Binomial Queues

CSE 373 - Data Structures
April 29, 2002

Readings and References

- Reading
  - › Section 6.8, *Data Structures and Algorithm Analysis in C*, Weiss
- Other References

Merging heaps

- Binary Heap is a special purpose hot rod
  - › FindMin, DeleteMin and Insert only
  - › does not support fast merges of two heaps
- For some applications, the items arrive in prioritized clumps, rather than individually
- Is there somewhere in the heap design that we can give up a little performance so that we can gain faster merge capability?

Binomial Queues

- Binomial Queues are designed to be merged quickly with one another
- Using pointer-based design we can merge large numbers of nodes at once by simply pruning and grafting tree structures
- More overhead than Binary Heap, but the flexibility is needed for improved merging speed

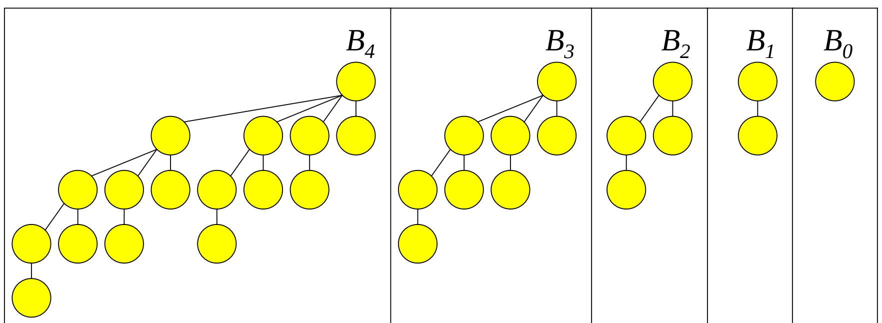
Worst Case Run Times

	Binary Heap	Binomial Queue
Insert	Θ(log N)	Θ(log N)
FindMin	Θ(1)	O(log N)
DeleteMin	Θ(log N)	Θ(log N)
Merge	Θ(N)	O(log N)

Binomial Queues

- Binomial queues give up Θ(1) FindMin performance in order to provide O(log N) merge performance
- A **binomial queue** is a collection (or *forest*) of heap-ordered trees
  - › Not just one tree, but a collection of trees
  - › each tree has a defined structure and capacity
  - › each tree has the familiar heap-order property

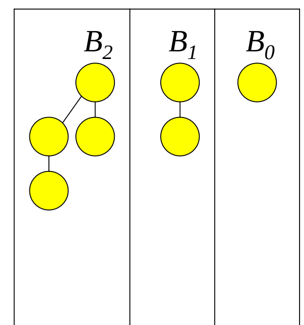
Binomial Queue with 5 Trees



depth	4	3	2	1	0
number of elements	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Structure Property

- Each tree contains two copies of the previous tree
  - › the second copy is attached at the root of the first copy
- The number of nodes in a tree of depth d is exactly 2^d



depth	2	1	0
number of elements	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Powers of 2

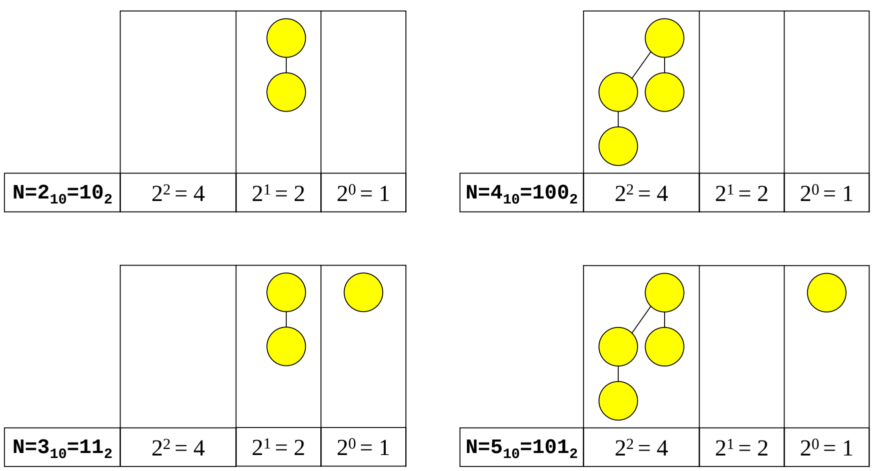
- Any number N can be represented in base 2
  - › A base 2 value identifies the powers of 2 that are to be included

$2^3 = 8_{10}$	$2^2 = 4_{10}$	$2^1 = 2_{10}$	$2^0 = 1_{10}$	Hex_{16}	$Decimal_{10}$
		1	1	3	3
	1	0	0	4	4
	1	0	1	5	5

Numbers of nodes

- Any number of entries in the binomial queue can be stored in a forest of binomial trees
- Each tree holds the number of nodes appropriate to its depth, ie 2^d nodes
- So the <u>structure</u> of a forest of binomial trees can be characterized with a single binary number
  - › $100_2 \rightarrow 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 4$ nodes

Structure Examples



What is a merge?

- There is a direct correlation between
  - › the number of nodes in the tree
  - › the representation of that number in base 2
  - › and the actual structure of the tree
- When we merge two queues, the number of nodes in the new queue is the *sum of* N_1+N_2
- We can use that fact to help see how fast merges can be accomplished

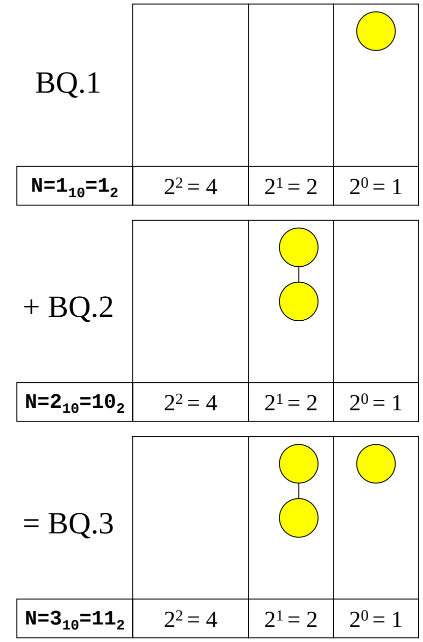
Merge by adding the trees

- A merge of two queues can be viewed as adding the two sets of trees together
  - › 0+0 = 0 $\rightarrow$ neither queue has a tree at that position and so neither does the sum
  - › 0+1 = 1 $\rightarrow$ only one of the queues has a tree at that position, and so it is copied into the sum
  - › ...

Merge BQ.1 and BQ.2

Note that nothing was done with any of the nodes in order to accomplish this.

There are no comparisons and there is no restructuring.



Merge by adding the trees

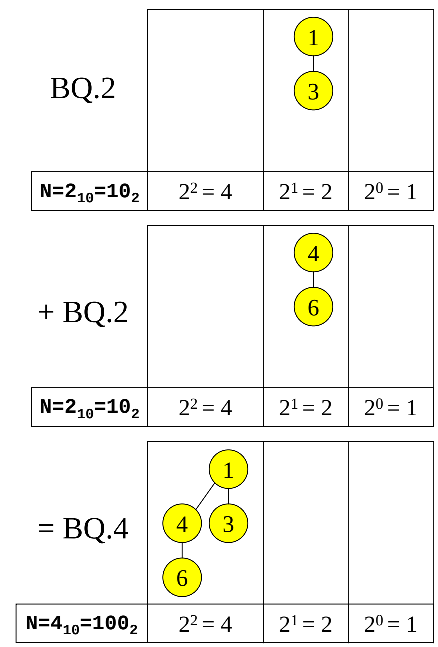
- A merge of two queues can be viewed as adding the two sets of trees together
  - › ...
  - › 1+1 = 2_{10} = 10_2 $\rightarrow$ both queues have a tree at that position and so the sum has a double-sized tree at the next higher position and nothing at the current position
  - › ...

Merge BQ.2 and BQ.2

There are two trees at position 1. So attach the tree with the larger root as a child of the tree with the smaller root, and put the resulting tree in the next higher position.

This is an add with a carry out.

It is accomplished with one comparison and one pointer change: O(1)



Merge by adding the trees

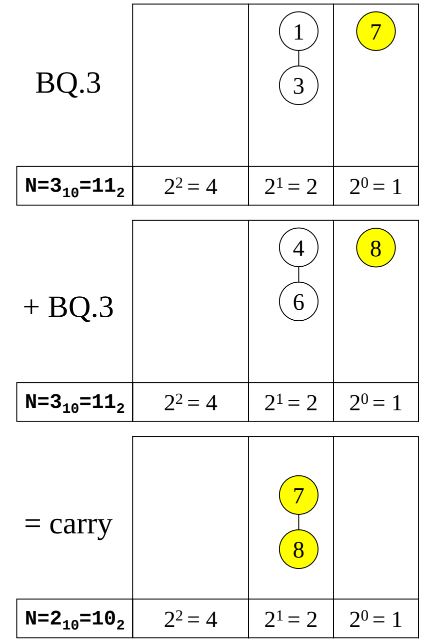
- A merge of two queues can be viewed as adding the two sets of trees together
  - ...
  - 1+1 + carry = $3_{10} = 11_2 \rightarrow$ both queues have a tree at that position and there is a carry from the previous position and so the sum has a double-sized tree at the next higher position and a tree at the current position

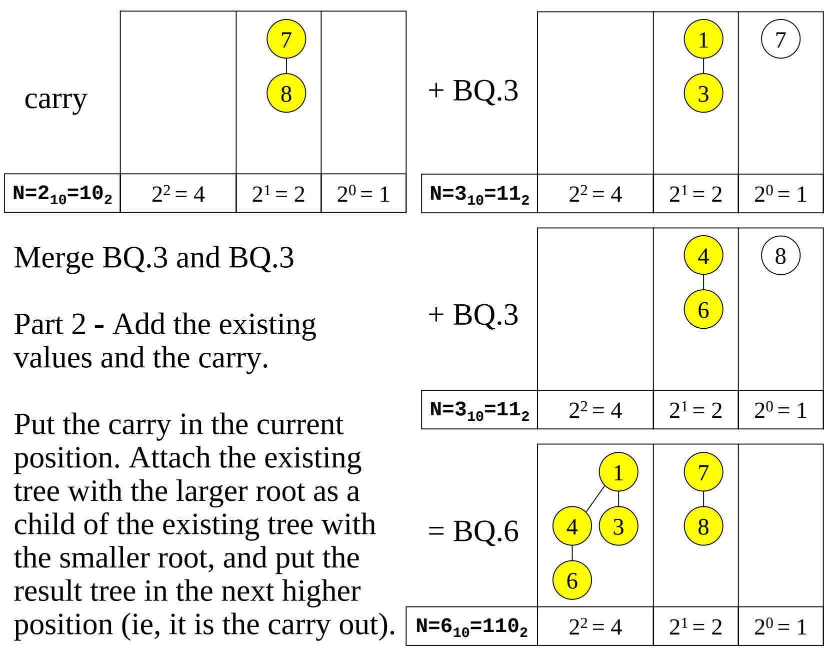
Merge BQ.3 and BQ.3

Part 1 - Form the carry.

There are two trees at position 0. So attach the tree with the larger root as a child of the tree with the smaller root, and put the resulting tree in the next higher position.

This is an add with a carry out.





Merge BQ.3 and BQ.3

Part 2 - Add the existing values and the carry.

Put the carry in the current position. Attach the existing tree with the larger root as a child of the existing tree with the smaller root, and put the result tree in the next higher position (ie, it is the carry out).

High Speed Merging

- Notice that although there are lots of nodes involved, the actual merge operation only touches the root nodes of a few trees
- Very fast compared to inserting the contents of an entire heap as we would have to do with binary heaps which would be $\Theta(N)$
- There are log N trees in each Binomial Queue and so the merge is O(log N)

Binomial Queues: Insert

- How would you insert a new item into the queue?
  - › Create a single node queue B_0 with the new item and merge with existing queue
  - › Again, O(log N) time

Binomial Queues: DeleteMin

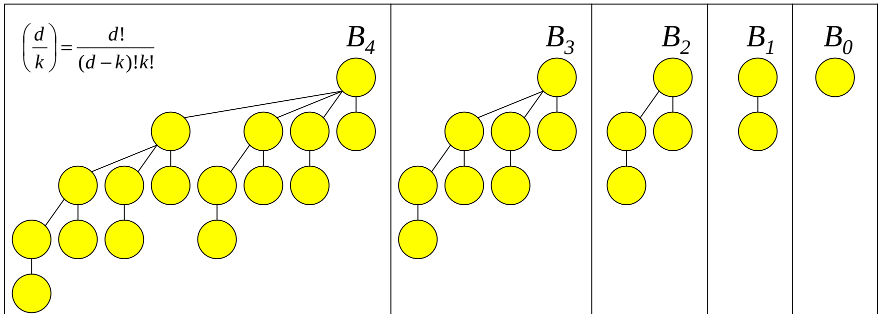
- Steps:
  - › Find tree B_k with the smallest root O(log N)
  - › Remove B_k from the queue O(1)
  - › Remove root of B_k (return this value) O(1)
    - You now have a new queue made up of the forest B_0, B_1, …, B_{k-1}.
  - › Merge this new queue with remainder of the original (from step 2) O(log N)
- Total time = O(log N)

Implementation

- Merge adds one binomial tree as child to another and DeleteMin requires fast access to all subtrees of root
  - › Need pointer-based implementation
  - › Use First-Child/Next-Sibling representation of trees
  - › Use array of pointers to root nodes of binomial trees

Why Binomial?

$$\binom{d}{k} = \frac{d!}{(d-k)!k!}$$



tree depth d	4	3	2	1	0
nodes at depth k	1, 4, 6, 4, 1	1, 3, 3, 1	1, 2, 1	1, 1	1