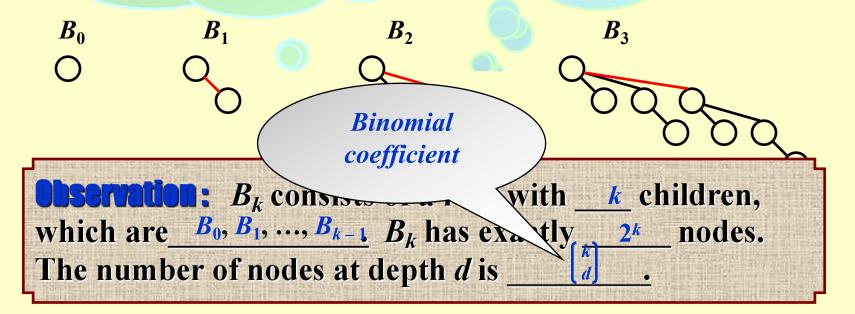
Binomial Queue

Structure:

A binomial queue is not a heap-ordered tree, but rather a collection of heap-ordered trees, known as a forest. Each heap-ordered tree is a binomial tree.

Constant!

So $O(\log N)$ for an insertion is A binomial tree of height 0 is a one-node tree. A binomial trisethe are treight k is formed by attaching a binomial tree, B_{k-1} , to the root of another binomial tree, B_{k-1} .

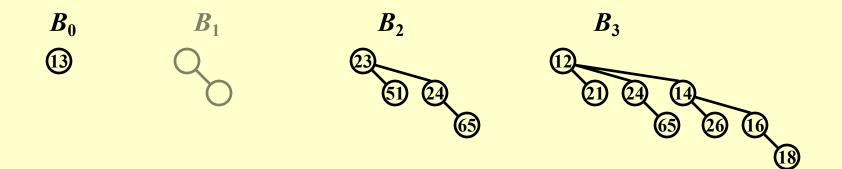


 B_k structure + heap order + one binomial tree for each height

A priority queue of any size can be uniquely represented by a collection of binomial trees.

Example Represent a priority queue of size 13 by a collection of binomial trees.

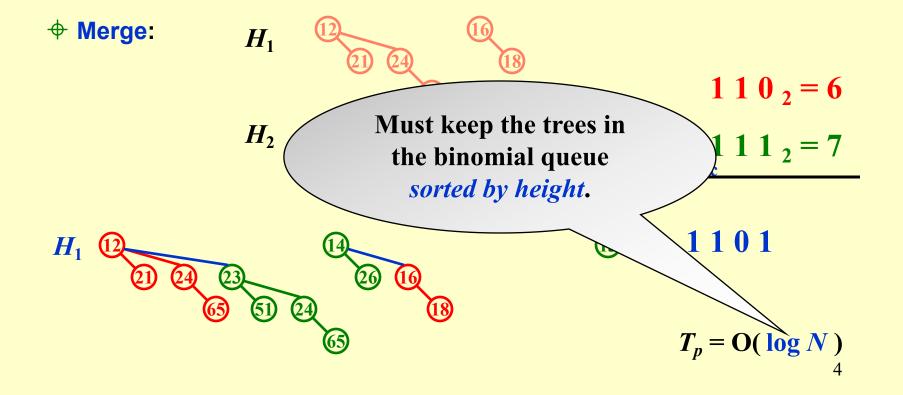
Solution: $13 = 2^0 + 0 \times 2^1 + 2^2 + 2^3 = 1101_2$



Operations:

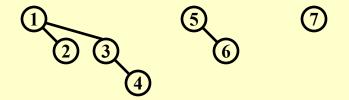
 Φ FindMin: The minimum key is in one of the roots. There are at most $\lceil \log N \rceil$ roots, hence $T_p = O(\log N)$.

Note: We can remember the minimum and update whenever it is changed. Then this operation will take O(1).



+ Insert: a special case for merging.

[Example] Insert 1, 2, 3, 4, 5, 6, 7 into an initially empty queue.



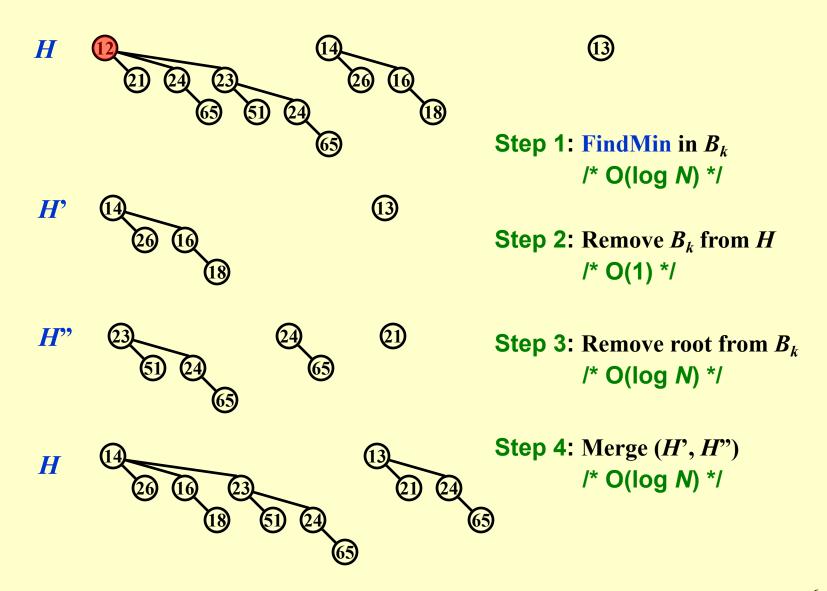
Note:

If the smallest nonexistent binomial tree is B_i , then

 $T_p = Const \cdot (i+1).$

Performing N Inserts on an initially empty binomial queue will take O(N) worst-case time. Hence the average time is constant.

DeleteMin (H):



Implementation:

Binomial queue = array of binomial trees

Operation	Property	Solution
DeleteMin	Find all the subtrees quickly	
Merge	The children are ordered by their sizes	
(4) (4) (4) (5) (5) (5) (5)		

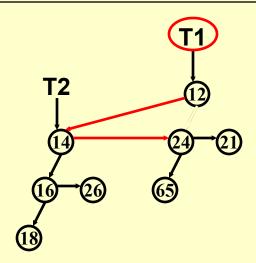
Discussion 7:

How can we implement the trees so that all the subtrees can be accessed quickly?

Discussion 8:

In which order must we link the subtrees?

```
typedef struct BinNode *Position;
typedef struct Collection *BinQueue;
typedef struct BinNode *BinTree; /* missing from p.176 */
struct BinNode
  ElementType
                  Element;
                   LeftChild;
  Position
                   NextSibling;
  Position
struct Collection
                CurrentSize; /* total number of nodes */
  int
  BinTree
                TheTrees[ MaxTrees ];
};
```



```
BinQueue Merge(BinQueue H1, BinQueue H2)
  BinTree T1, T2, Carry = NULL;
  int i, j;
  if (H1->CurrentSize + H2-> CurrentSize > Capacity ) ErrorMessage();
  H1->CurrentSize += H2-> CurrentSize;
  for ( i=0, j=1; j<= H1->CurrentSize; i++, j*=2 ) {
     T1 = H1->TheTrees[i]; T2 = H2->TheTrees[i]; /*current trees */
    case 0: /* 000 */
                                                   T2
                                                          T1
                                             Carry
        case 1: /* 001 */ break:
        case 2: /* 010 */ H1->TheTrees[i] = T2; H2->TheTrees[i] = NULL; break;
        case 4: /* 100 */ H1->TheTrees[i] = Carry; Carry = NULL; break;
        case 3: /* 011 */ Carry = CombineTrees(T1, T2);
                        H1->TheTrees[i] = H2->TheTrees[i] = NULL; break;
        case 5: /* 101 */ Carry = CombineTrees( T1, Carry );
                        H1->TheTrees[i] = NULL; break;
        case 6: /* 110 */ Carry = CombineTrees( T2, Carry );
                        H2->TheTrees[i] = NULL; break;
        case 7: /* 111 */ H1->TheTrees[i] = Carry;
                        Carry = CombineTrees( T1, T2 );
                        H2->TheTrees[i] = NULL; break;
    } /* end switch */
  } /* end for-loop */
  return H1;
```

```
ElementType DeleteMin(BinQueue H)
   BinQueue DeletedQueue;
   Position DeletedTree, OldRoot;
   ElementType MinItem = Infinity; /* the minimum item to be returned */
   int i, j, MinTree; /* MinTree is the index of the tree with the minimum item */
   if (IsEmpty(H)) { PrintErrorMessage(); return –Infinity; }
   for (i = 0; i < MaxTrees; i++) { /* Step 1: find the minimum item */
     if( H->TheTrees[i] && TheTrees[i]->Element < MinItem ) {
                                                            _____<u>/* end</u> if */
          MinItem = H->TheTree>
   } /* end for-i-loop */
                                        This can be replaced by
   DeletedTree = H->TheTrees/
                                      the actual number of roots
                                                                             H' */
   H->TheTrees[ MinTree ] = Nb.
   OldRoot = DeletedTree; /* Step 3.1...
   DeletedTree = DeletedTree->LeftChild; free(OldRoot);
   DeletedQueue = Initialize(); /* Step 3.2: create H" */
   DeletedQueue->CurrentSize = (1<<MinTree) - 1; /* 2MinTree - 1 */
   for (j = MinTree - 1; j >= 0; j - -) {
     DeletedQueue->TheTrees[j] = DeletedTree;
     DeletedTree = DeletedTree->NextSibling;
     DeletedQueue->TheTrees[i]->NextSibling = NULL;
   } /* end for-j-loop */
   H->CurrentSize -= DeletedQueue->CurrentSize + 1;
   H = Merge(H, DeletedQueue); /* Step 4: merge H' and H" */
   return MinItem;
```

【Claim】 A binomial queue of N elements can be built by N successive insertions in O(N) time.

Proof 1 (Aggregate):

Proof 2: An insertion that costs c units results in a net increase of 2-c trees in the forest.

 $C_i := \text{cost of the } i \text{th insertion}$

 $\Phi_i ::=$ number of trees *after* the *i*th insertion ($\Phi_0 = 0$)

$$C_i + (\Phi_i - \Phi_{i-1}) = 2$$
 for all $i = 1, 2, ..., N$

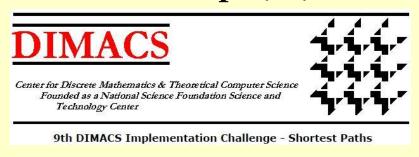
Add all these equations up $\sum_{i=1}^{N} C_i + \Phi_N - \Phi_0 = 2N$

$$\sum_{i=1}^{N} C_i = 2N - \Phi_N \le 2N = O(N)$$

$$T_{worst} = O(\log N)$$
, but $T_{amortized} = 2$



Research Project 1 Shortest Path Algorithm with Heaps (26)



This project requires you to implement Dijkstra's algorithm based on a min-priority queue, such as a *Fibonacci heap*. The goal of the project is to find the best data structure for Dijkstra's algorithm.

Detailed requirements can be downloaded from

https://pintia.cn/

Reference:

Data Structure and Algorithm Analysis in C (2nd Edition): Ch.5, p.170-180; Ch.11, p.430-435; M.A.Weiss 著、陈越改编,人民邮件出版社,2005

Introduction to Algorithms, 3rd Edition: Ch.19, p. 505-530; Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009