algorithm
- Branch-and-Bound problem: 0/1 Knapsack

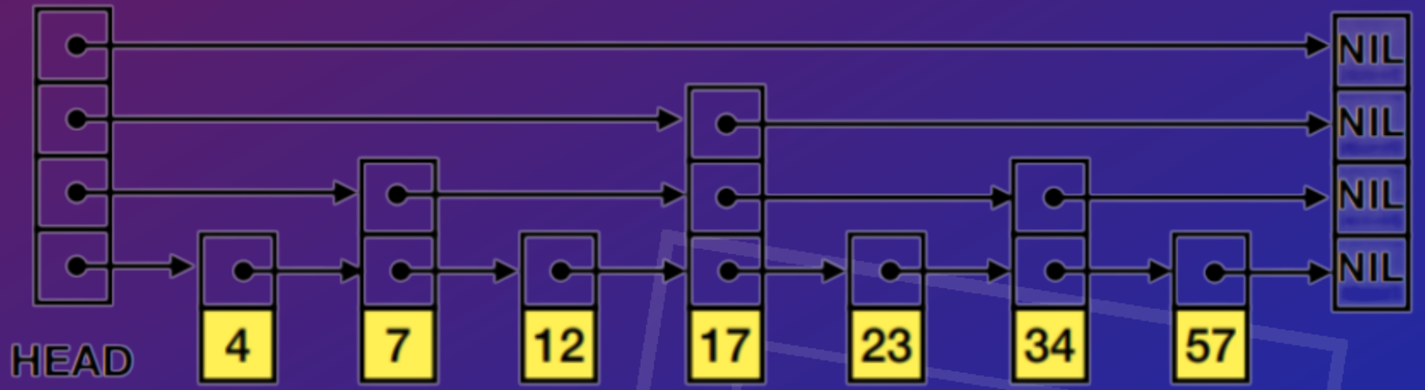
Concurrent Priority Queue Design Choices

Heap Based Priority Queue



- Computation Complexity: O(logN)
- Space Complexity: N + O(1)
- Limitation: Scalability

Skip-List Based Priority Queue



- Computation Complexity: O(logN)
- Space Complexity: O(N) + O(1)
- Dynamic memory management.

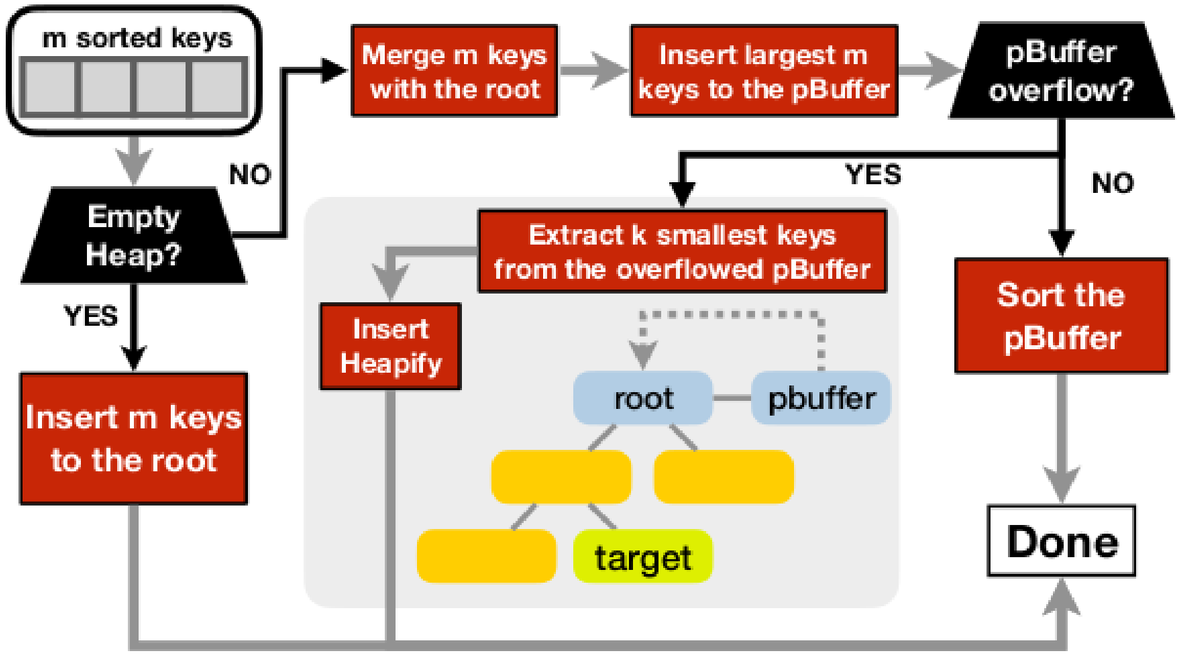
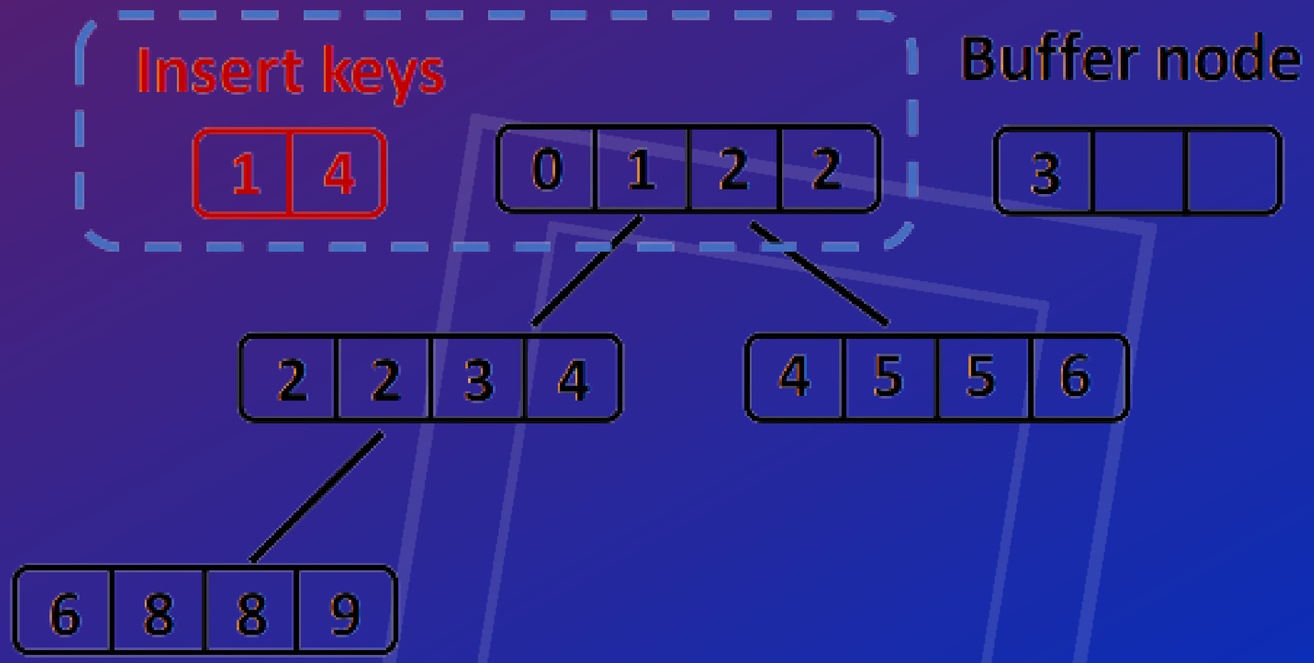


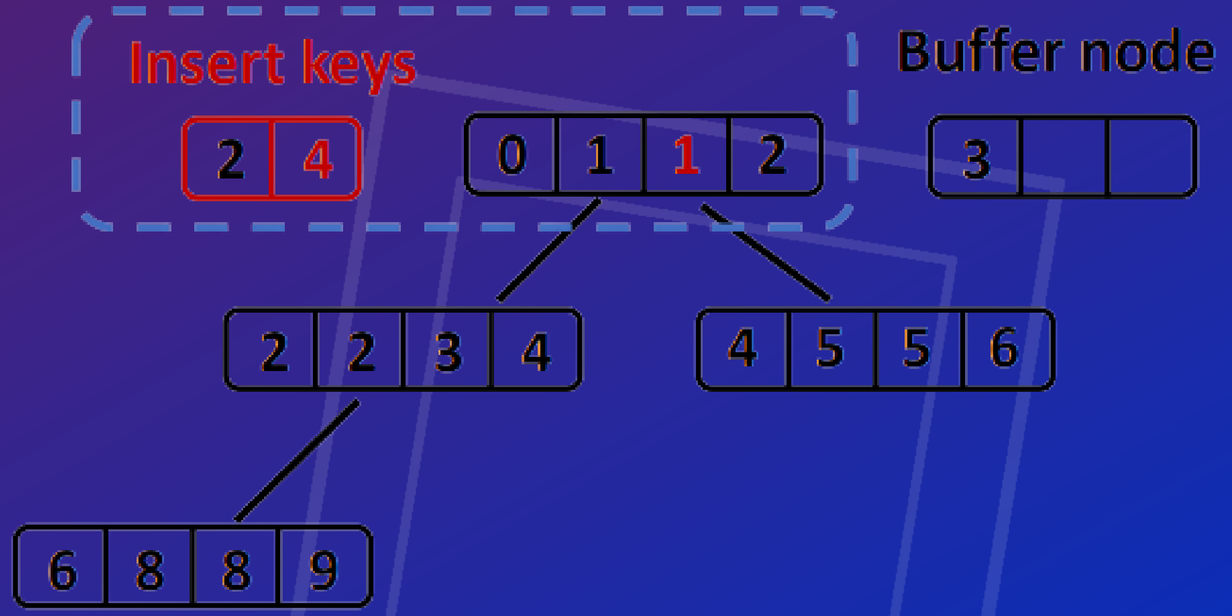
Figure 3: BGPQ Insert Operations

BGPQ INSERT OPERATION

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.

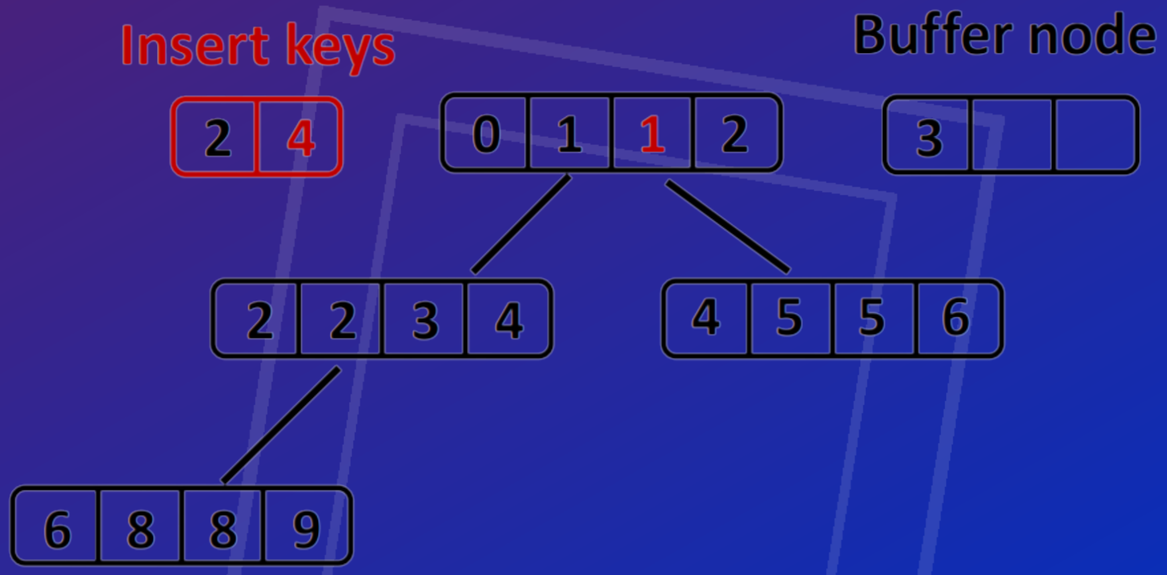


- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.



- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

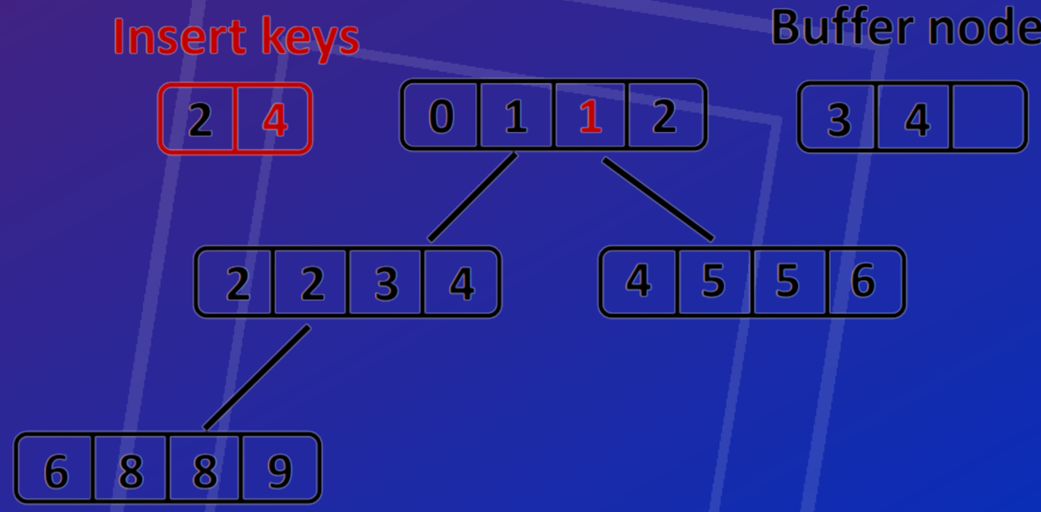
- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.



- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.

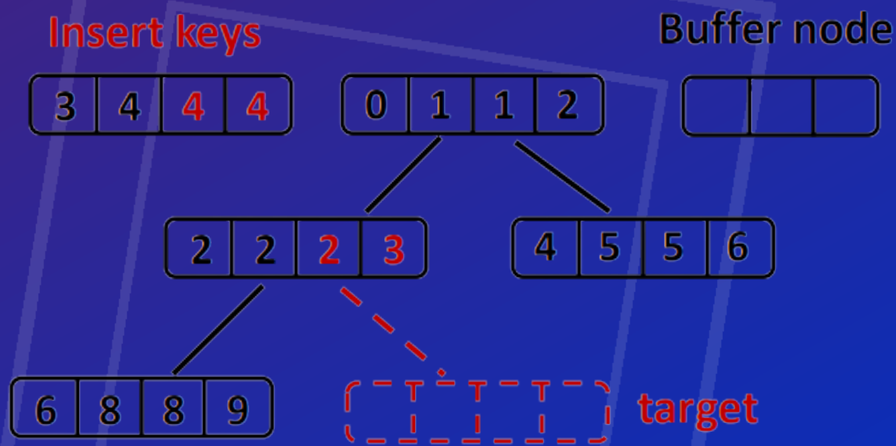
- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.



- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.

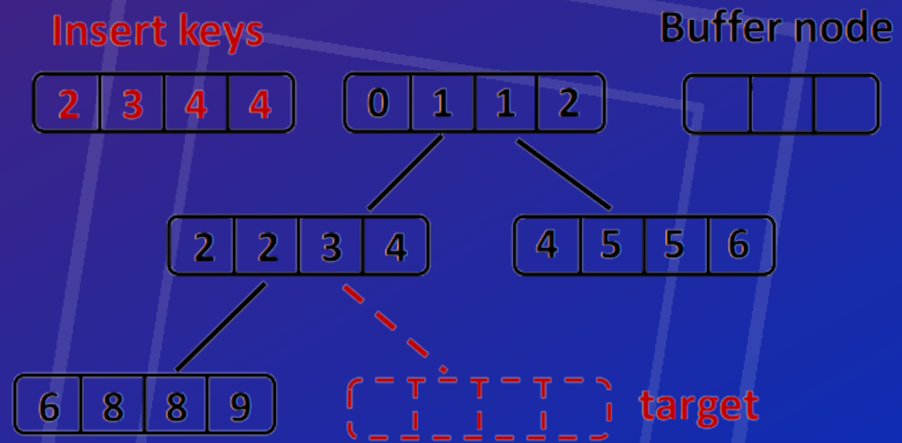
- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.



- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.

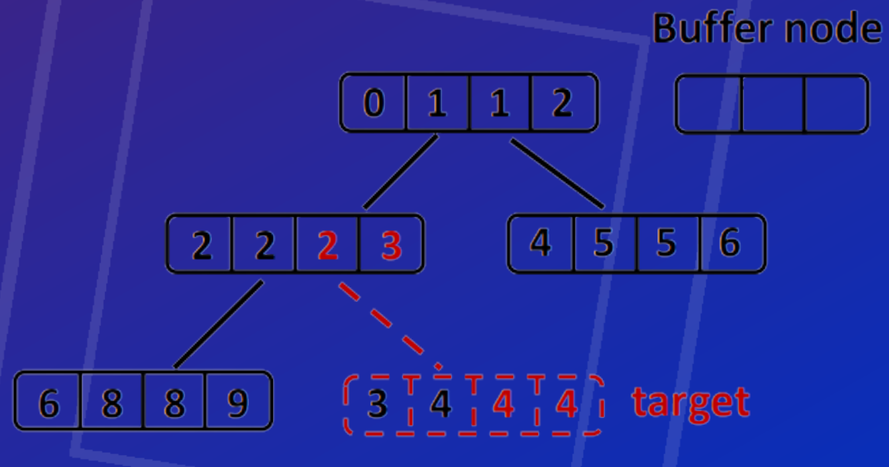
- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.



- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.

- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.



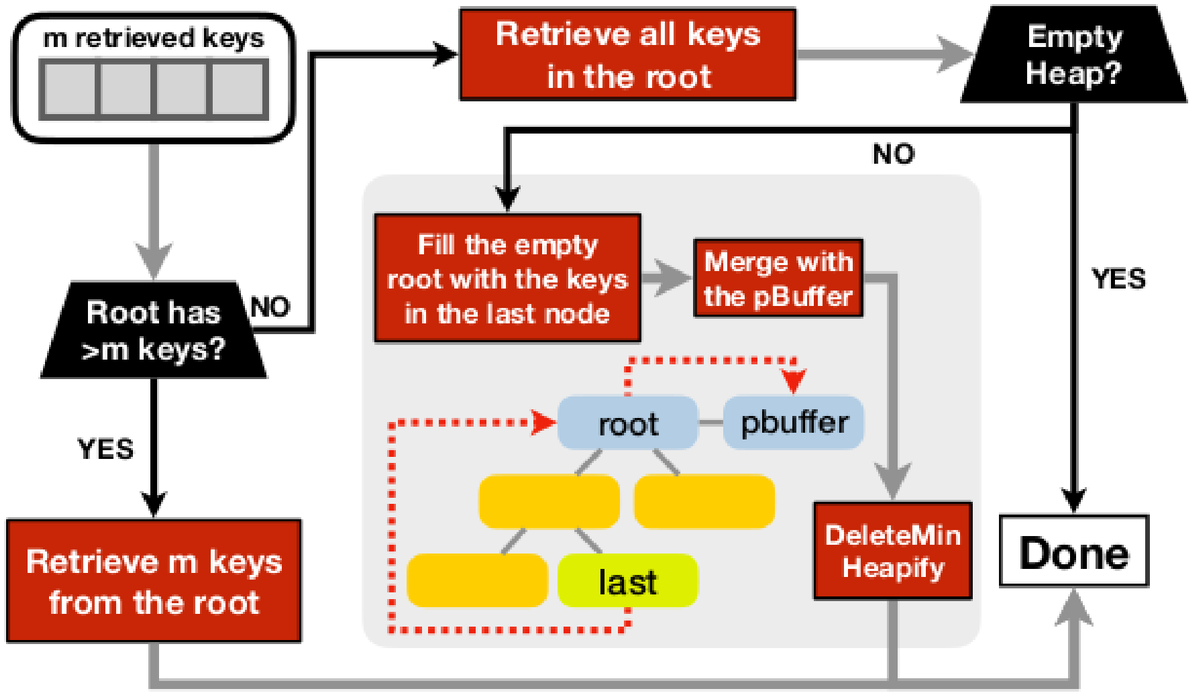
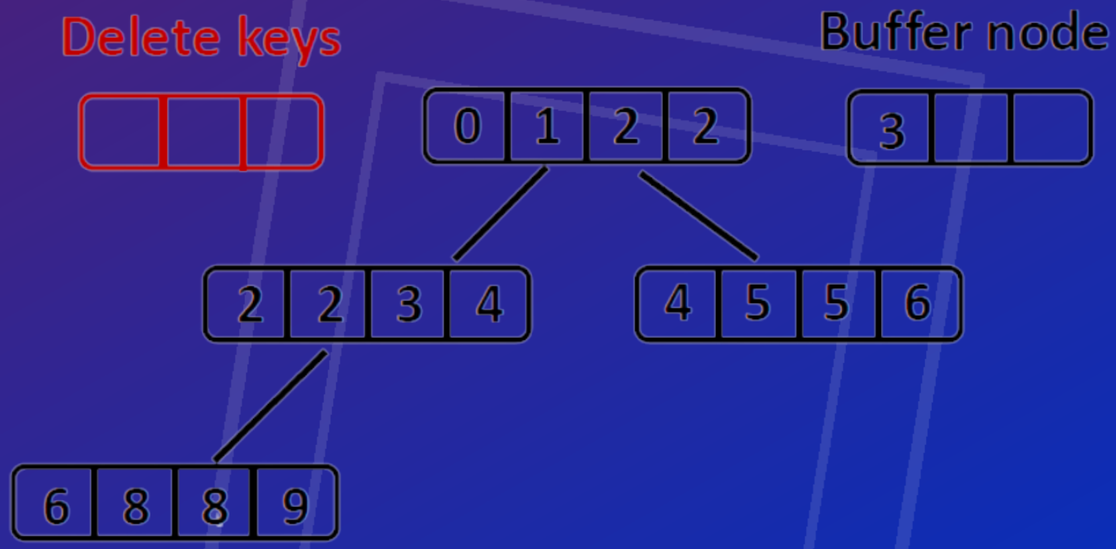


Figure 4: BGPQ DeleteMin Operations

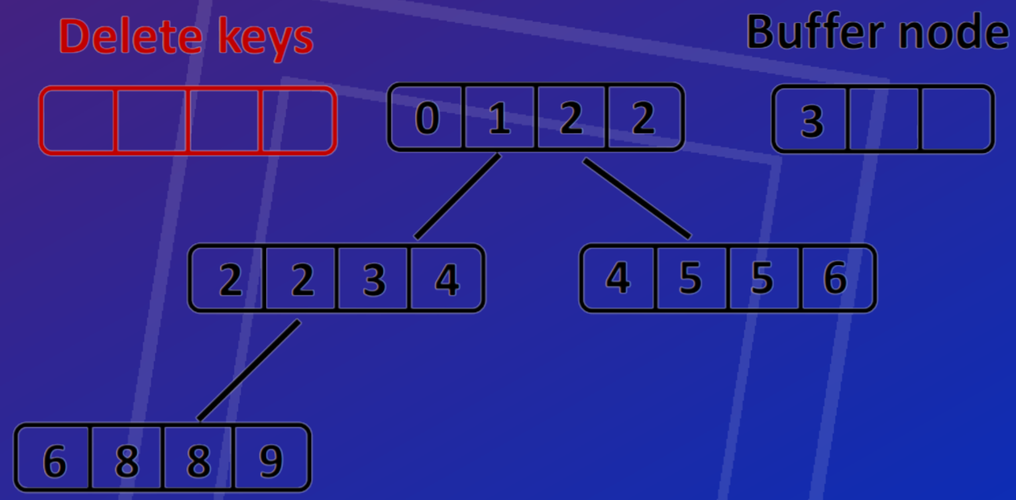
BGPQ DELETE OPERATION

- If the root contains enough keys to delete, they are retrieved directly.



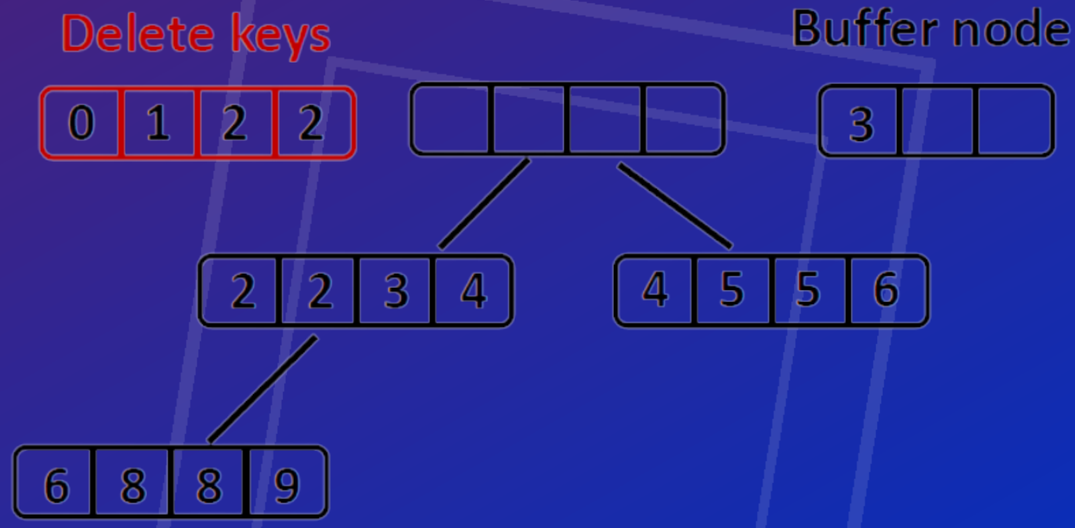
BGPQ Delete Operation

- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.



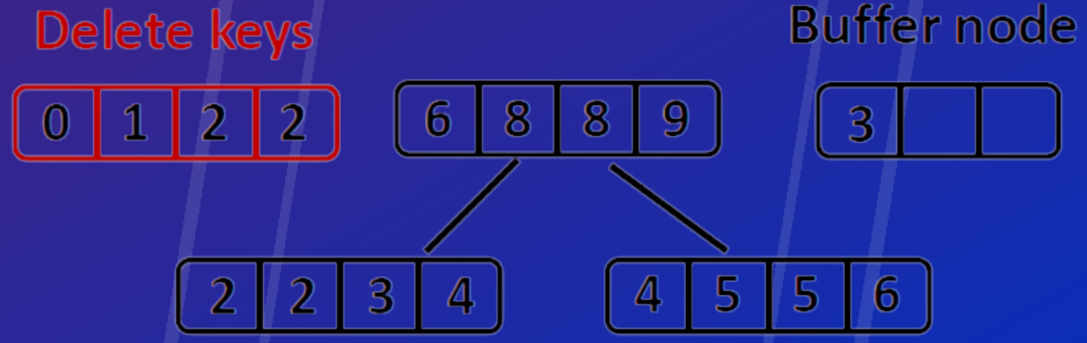
BGPQ Delete Operation

- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.



BGPQ Delete Operation

- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.



Conclusion

It was a wonderful learning experience for us while working on this project. The joy of working and the thrill involved while tackling the various problems and challenges gave us a feel of the industry. We studied existing concurrent priority queue implementations and their corresponding GPU friendliness and also successfully implemented BGPQ, a GPU-friendly priority queue. We enjoyed each and every bit of work we had put in.

Thank you!