Binomial Heaps

(This is from a chapter in the second edition of the text that is no longer in the third edition)

Priority queues

The heap data structure is not only used in sorting but also in priority queues. A **priority queue** is a data structure that maintains a set S of elements, each with an associated value call a **key**. Priority queues has many applications. A **max (or min) priority queue** support the following operations:

- **Insert** (S, x) — $O(\log n)$
- **Maximum** (S) — $O(1)$
- **Extract-Max** (S) — $O(\log n)$
- **Increase-Key** (S, x, k) — $O(\log n)$
- **Decrease-Key** (S, x, k) — $O(\log n)$

Problem with Binary Heap

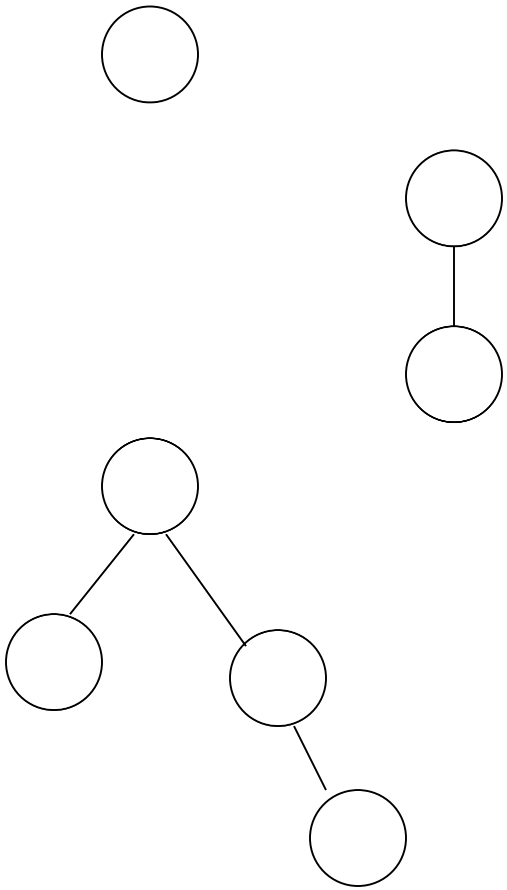
- Supports efficient insert and Extract-Min
- But merging two heaps (with total n nodes) will require O(n) time
- Not good enough for many applications
- So we now look at a kind of *Mergeable* heap which supports O(logn) merges.

Binomial Heaps

- Also called Binomial Queue (BQ)
- Consists of a forest of Binomial Trees (BiT)
  - Each BiT is a heap

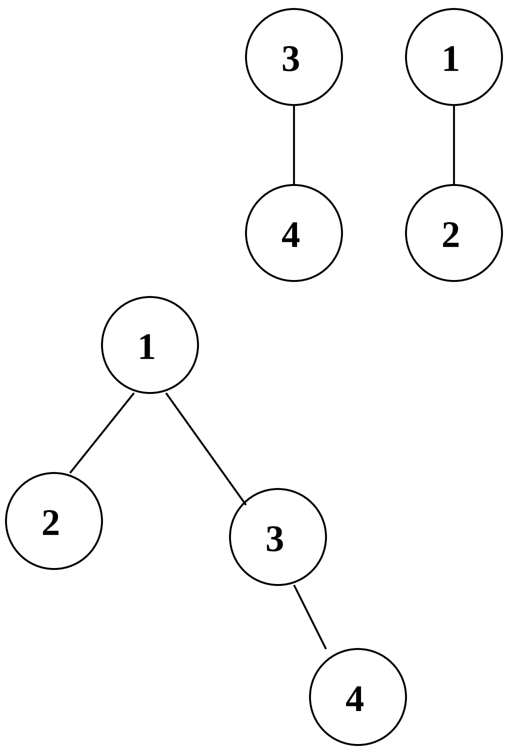
BiT

- B_k : Bit of height k, k=0,1,2,...
- B_0 : 1-node tree
- B_1 : 2-node tree
- B_2 : 4-node tree
- B_i : 2^i-node tree



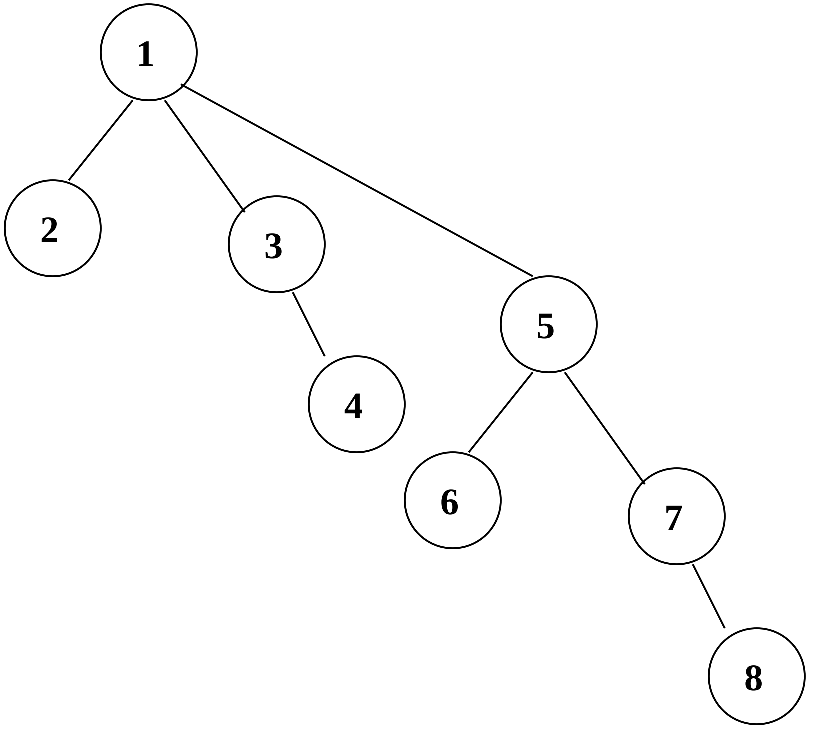
BiT

You construct a B_k tree by attaching a B_{k-1} tree to the root of another B_{k-1} tree, making sure that the Heap Property is preserved



BiT

A B_k tree therefore consists of a root with k child subtrees: B_0 , B_1 .... B_{k-1} trees

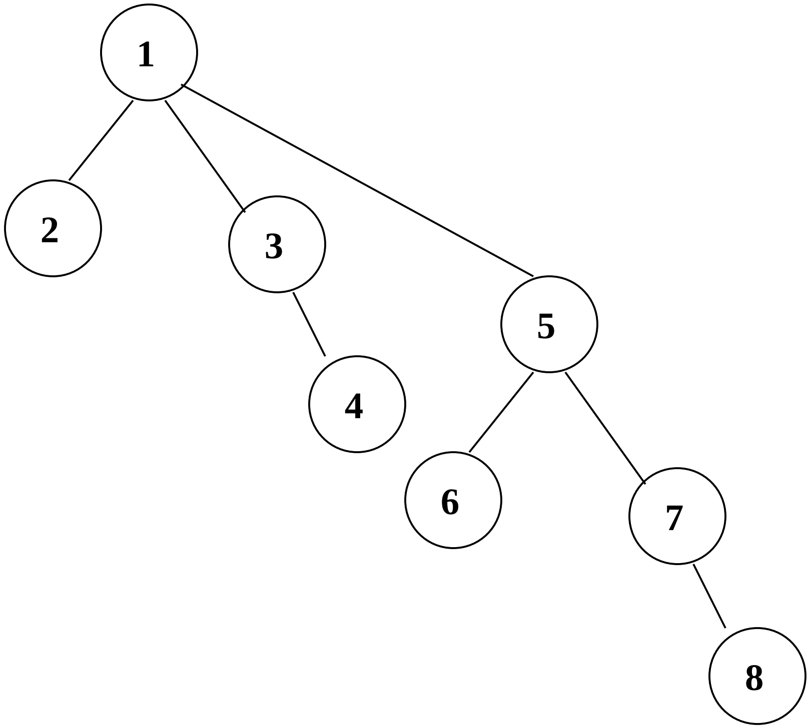


BiT

No. of nodes at depth d in a B_k tree is given by the "binomial coefficient"

$k!/d!(k-d)!$

Hence the name Binomial Tree



BQ

- A forest – a collection of heap-ordered trees
- Each tree is a Binomial (not Binary!) Tree
- At most one Binomial Tree (BiT) of any height in a BQ
- So a BQ is a collection of trees and satisfies:
  - Heap property
  - Structural property

BQ

- A priority queue of any size n can be uniquely represented by a BQ of size n
- To see how many and which BiTs are in the BQ, look at the binary representation of n
- This means that there will be $\lfloor \log n \rfloor + 1$ (i.e. $O(\log n)$) Binomial Trees in a BQ of size n

BQ operations

- Merge is the fundamental operation
- Merge can be done in O(logn) worst case time
- Other operations in terms of Merge

BQ Merge

- Merging two B_k trees to get a B_{k+1} tree is O(1) – why?
- How do we Merge two BQs?
  - Conceptually similar to binary addition of the two numbers representing the sizes of the two BQs to be merged.

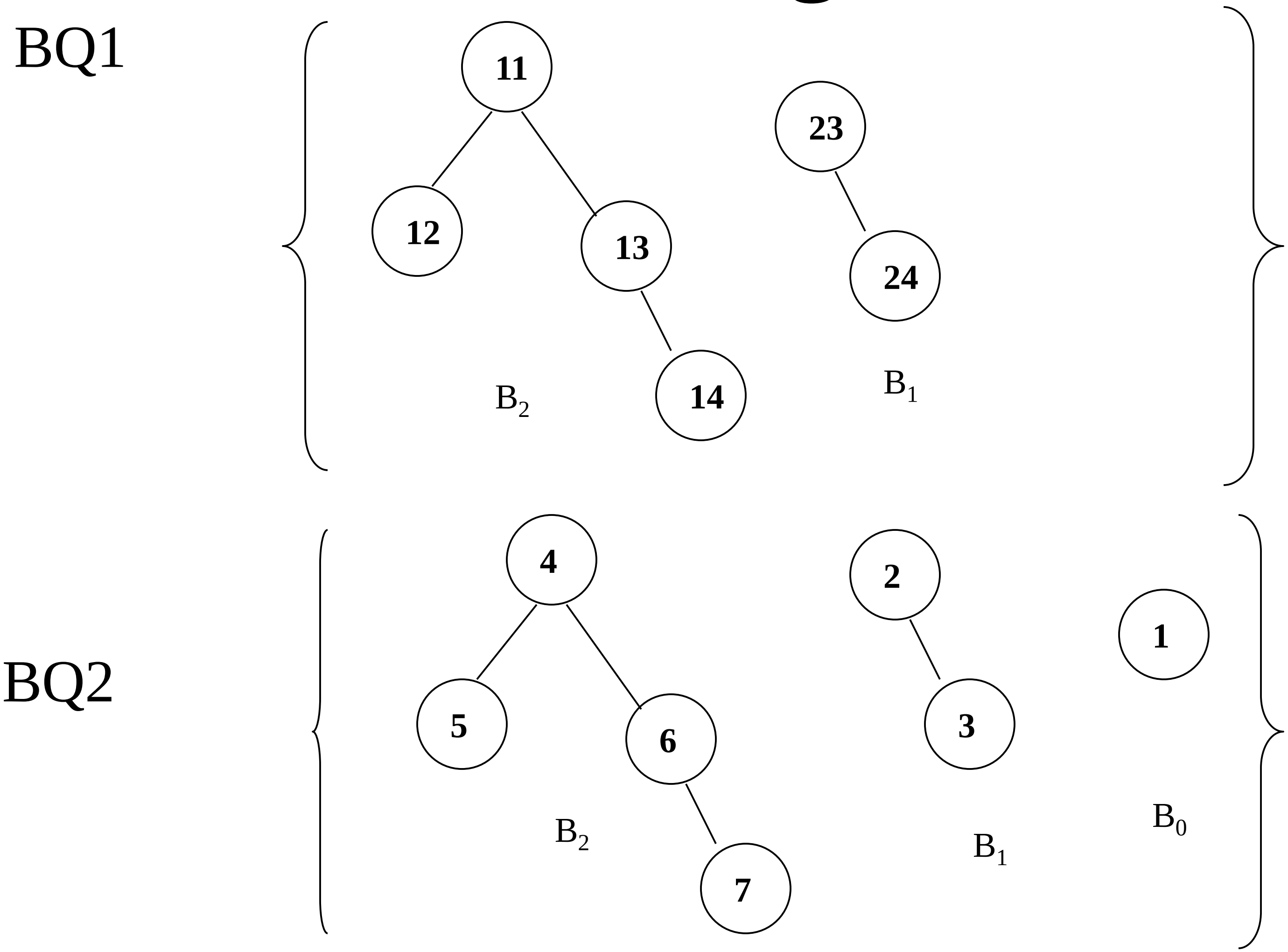
BQ Merge

- Suppose you want to merge two BQs:
  - BQ1 of size 6 = 110 $\rightarrow \{B_2, B_1\}$
  - BQ2 of size 7 = 111 $\rightarrow \{B_2, B_1, B_0\}$
  - The merged BQ will be of size 13 = 1101 $\rightarrow \{B_3, B_2, B_0\}$

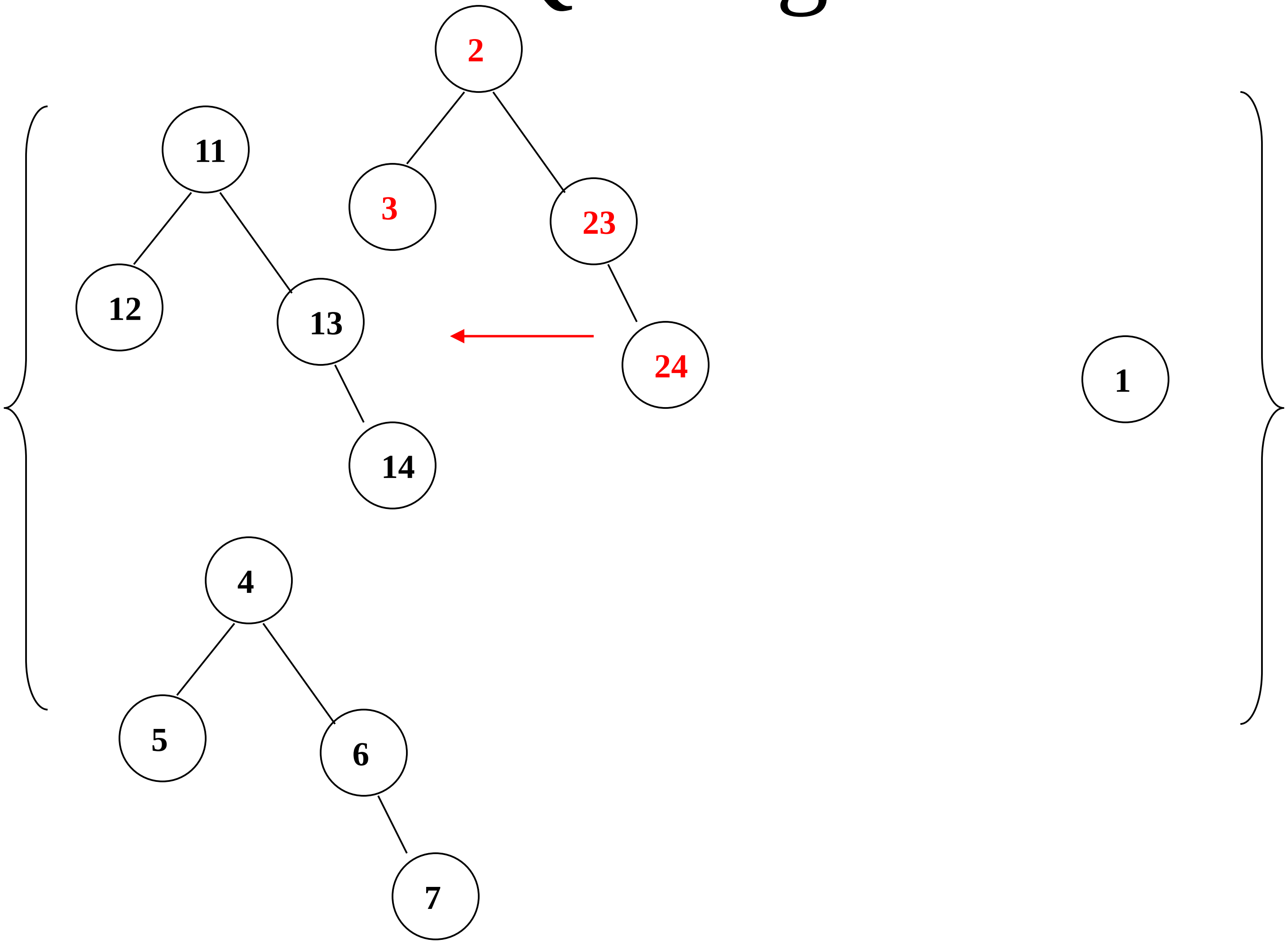
- The process is similar to binary addition with carry:

$$
\begin{array}{cccc}
\xleftarrow{\;1\;} & 1 \;\xleftarrow{\;1\;} & 1 & 0 \\
 & 1 & 1 & 1 \\
\hline
1 & 1 & 0 & 1
\end{array}
$$

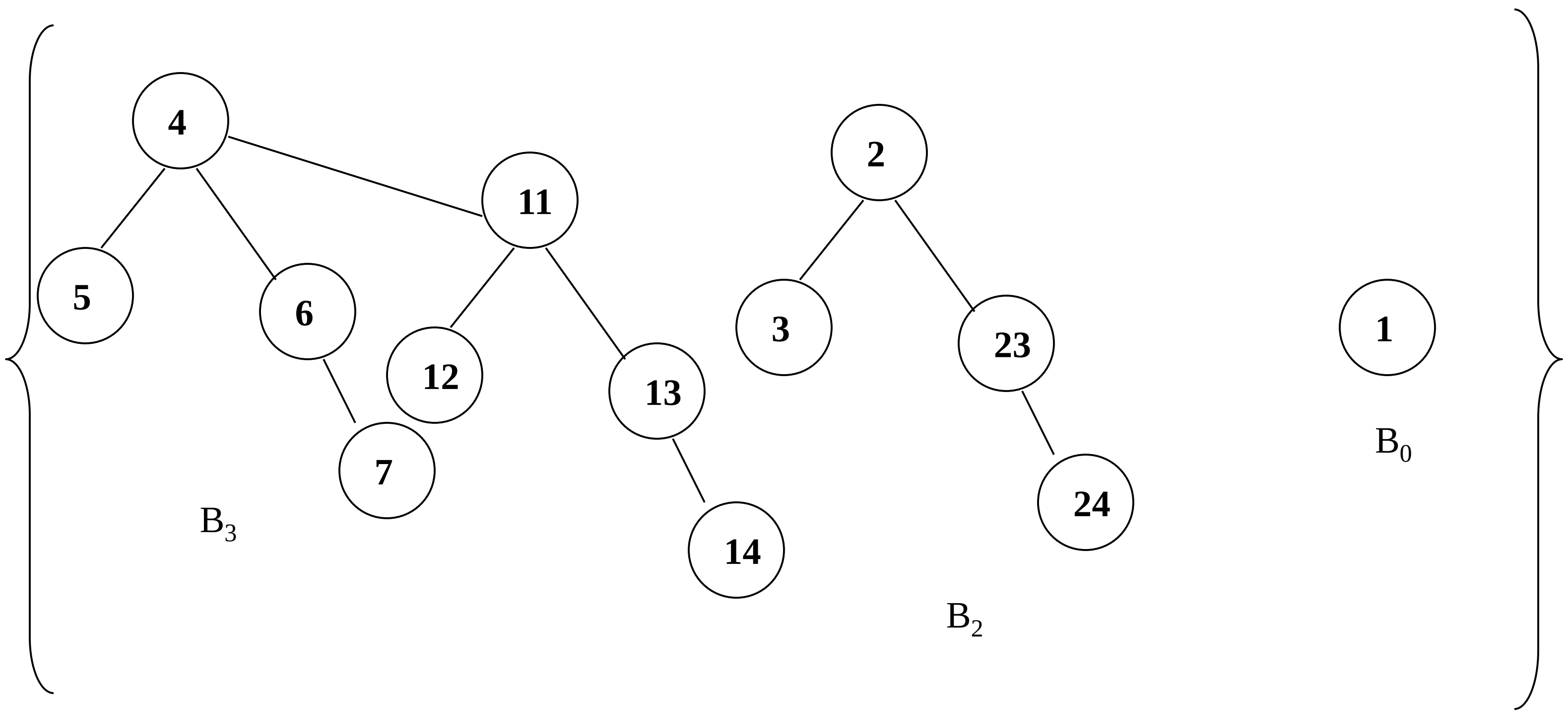
BQ Merge



BQ Merge



BQ Merge



BQ operations

- Insert (x, BQ)
  - Merge (BQ’ containing a B_0 tree, BQ)
  - Also O(logn) worst case

BQ operations

- Extract-Min(BQ) – O(logn) worst case
  - Scan roots of all BiTs in BQ to find the minimum root – O(logn) worst case
  - Delete the root and eventually return the data – O(1)
  - BQ1=BQ containing all the child subtrees of the deleted root
  - BQ2=BQ containing all the other BiTs from BQ
  - Merge BQ1 and BQ2 – O(logn) worst case