INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING

ICPP/2021/CHICAGO/USA

acm In-Cooperation

sighpc

AUGUST 9-12, 2021

BGPQ: A Heap-Based Priority Queue Design for GPUs

Yanhao Chen
yc827@cs.rutgers.edu
Rutgers, The State University of New Jersey
New Brunswick, New Jersey, USA

Fei Hua
huafei90@gmail.com
Rutgers, The State University of New Jersey
New Brunswick, New Jersey, USA

Yuwei Jin
yj243@scarletmail.rutgers.edu
Rutgers, The State University of New Jersey
New Brunswick, New Jersey, USA

Eddy Z. Zhang
eddy.zhengzhang@gmail.com
Rutgers, The State University of New Jersey
New Brunswick, New Jersey, USA

Priority Queues

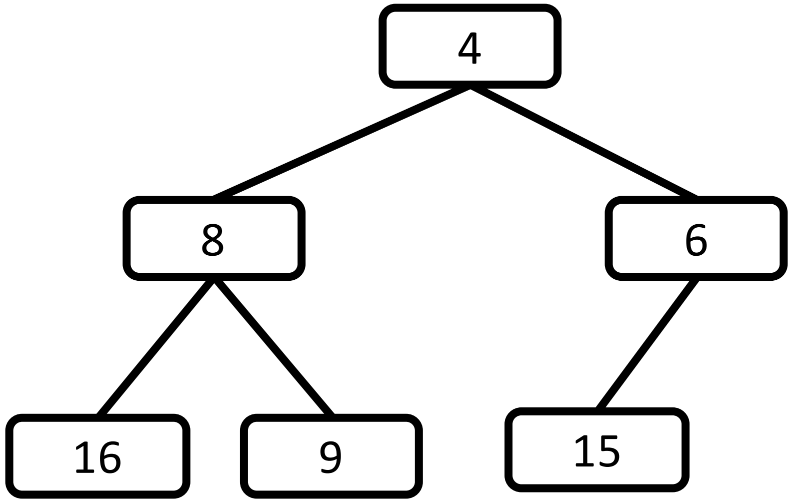
- Priority queues are fundamental data structures.
  - Insert and DeleteMin Operations.
  - Search problem: A* search algorithm
  - Graph problem: Dijkstra’s algorithm
  - Branch-and-Bound problem: 0/1 Knapsack

Concurrent Priority Queues

- Concurrent priority queues for **multi-core CPUs** have been well studied.
  - Heap based: Nageshwara and Kumar [21], Hunt et al.[14], …
  - Skip-List based: Sundell and Tsigas[27], Linden and Jonsson [16], …
  - Others: MD-List[30], CBPQ [3], …
- Very few studies on the concurrent priority queues for **many-core GPUs**.
  - Heap based: He et al.[12]
  - Skiplist based: Moscovici et al. [20]

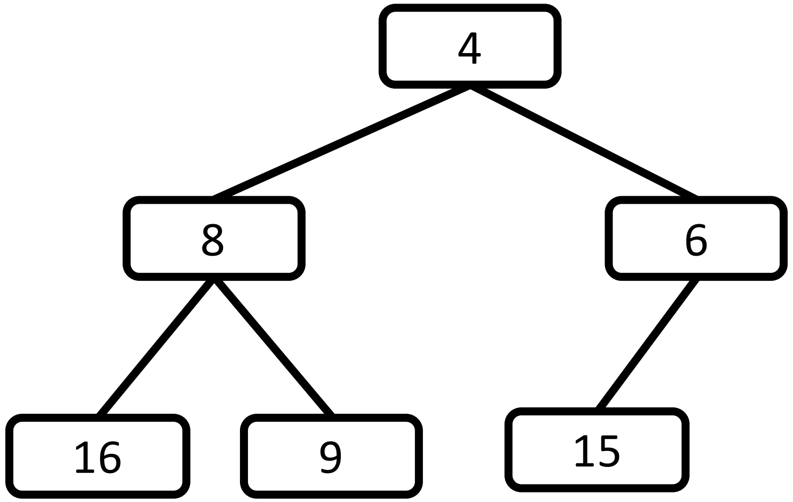
Concurrent Priority Queue Design Choices

Heap Based Priority Queue



Concurrent Priority Queue Design Choices

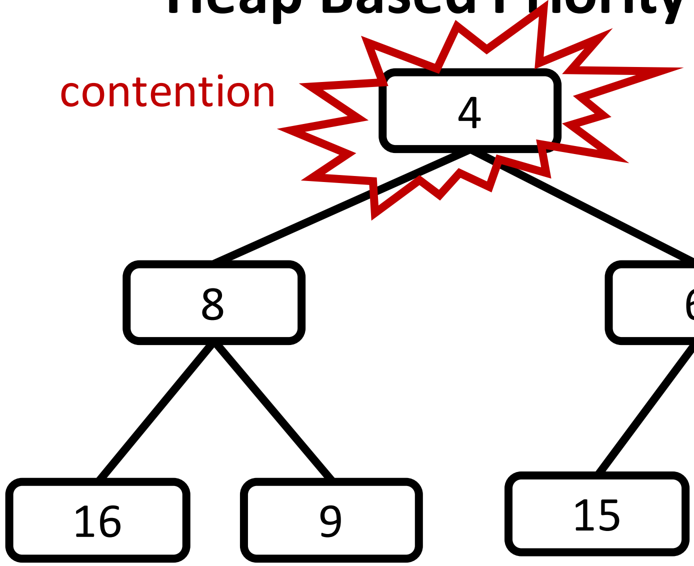
Heap Based Priority Queue



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

Concurrent Priority Queue Design Choices

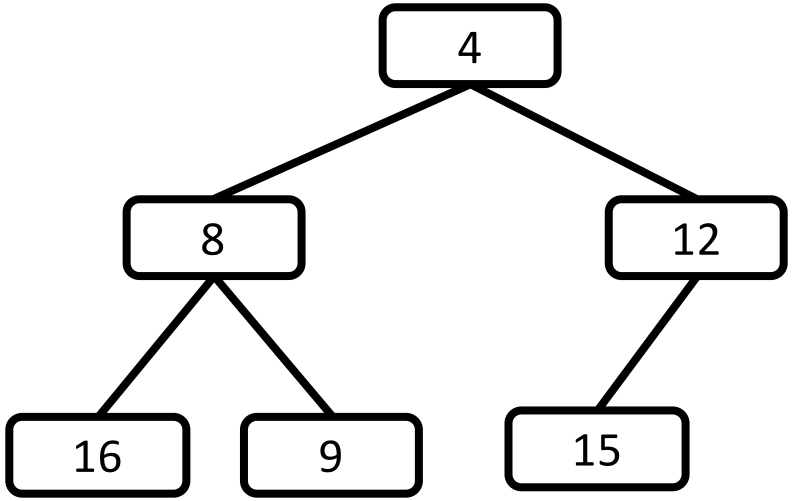
Heap Based Priority Queue



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

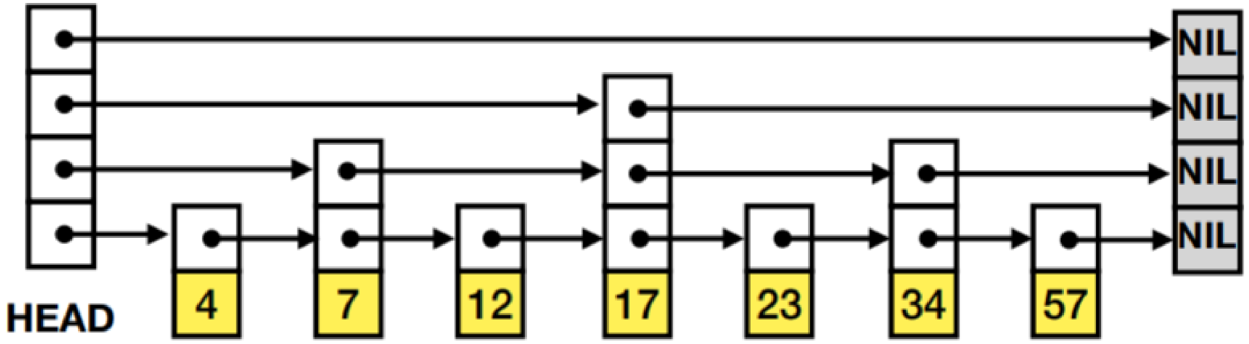
Concurrent Priority Queue Design Choices

Heap Based Priority Queue



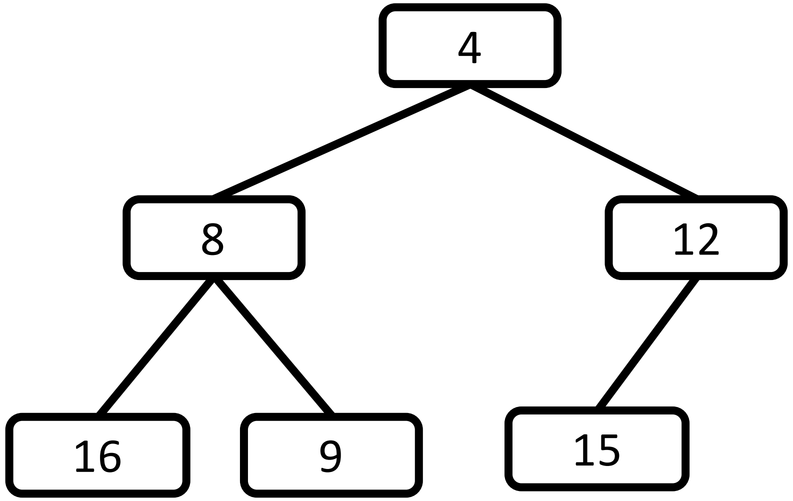
- Computation Complexity: O(logN)
- Space Complexity: N + O(1)
- Limitation: Scalability

Skip-List Based Priority Queue



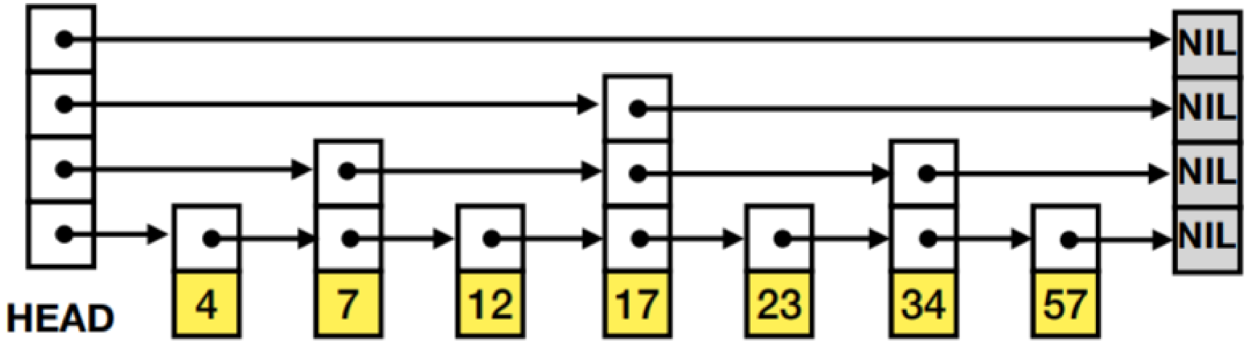
Concurrent Priority Queue Design Choices

Heap Based Priority Queue



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

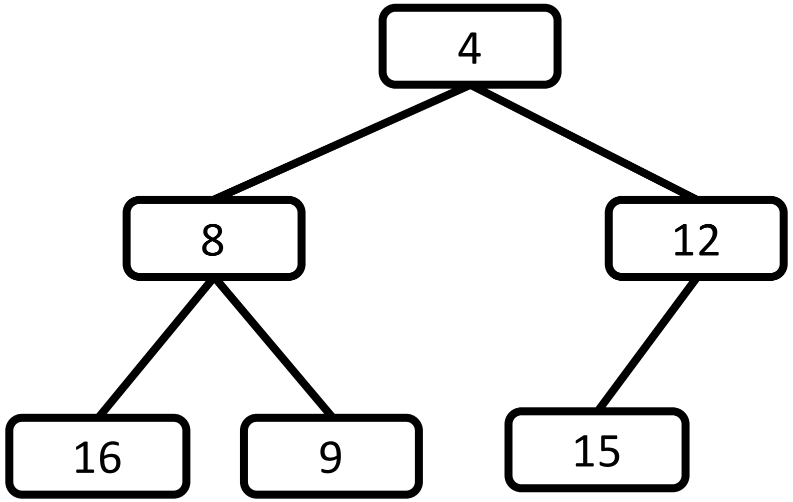
Skip-List Based Priority Queue



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

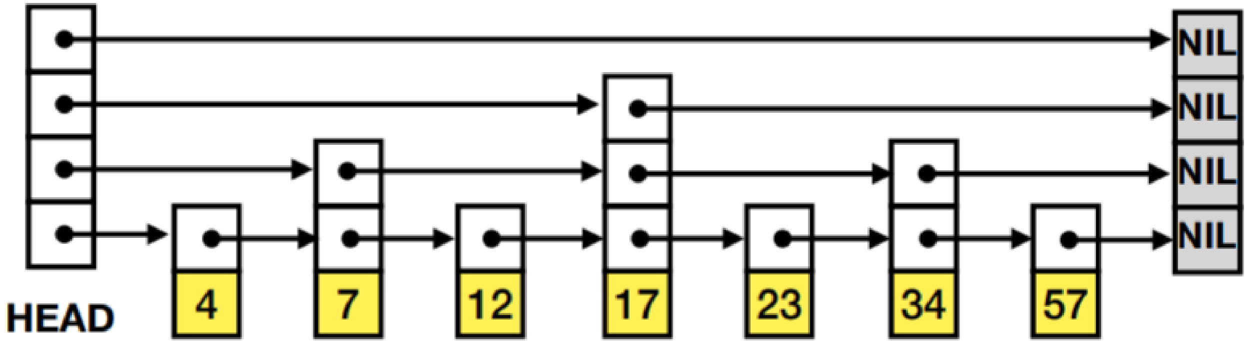
Concurrent Priority Queue Design Choices

Heap Based Priority Queue



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

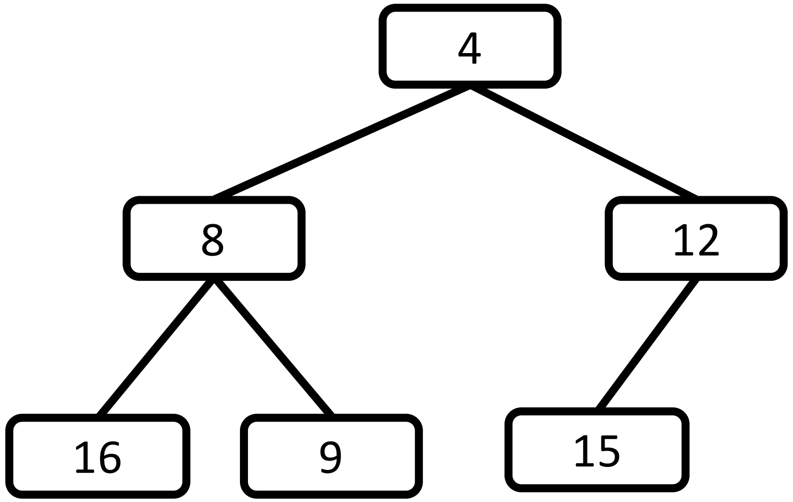
Skip-List Based Priority Queue



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

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.

Concurrent Priority Queue Design Choices

- Parallelism Exploitation
  - Most existing approaches exploit only task parallelism.
  - Very few studies have exploited data parallelism.
    - CPU: Deo and Prasad [8]
    - GPU: He et al.[12], Moscovici et al. [20]
- Operation collaboration

Concurrent Priority Queue Design Choices - Summary

	CPU Implementations					GPU Implementations		
	Hunt [14]	CBPQ [3]	STSL [27]	LJSL [16]	Spray List [1]	GFSL [20]	P-Sync [12]	BGPQ
Data Structure	Heap	Linked List + Skip-List	Skip-List	Skip-List	Skip-List	Skip-List	Heap	Heap
Data Parallelism	X	X	X	X	X	O	O	O
Task parallelism	O	O	O	O	O	O	O	O
Operation collaboration	X	O	X	X	X	X	X	O
Memory Efficiency	O	X	X	X	X	X	O	O

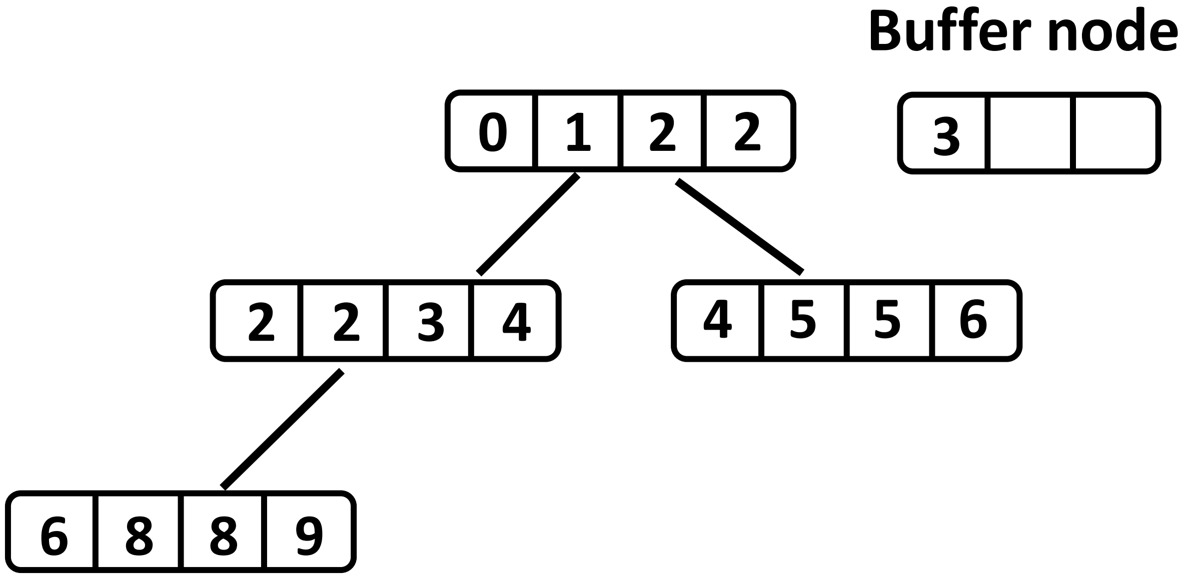
STSL: Sundell and Tsigas's skip-list based implementation.
LJSL: Linden and Jonssan's skip-list based implementation.
Spray List: a relaxed skip-list based implementation.
P-Sync: He et al.'s GPU heap-based implementation.

BGPQ Design Choices

- We proposed a **B**atched-based **G**PU **P**riority **Q**ueue (BGPQ).
- BGPQ uses the **Heap** as the underlying data structure.
- BGPQ exploits both task and data parallelism.
- Thread collaboration methods are used in BGPQ for better scalability.
- BGPQ is the first linearizable GPU priority queue implementation.

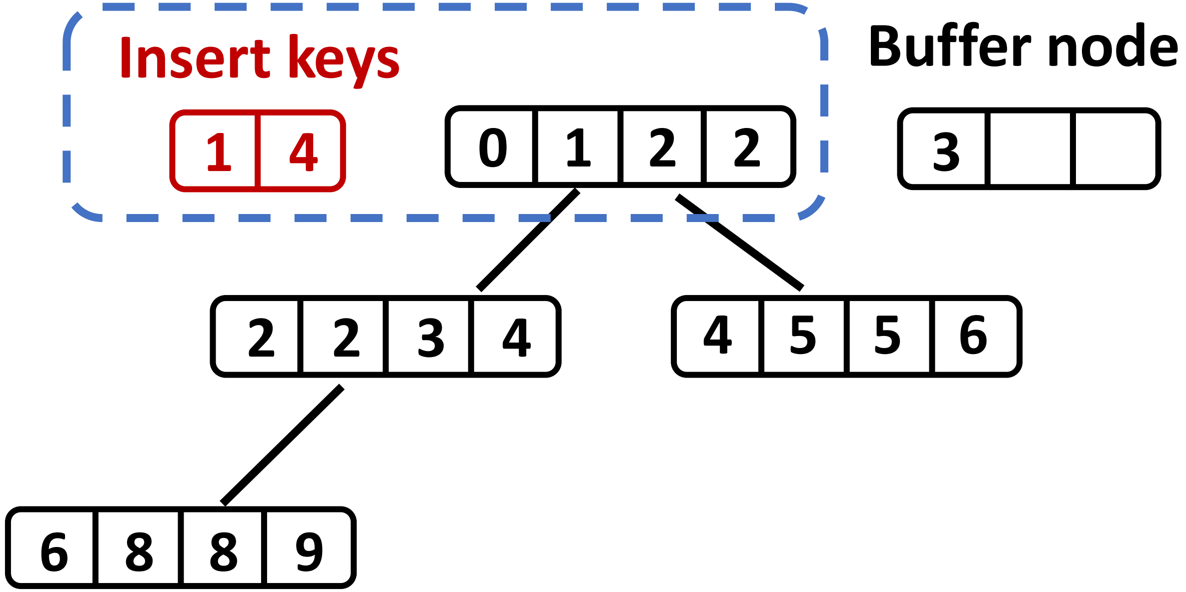
BGPQ Overview

- Each node contains exactly *k* keys.
  - Except root and buffer.
- The smallest key in the node is larger than the largest key of its children's.



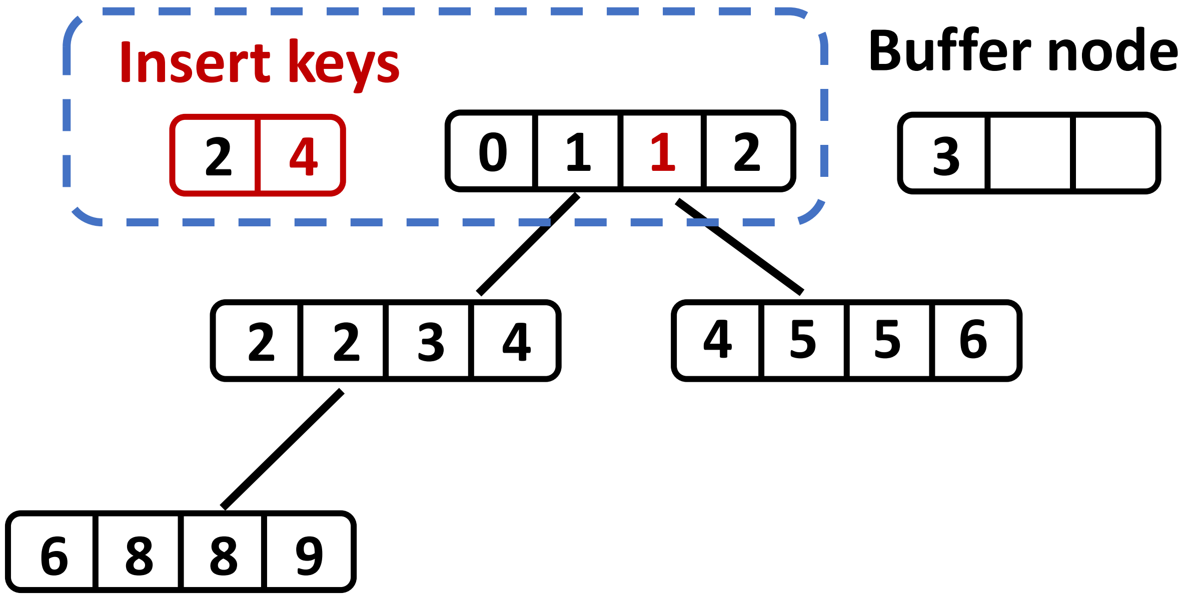
BGPQ Insert Operation

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



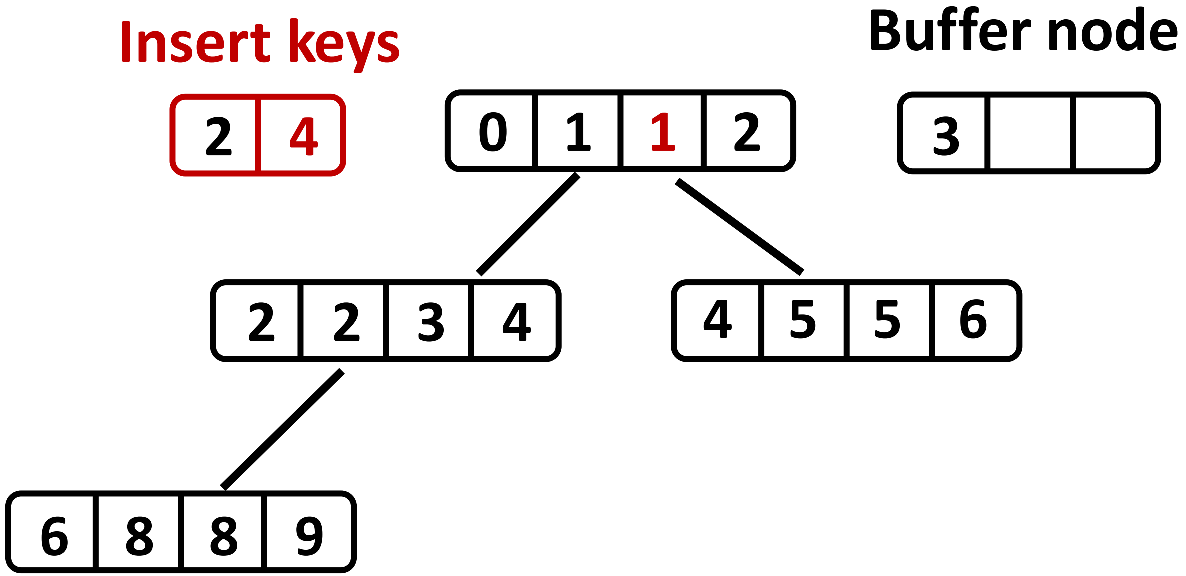
BGPQ Insert Operation

- 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.



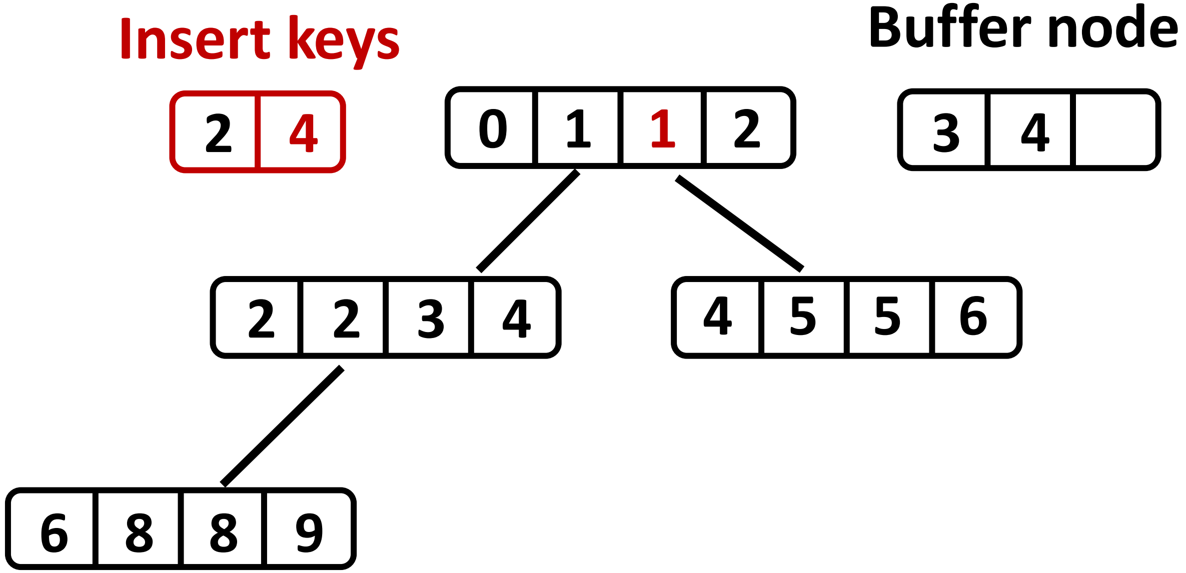
BGPQ Insert Operation

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



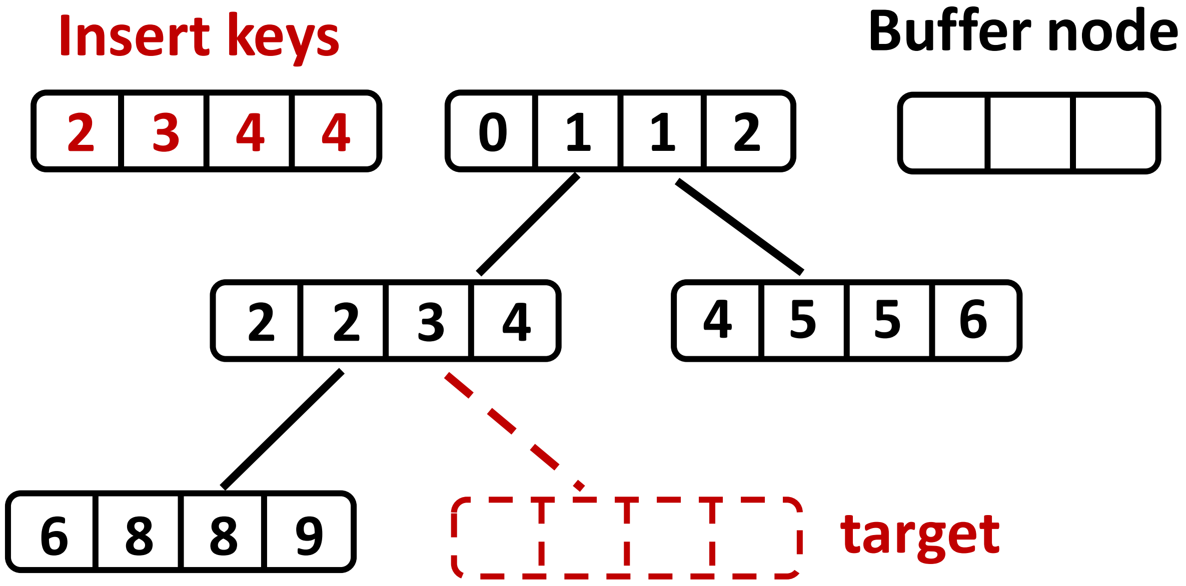
BGPQ Insert Operation

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



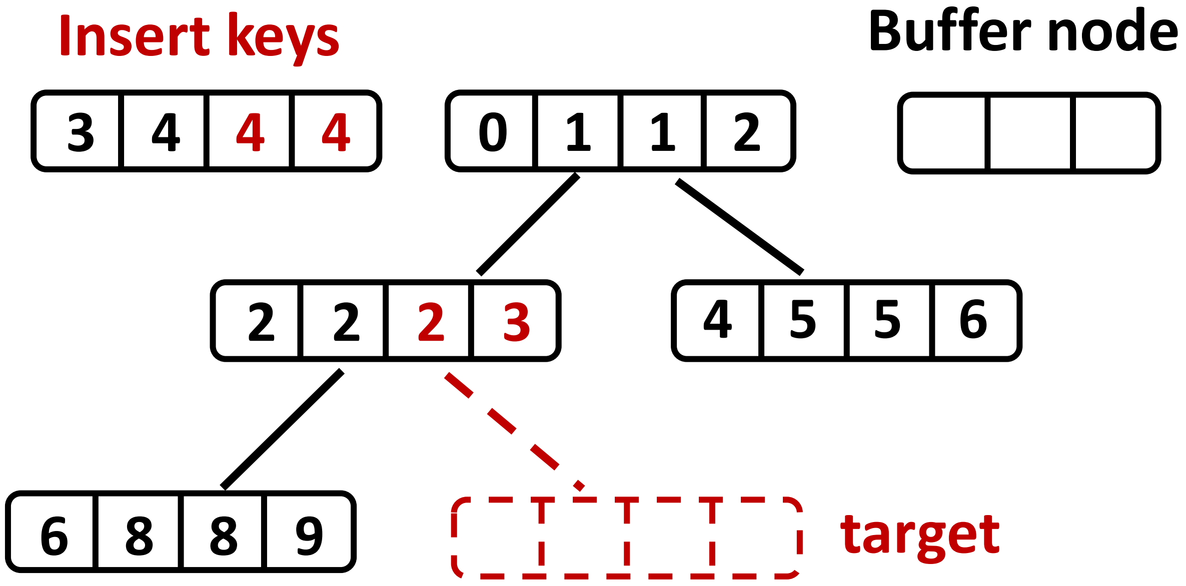
BGPQ Insert Operation

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



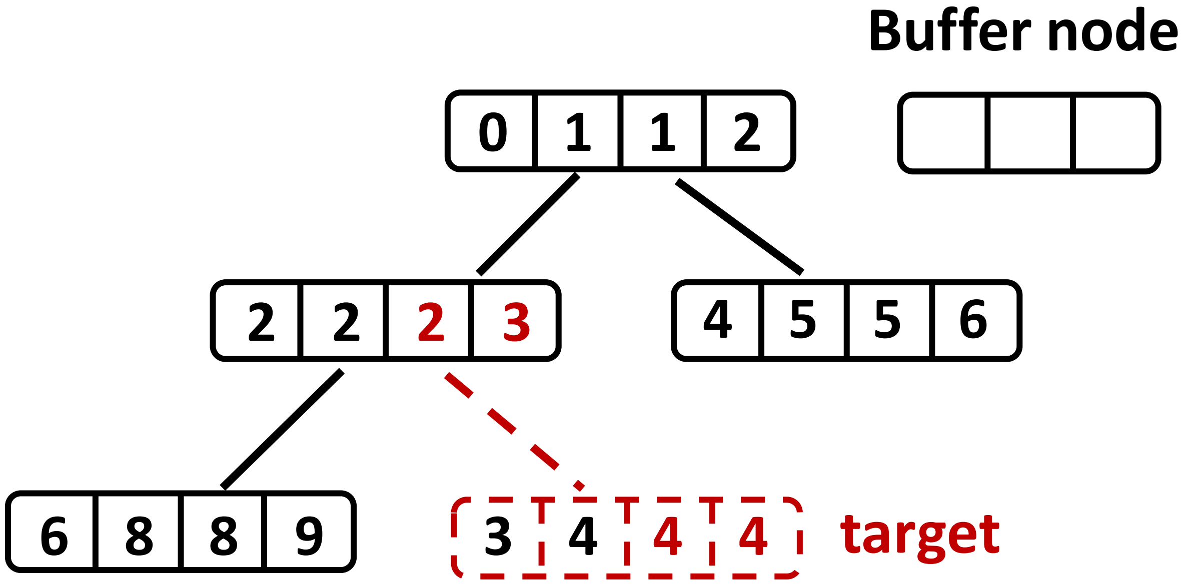
BGPQ Insert Operation

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



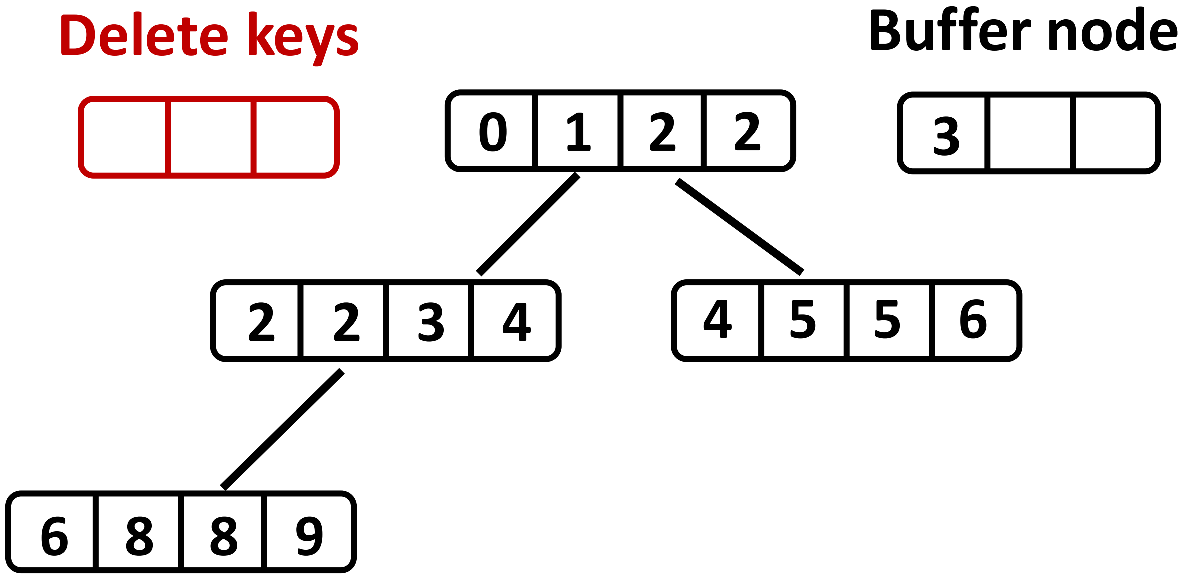
BGPQ Insert Operation

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



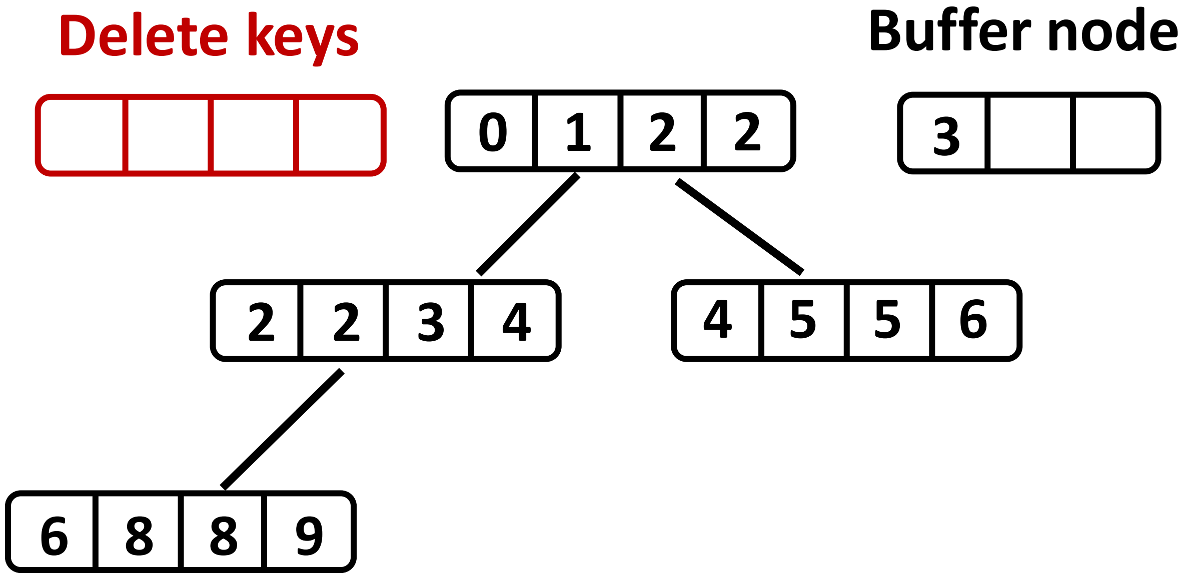
BGPQ DeleteMin Operation

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



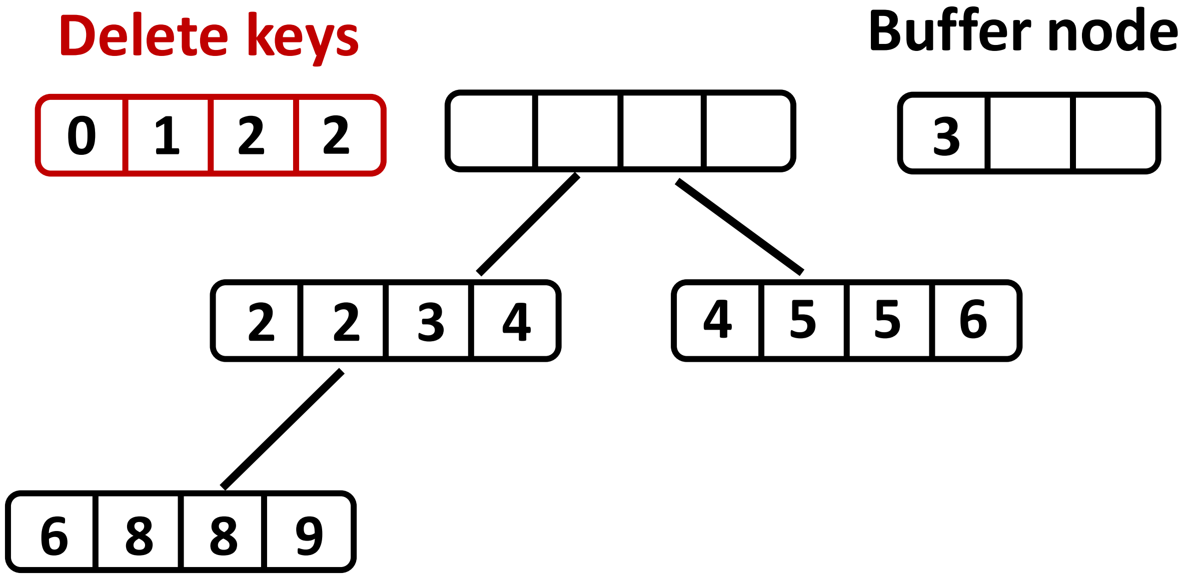
BGPQ DeleteMin Operation

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



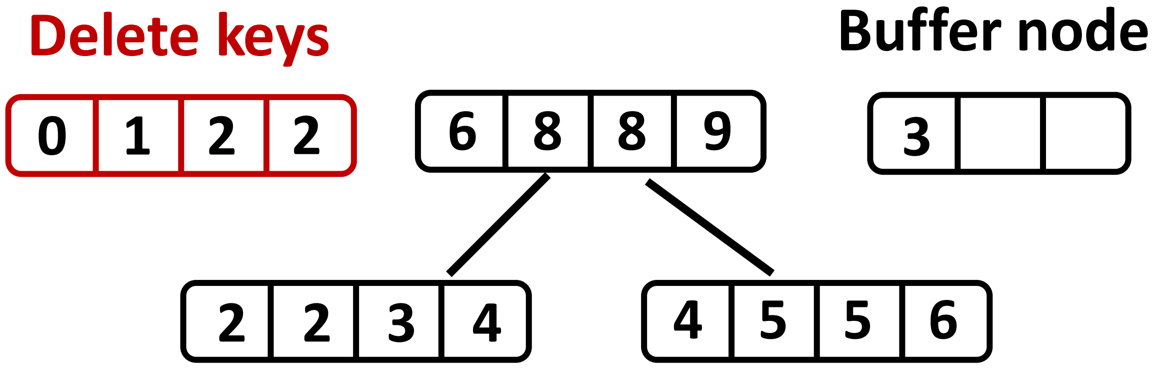
BGPQ DeleteMin Operation

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



BGPQ DeleteMin Operation

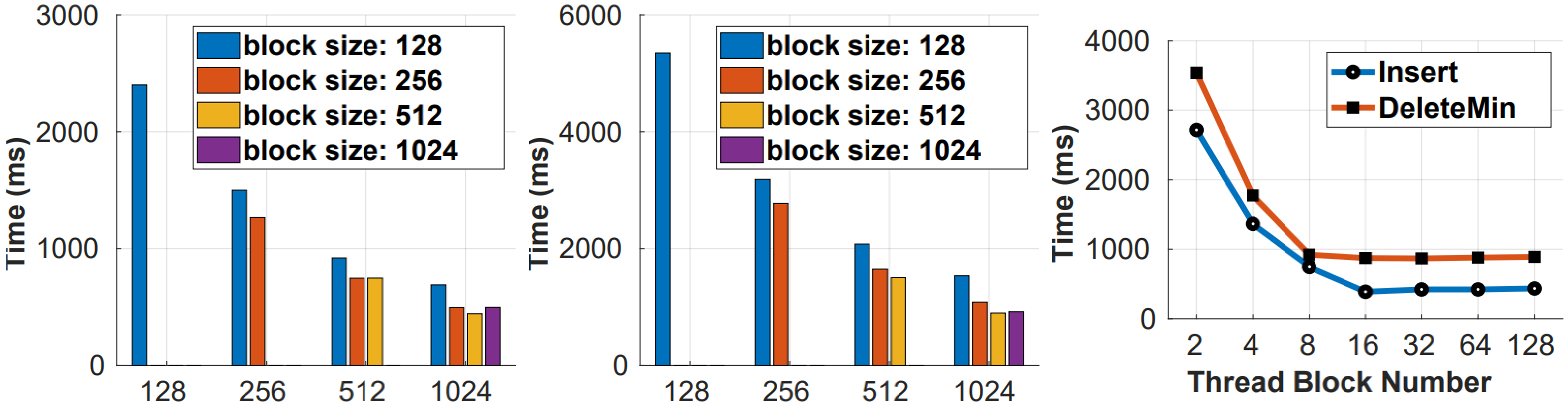
- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.



Evaluation

- We compare BGPQ with:
- Four CPU baselines
  - Intel Thread Building Blocks - TBB [29]
  - Chunk Based Priority Queue - CBPQ [3]
  - Skip-List based Priority Queue - LJSL [16] and SprayList [1]
- Four Xeon E7-4879 processors, 1TB memory, 2.4 GHz, 80 threads
- One GPU baseline
  - He et al. – P-Sync [12]
- Nvidia TITAN X GPU w/ 28 SMs

Evaluation - BGPQ Performance

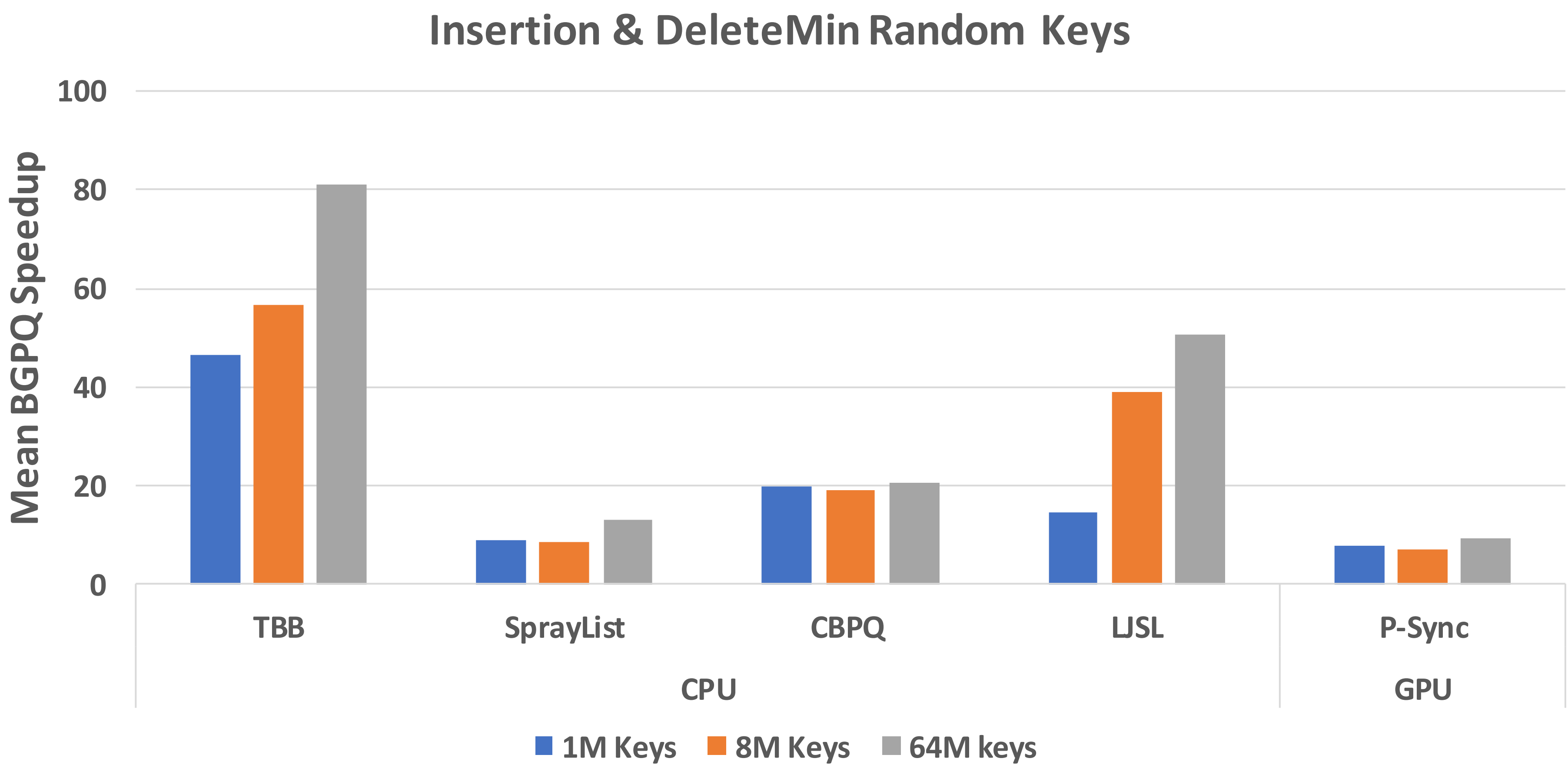


(a) Insert operation

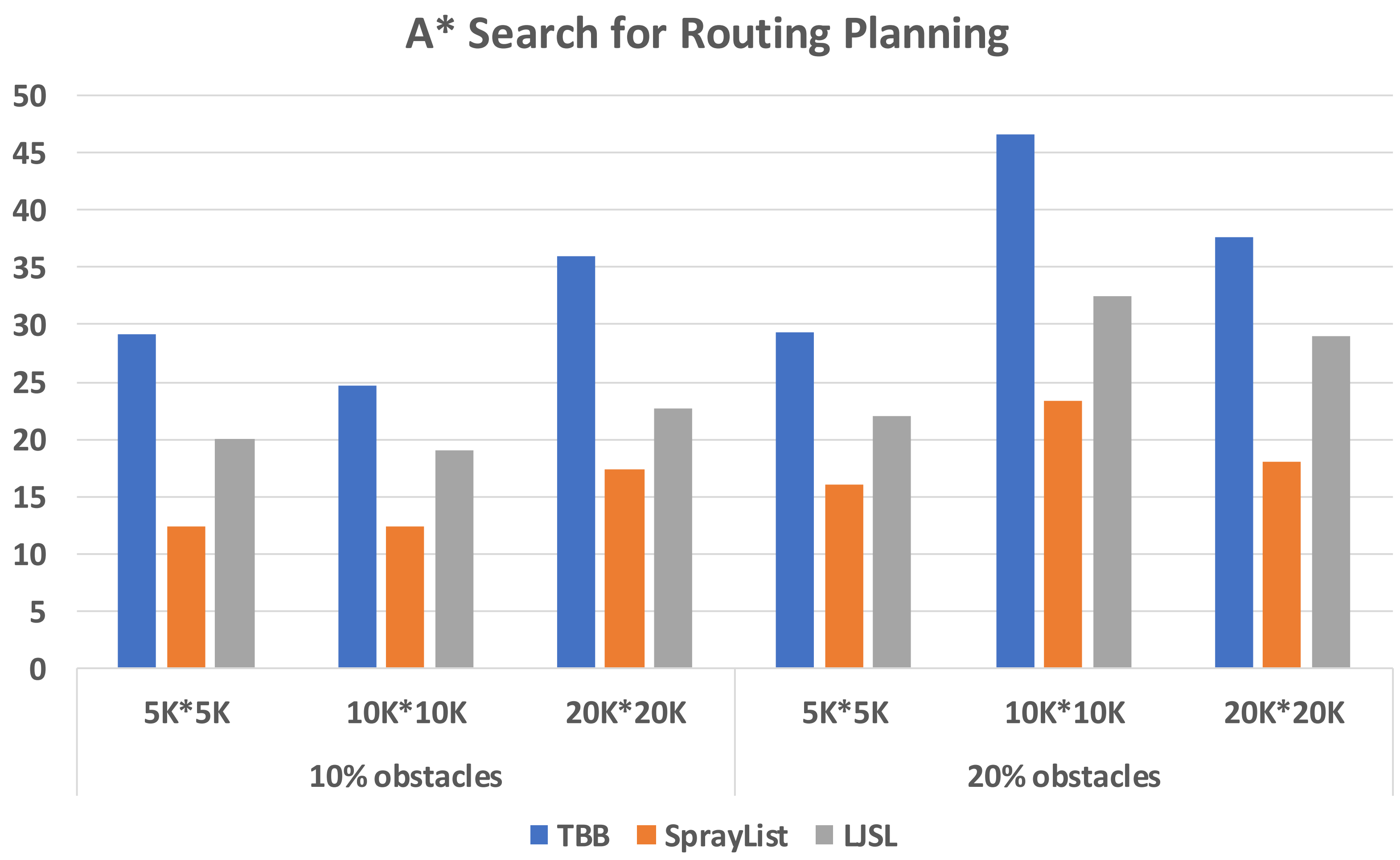
(b) DeleteMin operation

(c) performance w.r.t thread block numbers

Evaluation - BGPQ Performance



Evaluation – A* searching



Conclusion

- We study existing concurrent priority queue implementations and their corresponding GPU friendliness.
- We present BGPQ, a GPU-friendly priority queue implementation.
- We compare the performance of BGPQ with 4 CPU baselines and 1 GPU baseline and show significant speedup.

Question?