

Data Structures – Week #8

Heaps (Priority Queues)

Outline

- Motivation for Heaps
- Implementation Alternatives of PQs
- Binary Heaps
- Basic Heap Operations (Insert, Delete Min)
- Other Heap Operation
 - BuildHeap, DecreaseKey, IncreaseKey, Delete
- d-Heaps
- Leftist Heaps
- Binomial Heaps

Motivation for Heaps

- Priority queues are queues in which jobs with *different priorities* are enqueued and handled accordingly.
- Heaps are data structures that are used to implement priority queues.
- Heaps can be represented in an array since a complete binary tree is very regular.
- Two basic operations
 - Insert (average $O(1)$, worst case $O(\log n)$), and
 - DeleteMin ($O(\log n)$).

March 10, 2021

Borahan Tümer, Ph.D.

3

Implementation Issues

Implementation	Insertion	DeleteMin
Array (Cmplt BT)	$O(1)$	$O(\log(n))$
Linked List	$O(1)$	$O(n)$
Linked List (sorted)	$O(n)$	$O(1)$
BST	$O(\log(n))$	$O(\log(n))$

March 10, 2021

Borahan Tümer, Ph.D.

4

Keys in Heaps

- In the scope of this class (CSE 225),
 - a *key* in a heap is the *priority value of the corresponding node* which determines the position of its node in the heap;
 - i.e., in a *min-heap* the node with the *minimum key* is the node with the *highest priority*, hence it is the *root*.

March 10, 2021

Borahan Tümer, Ph.D.

5

Binary Heaps

- A binary heap is
 - a completely filled binary tree with the possible exception of the bottom level, (known as a *complete binary tree*)
 - filled from *left to right*
 - with two properties:
 - *Structure property*
 - *Heap order property*

March 10, 2021

Borahan Tümer, Ph.D.

6

Structure Property

If a complete binary tree is represented in an array, then for *any element* in array position i , the *left child* is in position $2i$ and the *right child* in $2i+1$ iff they exist (i.e., $2i < n$ and $2i+1 < n$, respectively).

March 10, 2021

Borahan Tümer, Ph.D.

7

(Min)Heap Order Property

In a heap, *for every node X , the key in the parent of X is smaller than (or equal to) the key in X* , with the exception of the root since it has no parent. (Depending on the application, the opposite of this may be considered as the heap order property, too!!!)

March 10, 2021

Borahan Tümer, Ph.D.

8

Insertion

- **Steps of Insertion operation**
 - Create a hole in the next available location;
 - If heap order property is not violated
 - Then we are done;
 - Else
 - loop
 - » exchange the hole with the parent node
 - until the heap property is restored (i.e., *percolate the hole up*)
- Worst time best upper bound: $O(\log(n))$

March 10, 2021

Borahan Tümer, Ph.D.

9

Insert Function

```
void insert(ElmntType x, PrQ h)
{ // array starts at cell #1 not #0
  int i;
  if (isFull(h)){
    display("queue full: unsuccessful insertion");
    return;
  }
  for (i = ++h->Size; h->elements[i/2] > x; i/=2 )
    h->elements[i] = h->elements[i/2];
  h->elements[i] = x;
}
```

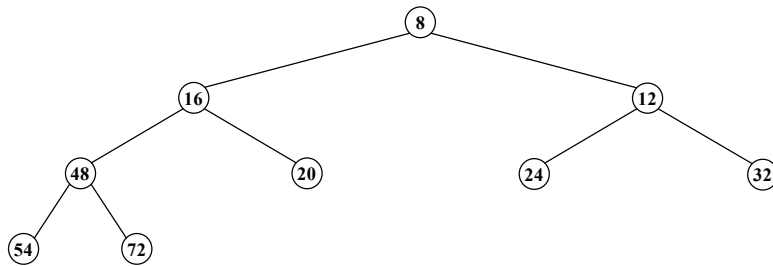
March 10, 2021

Borahan Tümer, Ph.D.

10

Animated Insertion Example – Current Heap

48 16 24 20 8 12 32 54 72



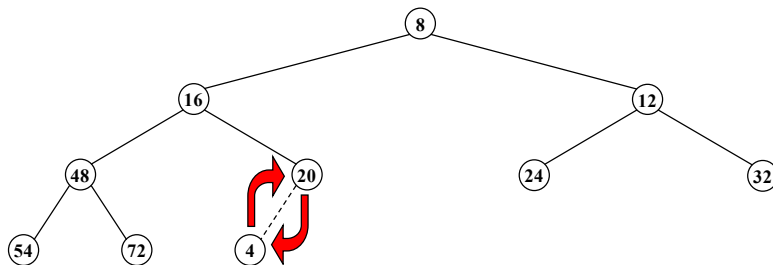
March 10, 2021

Borahan Tümer, Ph.D.

11

Inserting 4...

48 16 24 20 8 12 32 54 72 4



4 < 20

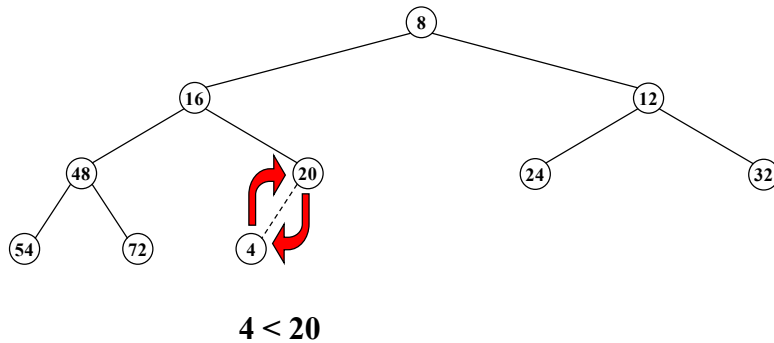
March 10, 2021

Borahan Tümer, Ph.D.

12

Inserting 4...

48 16 24 20 8 12 32 54 72 4



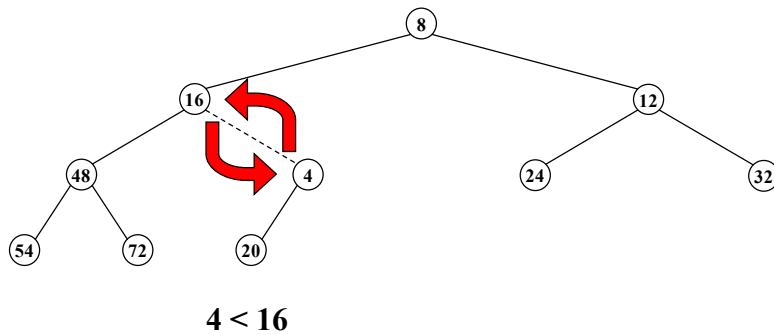
March 10, 2021

Borahan Tümer, Ph.D.

13

Inserting 4...

48 16 24 20 8 12 32 54 72 4



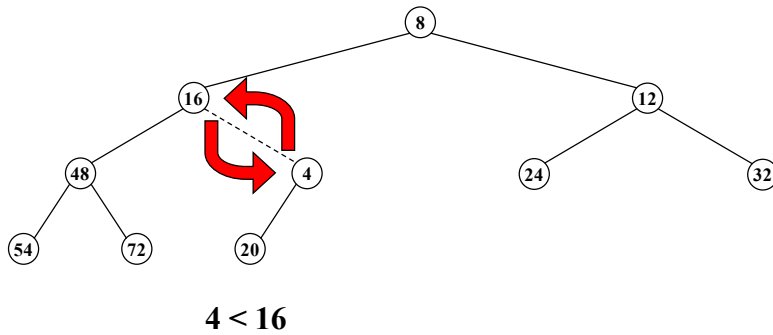
March 10, 2021

Borahan Tümer, Ph.D.

14

Inserting 4...

48 16 24 20 8 12 32 54 72 4



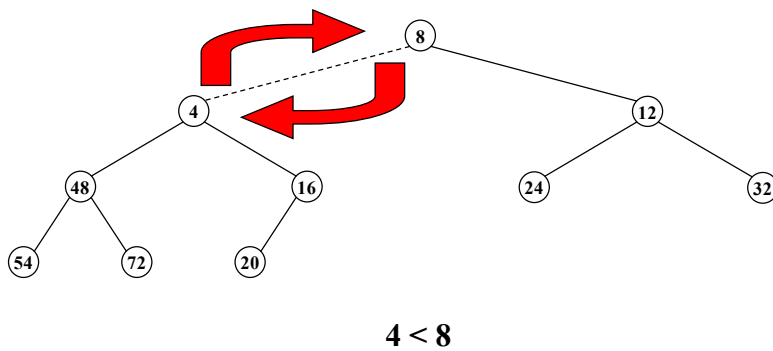
March 10, 2021

Borahan Tümer, Ph.D.

15

Inserting 4...

48 16 24 20 8 12 32 54 72 4



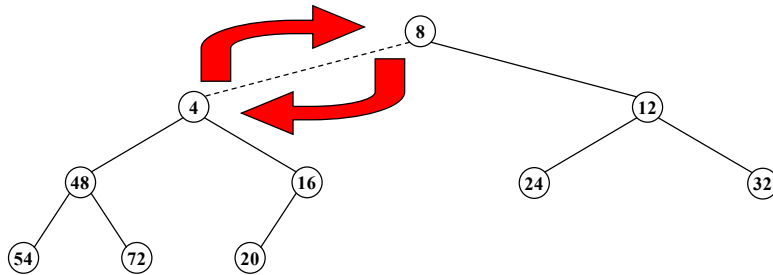
March 10, 2021

Borahan Tümer, Ph.D.

16

Inserting 4...

48 16 24 20 8 12 32 54 72 4



$$4 < 8$$

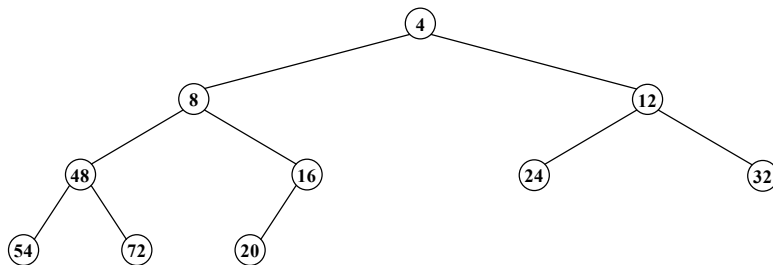
March 10, 2021

Borahan Tümer, Ph.D.

17

Inserting 4...

48 16 24 20 8 12 32 54 72 4



March 10, 2021

Borahan Tümer, Ph.D.

18

DeleteMin Operation

• Steps of DeleteMin operation

- Remove the minimum element (at the root) from the heap;
- If the last element can be placed in the hole
 - Then we are done;
 - Else
 - Loop
 - » exchange the hole with the smaller child node
 - until the last element moves in the heap (i.e., *percolate the hole down*).

March 10, 2021

Borahan Tümer, Ph.D.

19

DeleteMin Operation

```

ElmntType DeleteMin(PrQ h)
{
    int i, chld;
    ElmntType minelm, lastelm;
    if isEmpty(h){
        display("queue empty")
        return (h->elements[0]);
    }
    minelm=h->elements[1];
    lastelm=h->elements[h->size--];

```

... Cont'd at the next page!

March 10, 2021

Borahan Tümer, Ph.D.

20

DeleteMin Operation... (cont'd)

```

for (i=1; i * 2 <= h->size; i=chld) {
    // find smaller child
    chld=i*2;
    if (chld != h->size && h->elements[chld+1] < h->elements[chld]) chld++;
    // percolate one level
    if (lastelm > h->elements[chld])
        h->elements[i] = h->elements[chld];
    else break;
}
h->elements[i] = lastelm;
//restore min-heap property in case it is violated by placing lastelm to heap's ith node
for (j = i; h->elements[j/2] > lastelm; j/=2)
    h->elements[j] = h->elements[j/2];
h->elements[j] = lastelm;
return minelm;
}

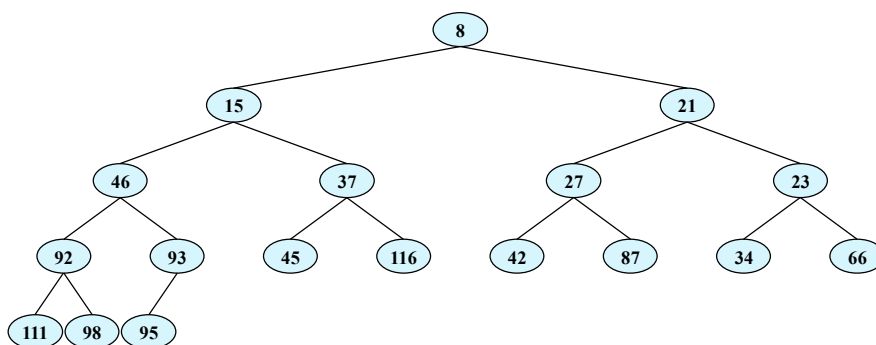
```

March 10, 2021

Borahan Tümer, Ph.D.

21

DeleteMin Operation

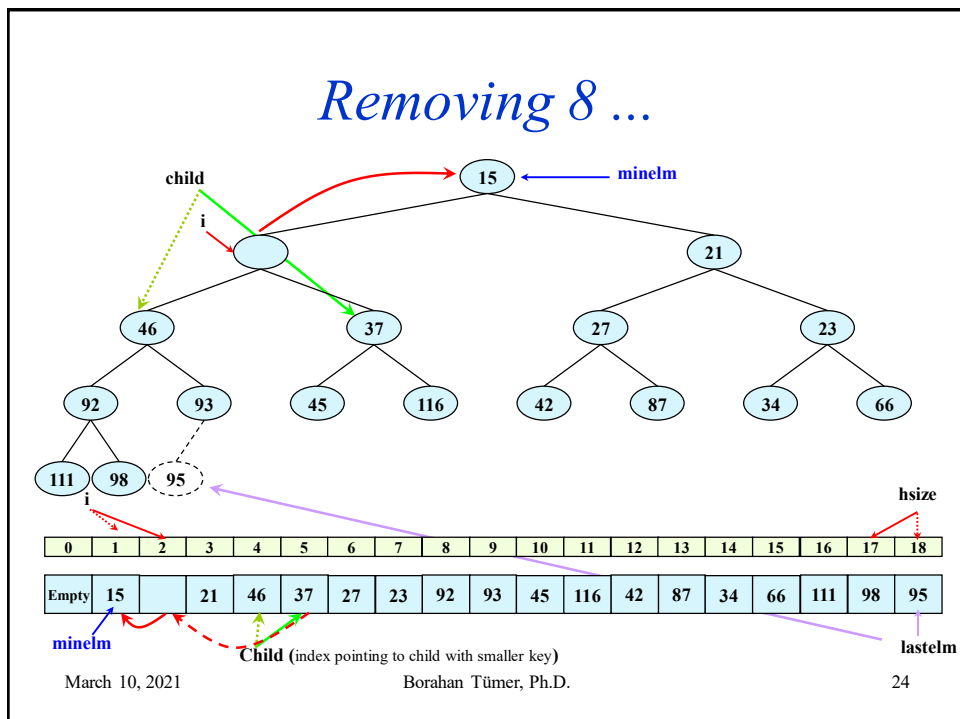
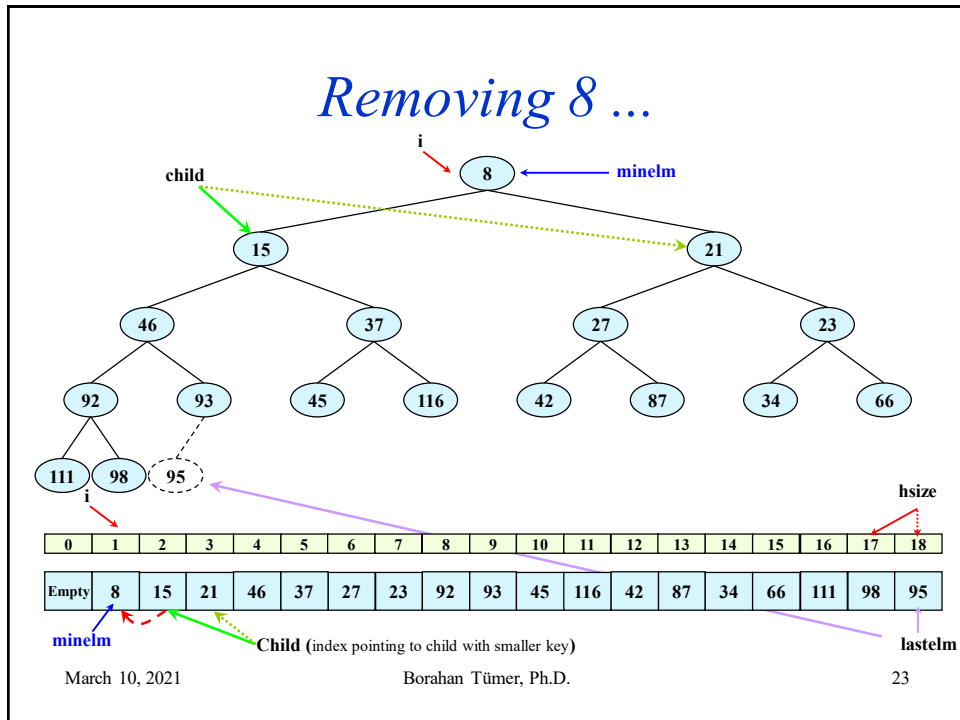


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Empty	8	15	21	46	37	27	23	92	106	45	116	42	87	34	66	111	98	95

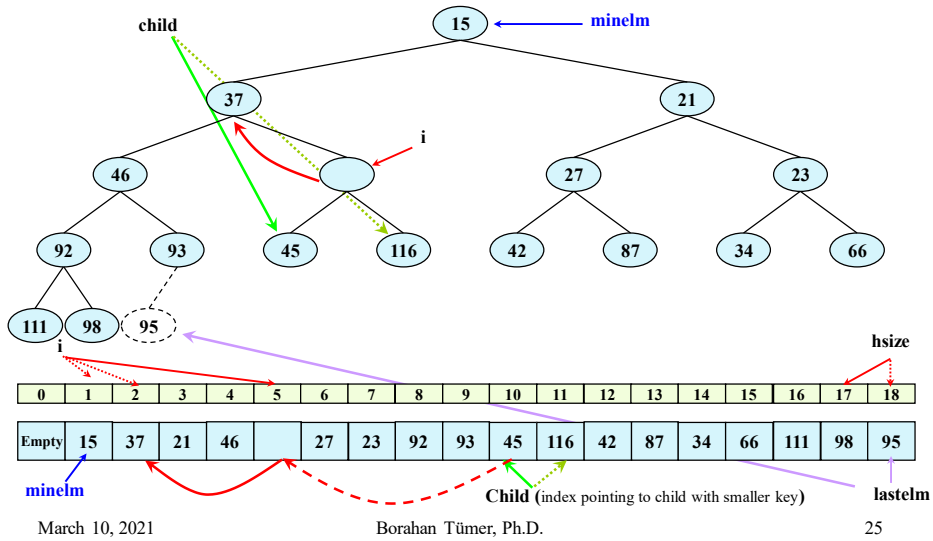
March 10, 2021

Borahan Tümer, Ph.D.

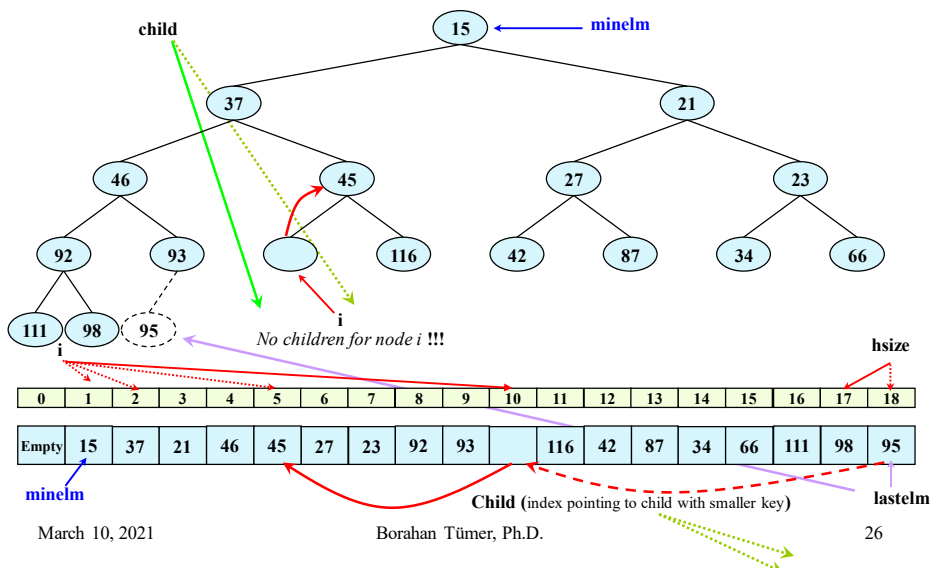
22



Removing 8 ...



Removing 8 ...



Other Heap Operations

- **Other Heap Operations**
 - DecreaseKey(I,D,HeapHeader);
 - IncreaseKey(I,D,HeapHeader);
 - Delete(I,HeapHeader);
 - BuildHeap(HeapHeader);

March 10, 2021

Borahan Tümer, Ph.D.

29

DecreaseKey and IncreaseKey

- **DecreaseKey (I,D,HeapHeader) & IncreaseKey (I,D,HeapHeader)**
- These two operations decrease or increase the key at position i of the heap the root of which is pointed to by the HeapHeader by the amount D, respectively.
- Any of these two operations may violate the heap order. By percolating the modified key up or down after the DecreaseKey operation or IncreaseKey operation, respectively, the heap order property may be restored.

March 10, 2021

Borahan Tümer, Ph.D.

30

Removal of any Key

- **Delete (I,HeapHeader)**
- This operation removes any key in the heap structure.
- Again, after the removal of the element from the heap, the heap order property may be violated.
- In this case, we may consider
 - the node of the key removed as the root, and
 - the corresponding subtree as the tree we perform a *DeleteMin* on.
- Then using the *DeleteMin* operation, the heap order property is restored.

March 10, 2021

Borahan Tümer, Ph.D.

31

BuildHeap

- **BuildHeap(HeapHeader)**
- This operation is used to build a heap from a set of input data (e.g., numbers).
- Assuming that a set of numbers are arbitrarily (i.e., with no consideration of the heap order property) placed in a complete binary tree, we build a binary heap in
- For n numbers, this operation can be performed in n successive inserts. Since an insert takes $O(1)$ in average and $O(\log(n))$ worst case, *Buildheap* takes an average time of $n * O(1) = O(n)$ and $n * O(\log(n)) = O(n * \log(n))$ in the worst case.

March 10, 2021

Borahan Tümer, Ph.D.

32

BuildHeap Algorithm

- Starting from
 - the rightmost subtree with a height of 1,
 - Loop
 - compare the children and find the smaller child
 - compare the smaller child with the parent
 - exchange the smaller child with the parent.
 - until all nodes in the tree are processed.
- Subtrees with roots of height greater than 1,
 - the parent must be percolated down until the heap order property is restored.
- An example follows.

March 10, 2021

Borahan Tümer, Ph.D.

33

Constructing a MinHeap – Animation

48

48

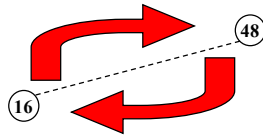
March 10, 2021

Borahan Tümer, Ph.D.

34

Constructing a MinHeap – Animation

48 16



$$48 > 16$$

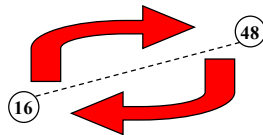
March 10, 2021

Borahan Tümer, Ph.D.

35

Constructing a MinHeap – Animation

48 16



$$48 > 16$$

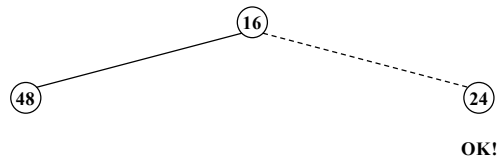
March 10, 2021

Borahan Tümer, Ph.D.

36

Constructing a MinHeap – Animation

48 16 24



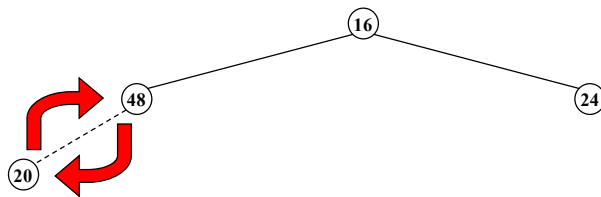
March 10, 2021

Borahan Tümer, Ph.D.

37

Constructing a MinHeap – Animation

48 16 24 20



$48 > 20$

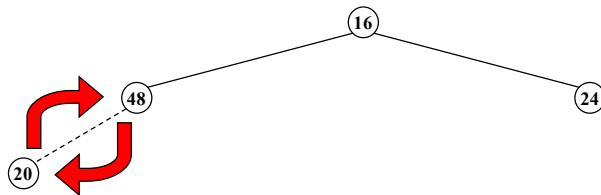
March 10, 2021

Borahan Tümer, Ph.D.

38

Constructing a MinHeap – Animation

48 16 24 20



$$48 > 20$$

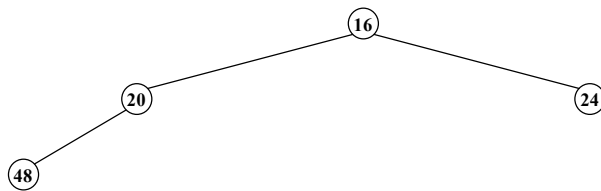
March 10, 2021

Borahan Tümer, Ph.D.

39

Constructing a MinHeap – Animation

48 16 24 20



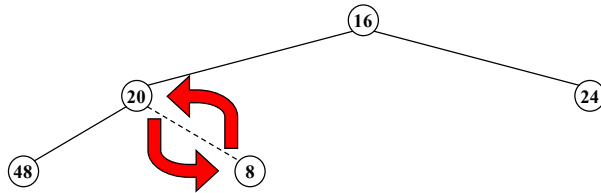
March 10, 2021

Borahan Tümer, Ph.D.

40

Constructing a MinHeap – Animation

48 16 24 20 8



$$20 > 8$$

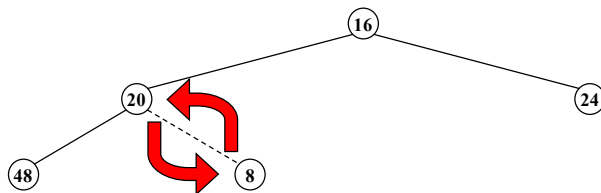
March 10, 2021

Borahan Tümer, Ph.D.

41

Constructing a MinHeap – Animation

48 16 24 20 8



$$20 > 8$$

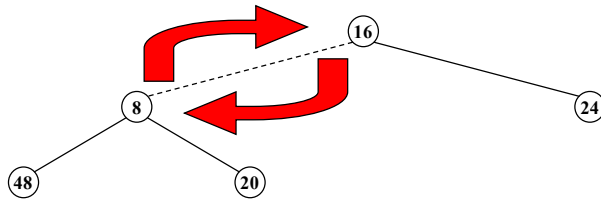
March 10, 2021

Borahan Tümer, Ph.D.

42

Constructing a MinHeap – Animation

48 16 24 20 8



$$16 > 8$$

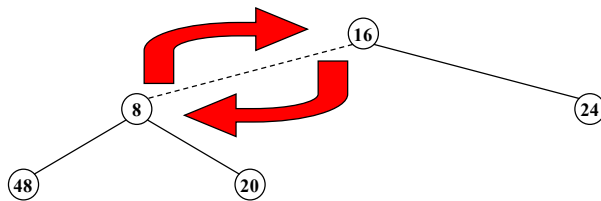
March 10, 2021

Borahan Tümer, Ph.D.

43

Constructing a MinHeap – Animation

48 16 24 20 8



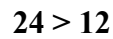
$$16 > 8$$

March 10, 2021

Borahan Tümer, Ph.D.

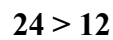
44

48 16 24 20 8 12



45

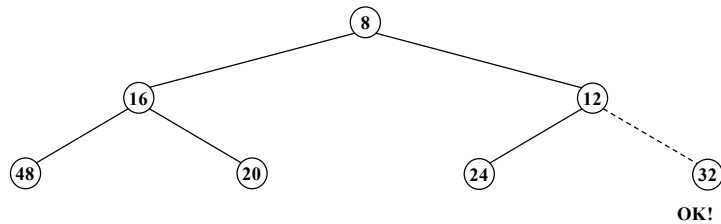
48 16 24 20 8 12



46

Constructing a MinHeap – Animation

48 16 24 20 8 12 32



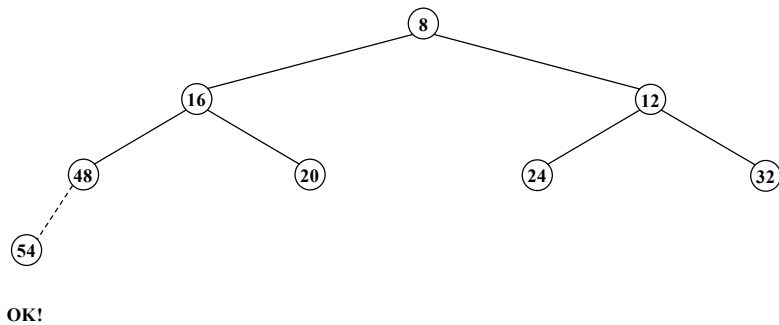
March 10, 2021

Borahan Tümer, Ph.D.

47

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54



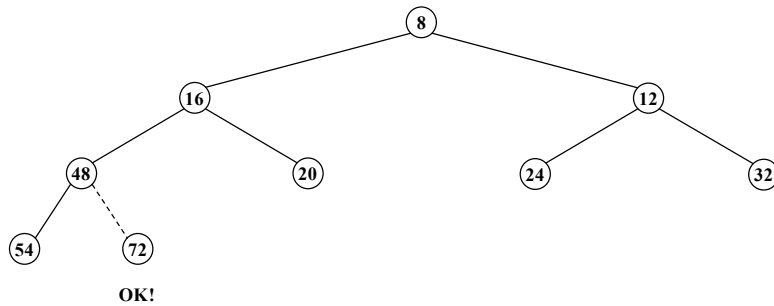
March 10, 2021

Borahan Tümer, Ph.D.

48

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72



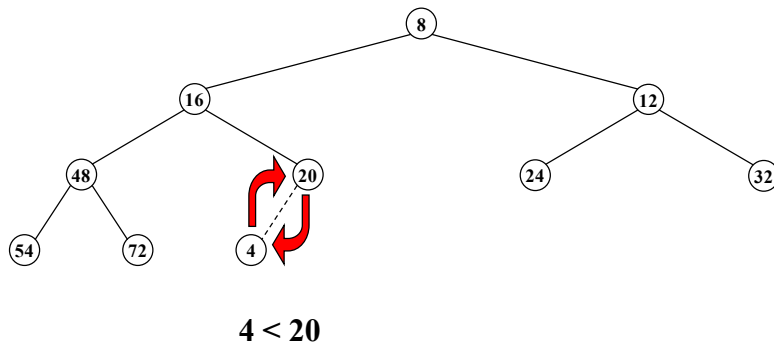
March 10, 2021

Borahan Tümer, Ph.D.

49

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



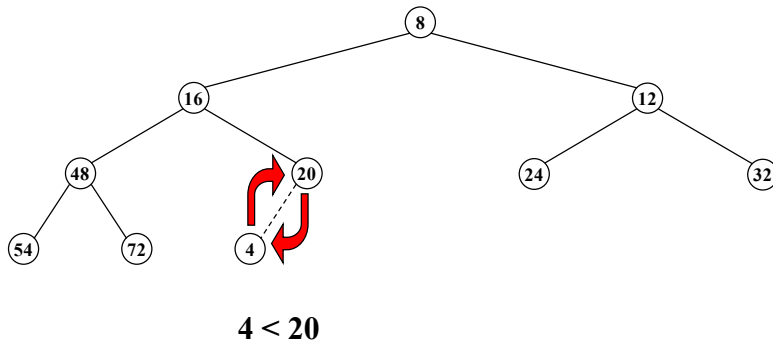
March 10, 2021

Borahan Tümer, Ph.D.

50

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



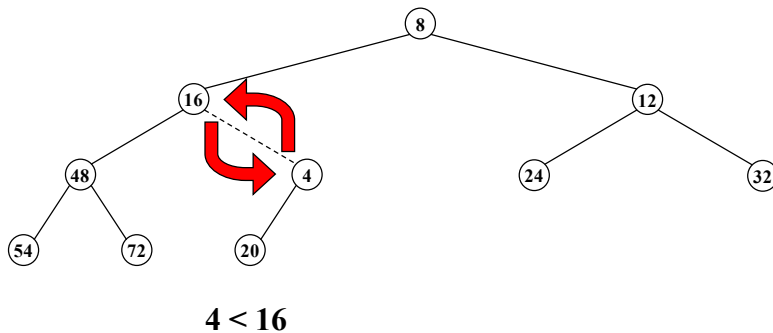
March 10, 2021

Borahan Tümer, Ph.D.

51

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



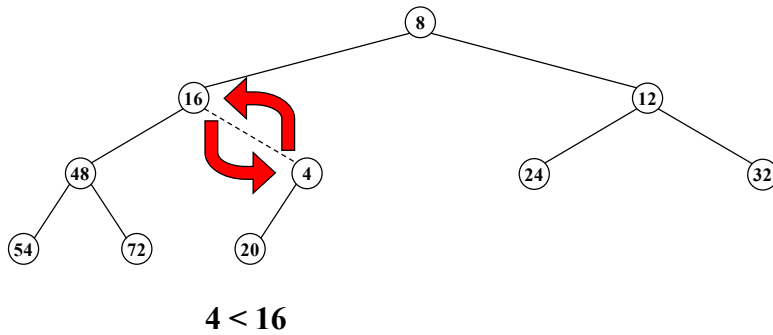
March 10, 2021

Borahan Tümer, Ph.D.

52

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



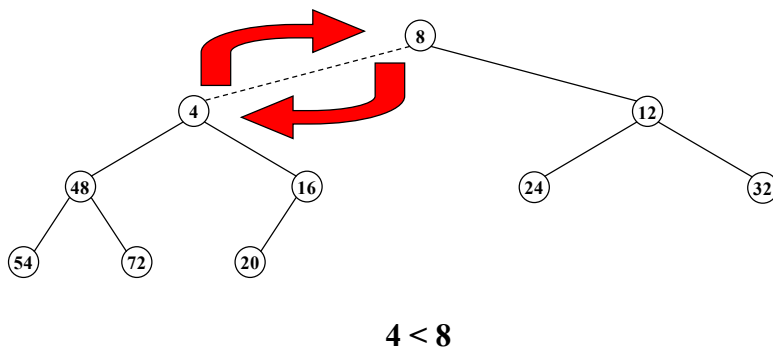
March 10, 2021

Borahan Tümer, Ph.D.

53

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



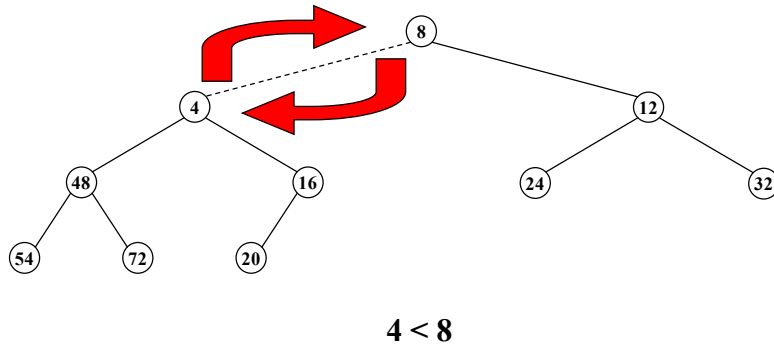
March 10, 2021

Borahan Tümer, Ph.D.

54

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4



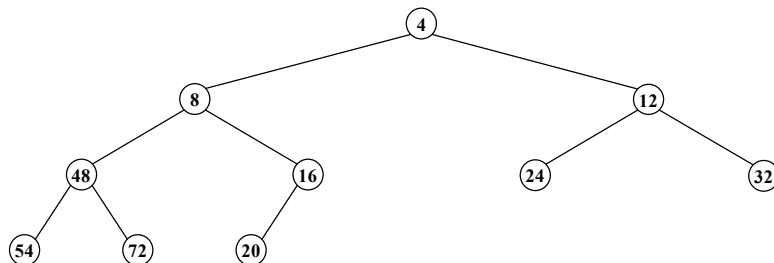
March 10, 2021

Borahan Tümer, Ph.D.

55

Constructing a MinHeap – Animation

48 16 24 20 8 12 32 54 72 4

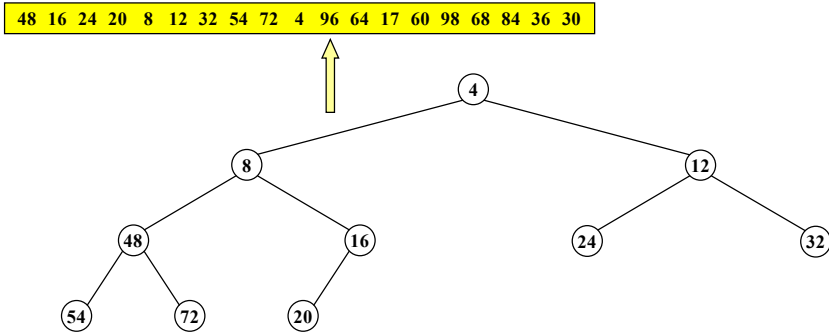


March 10, 2021

Borahan Tümer, Ph.D.

56

Constructing a MinHeap – Classwork

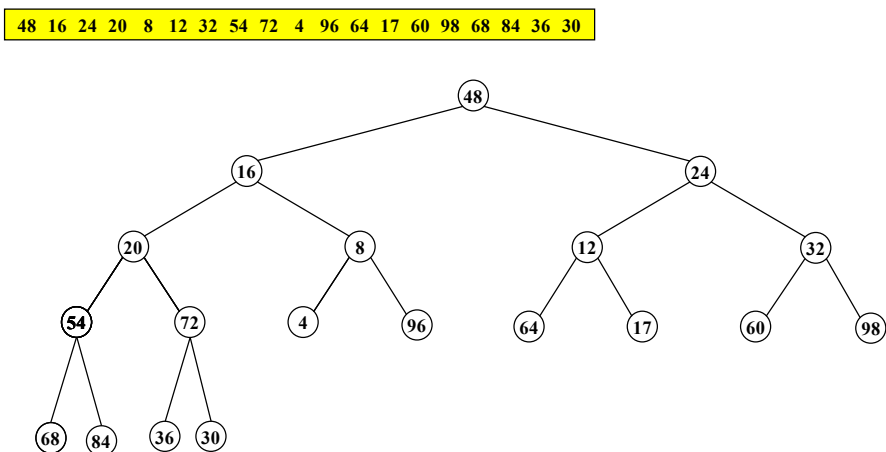


March 10, 2021

Borahan Tümer, Ph.D.

57

Constructing a MinHeap



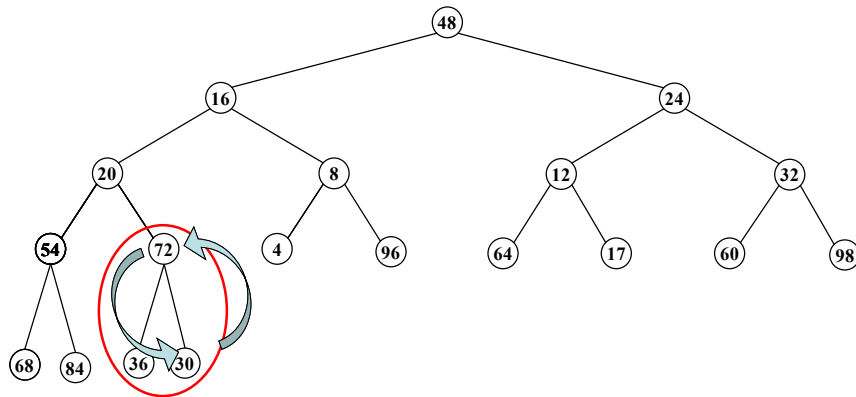
March 10, 2021

Borahan Tümer, Ph.D.

58

Constructing a MinHeap

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



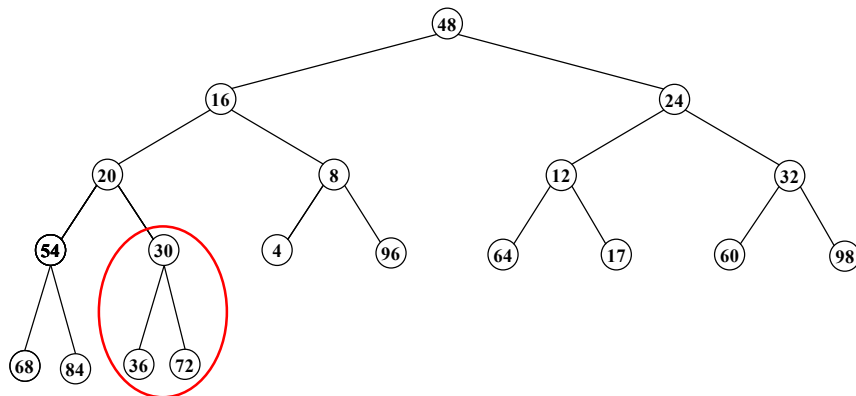
March 10, 2021

Borahan Tümer, Ph.D.

59

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



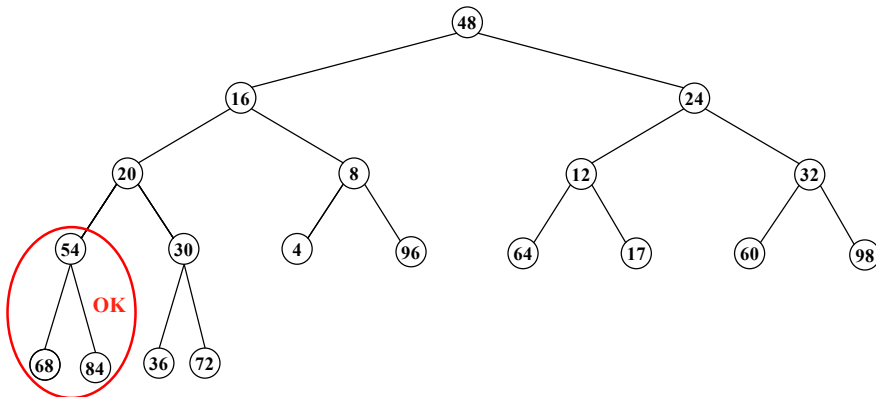
March 10, 2021

Borahan Tümer, Ph.D.

60

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



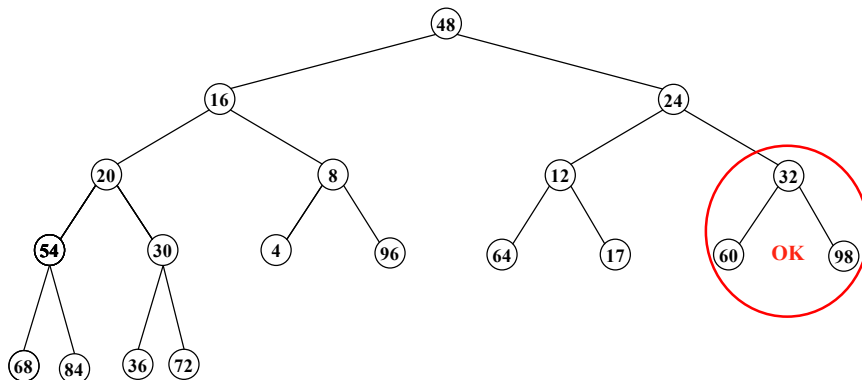
March 10, 2021

Borahan Tümer, Ph.D.

61

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



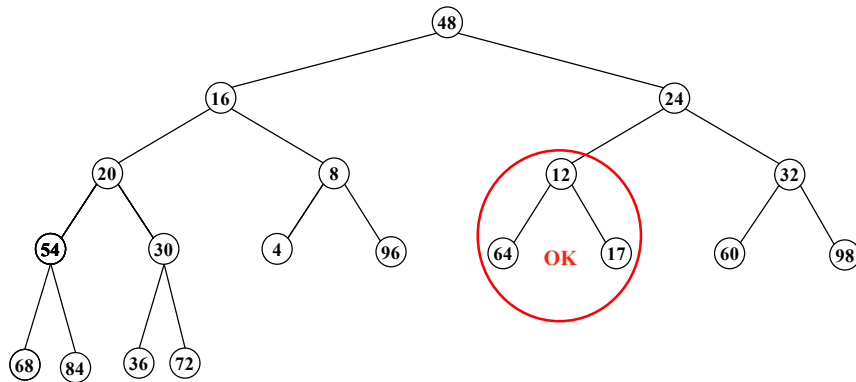
March 10, 2021

Borahan Tümer, Ph.D.

62

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



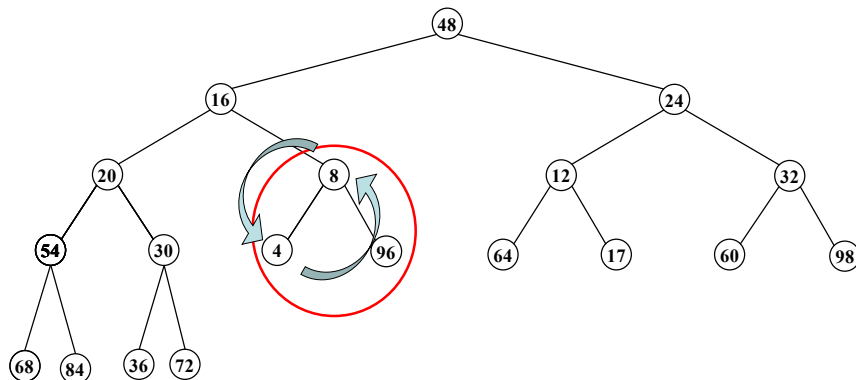
March 10, 2021

Borahan Tümer, Ph.D.

63

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



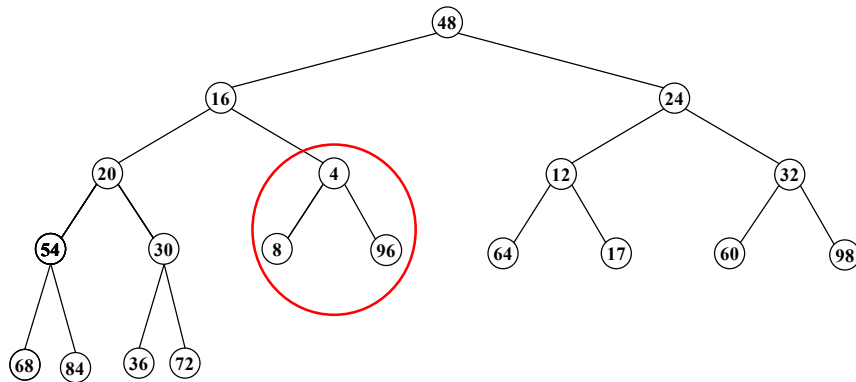
March 10, 2021

Borahan Tümer, Ph.D.

64

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



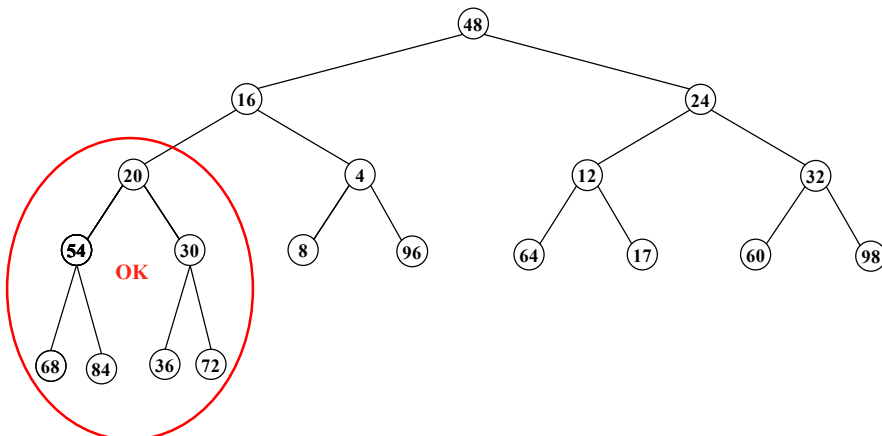
March 10, 2021

Borahan Tümer, Ph.D.

65

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



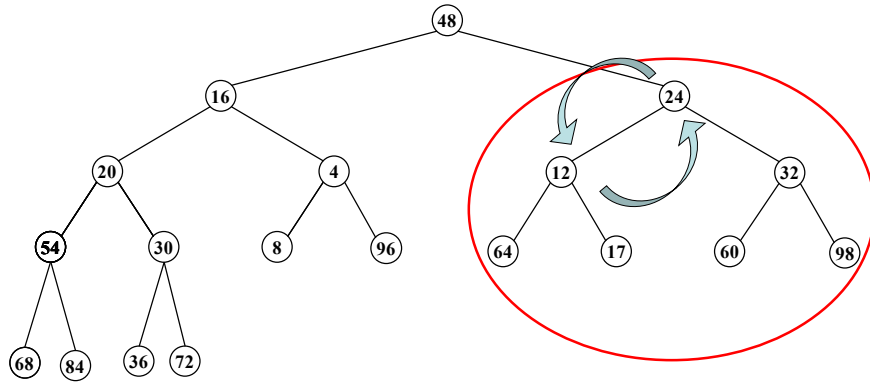
March 10, 2021

Borahan Tümer, Ph.D.

66

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



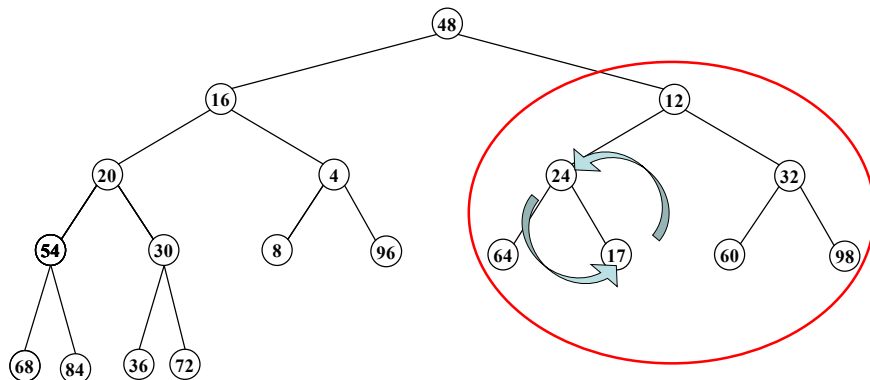
March 10, 2021

Borahan Tümer, Ph.D.

67

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



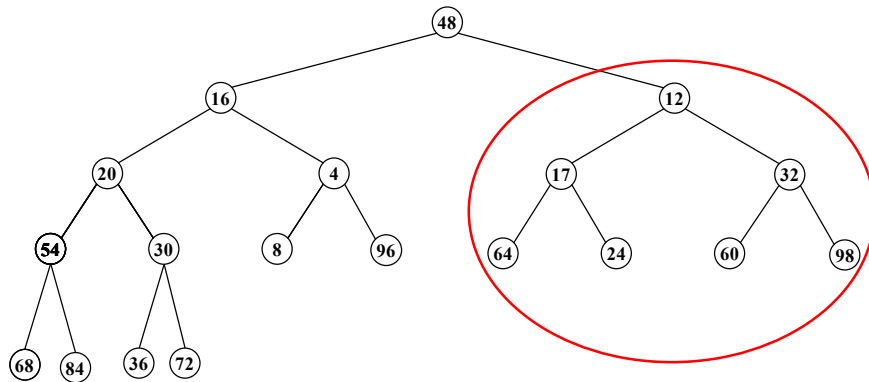
March 10, 2021

Borahan Tümer, Ph.D.

68

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



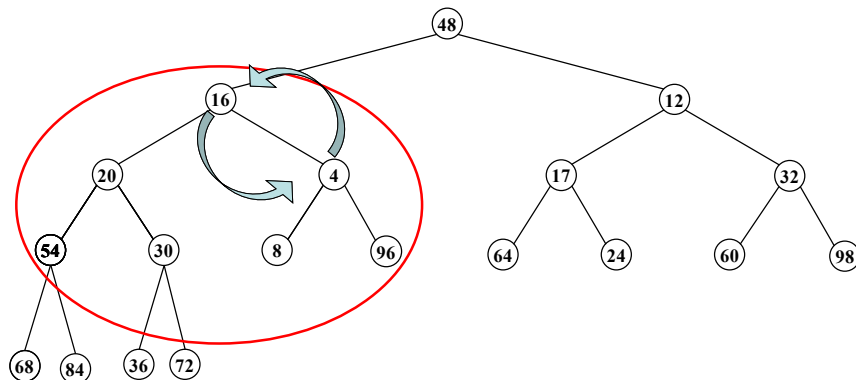
March 10, 2021

Borahan Tümer, Ph.D.

69

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



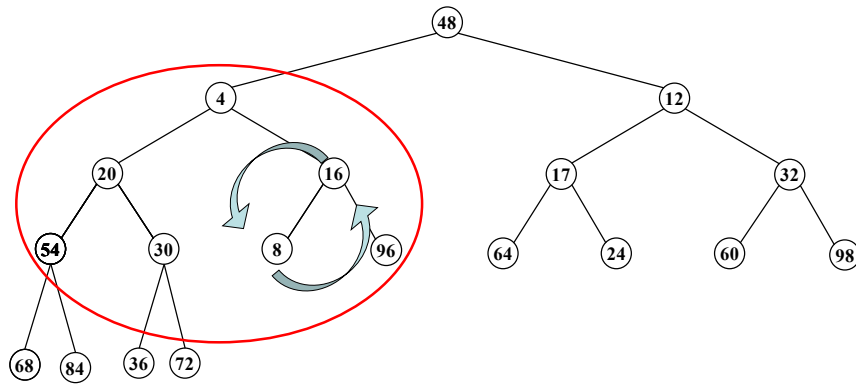
March 10, 2021

Borahan Tümer, Ph.D.

70

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



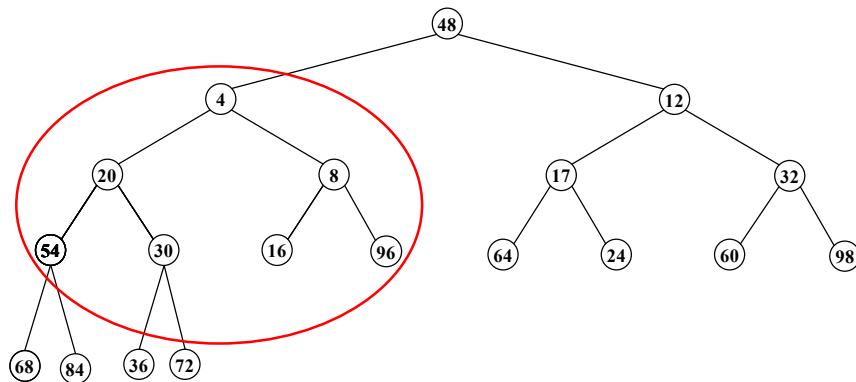
March 10, 2021

Borahan Tümer, Ph.D.

71

Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



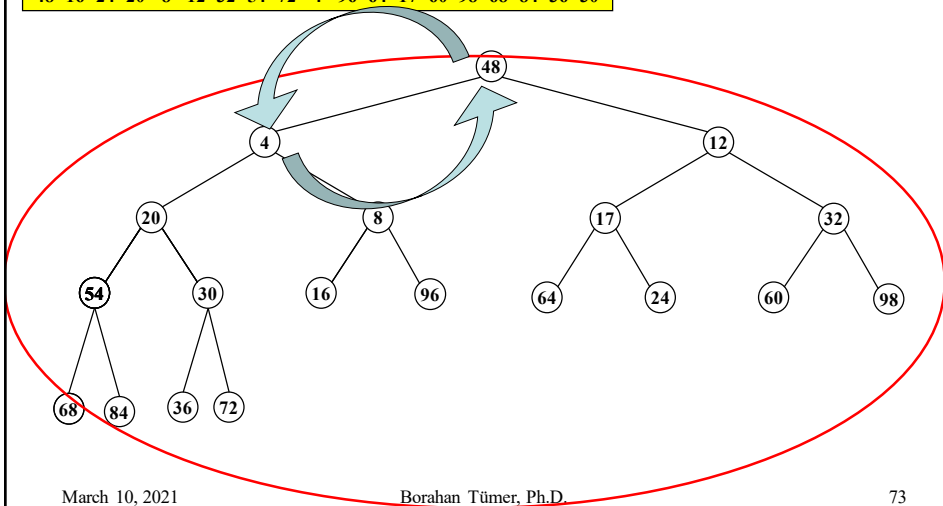
March 10, 2021

Borahan Tümer, Ph.D.

72

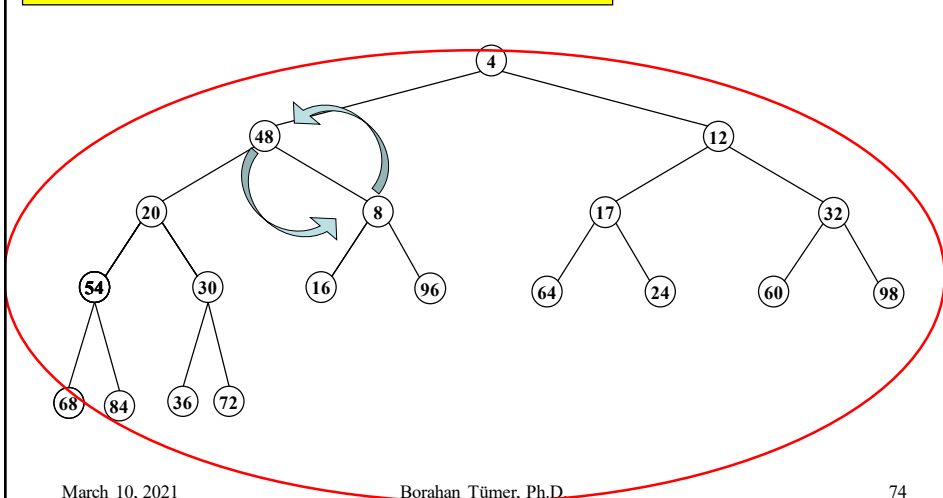
Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



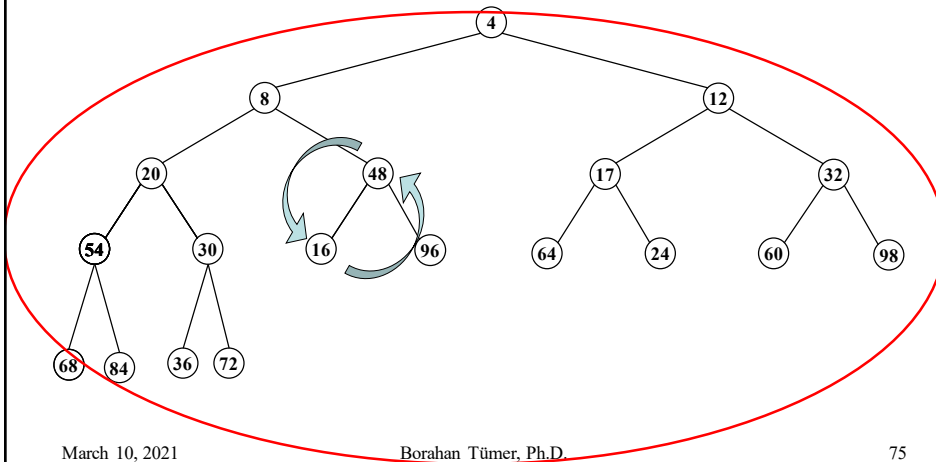
Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



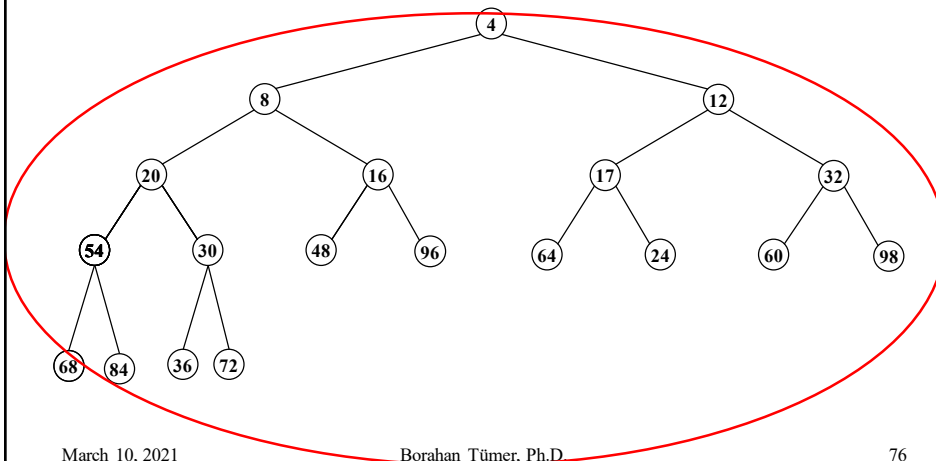
Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



Constructing a MinHeap -

48 16 24 20 8 12 32 54 72 4 96 64 17 60 98 68 84 36 30



d-Heaps

- **d-Heaps**
- A simple generalization to binary heaps is a d-heap,
- which is exactly like a binary heap except that all nodes have d children (i.e., a binary heap is a 2-heap).

March 10, 2021

Borahan Tümer, Ph.D.

77

Leftist Heaps

March 10, 2021

Borahan Tümer, Ph.D.

78

Motivation for Leftist Heaps

- Leftist heaps make *merging* possible in $O(\log n)$ time (*$\log n$ insertions* each with $O(1)$ average time) using only an array as in binary heaps.
- LHs have both
 - a *structural* property, and
 - an *ordering* property.
- A LH has the *same heap order property*.
- *A LH is a binary tree.*
- Difference between a LH and a binary heap is
 - a *LH is not perfectly balanced.*

March 10, 2021

Borahan Tümer, Ph.D.

79

Leftist Heap Property

- Definition:
 - *Null path length* of a node X , $Npl(X)$, is defined as the *length of the shortest path from X to a node without two children.*
- By definition, $Npl(NULL) = -1$.
- $Npl(\text{leaf}) = 0$.
- LH property is that for every node X in the heap,
 - *$Npl(LCX) \geq Npl(RCX)$*
 - where LCX and RCX denote the left child and the right child of node X , respectively.

March 10, 2021

Borahan Tümer, Ph.D.

80

Leftist Heap Operations (*Merging*)

- The fundamental operation is *merging*.
- Two solutions
 - *recursive* version
 - *non-recursive* version
- Check and make sure that both binary trees are actually LHs!

March 10, 2021

Borahan Tümer, Ph.D.

82

Recursive Merging Algorithm

1. Input: two LHs.
2. First check that both binary trees are LHs.
3. If either heap is empty, then the result of the merge is the other heap.
4. If not, then compare the roots
5. Recursively merge the heap with the larger root with right subheap of the heap with the smaller root.

March 10, 2021

Borahan Tümer, Ph.D.

83

Recursive Merging Algorithm

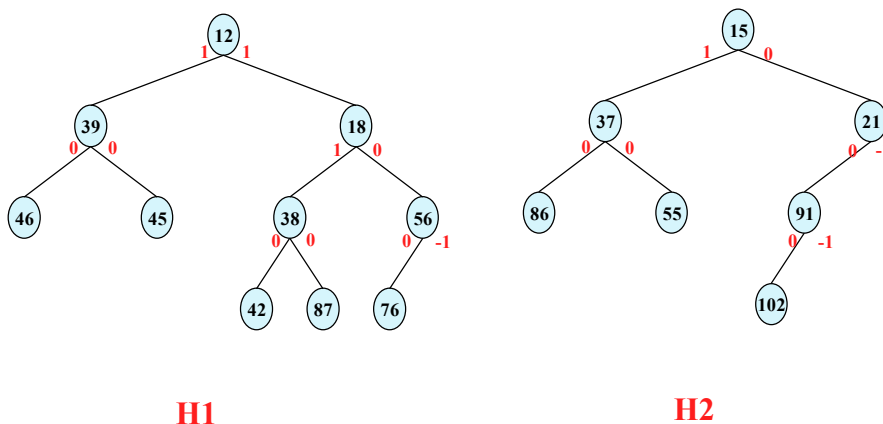
6. The recursive function at Step 5 will invoke itself until the base condition at Step 3 (i.e., that one LH is empty) will be attained. At this point the execution will start returning step by step to the original function call while building up the merged heap starting from the bottom level.
7. At each step, check if the LH property is violated. If so, swap the right child and the left child.
8. After each swap, compute the new $Npl(LH)$ by adding 1 to the $Npl(new\ RC)$
9. End of the recursive algorithm

March 10, 2021

Borahan Tümer, Ph.D.

84

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

85

Recursive Merging Algorithm

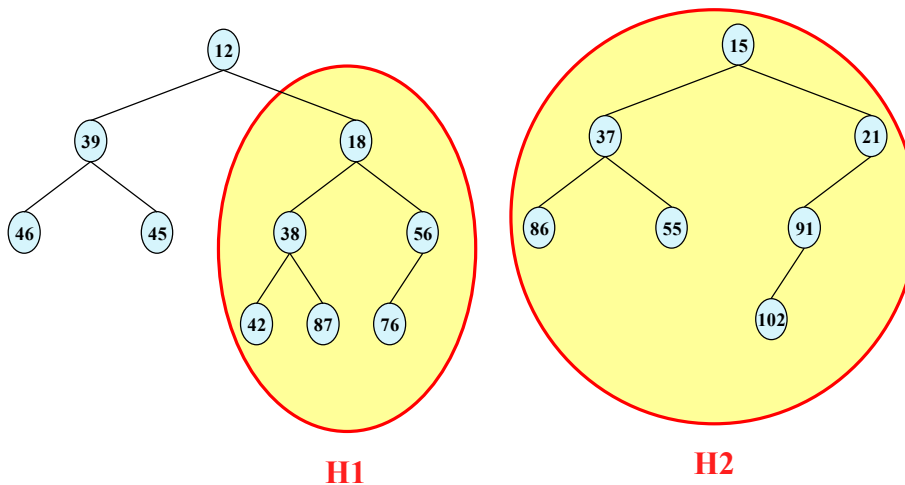
1. Input: two LHs H1 and H2 (on slide 65).
2. Both binary trees are LHs.
3. No heap is empty.
4. $12 < 15$
5. Recursively merge the heap with 15 with right subheap of the heap with 12.

March 10, 2021

Borahan Tümer, Ph.D.

86

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

87

Recursive Merging Algorithm

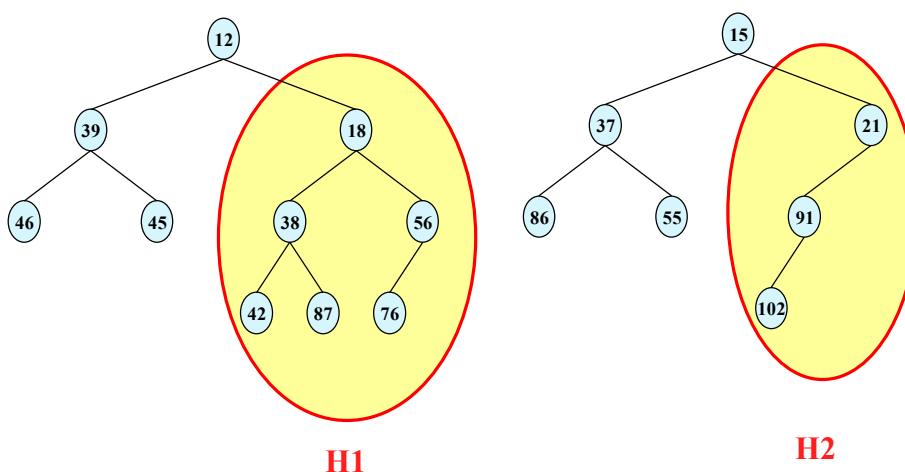
1. Input: two LHs H1 (within the red ellipse) and H2 (see slide 67).
2. Both binary trees are LHs.
3. No heap is empty.
4. $15 < 18$
5. Recursively merge the heap with 18 with right subheap of the heap with 15.

March 10, 2021

Borahan Tümer, Ph.D.

88

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

89

Recursive Merging Algorithm

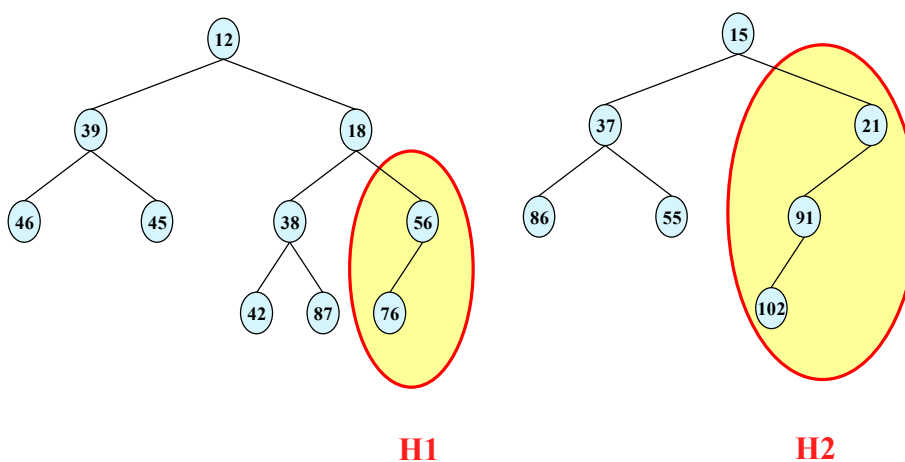
1. Input: two LHs H1 and H2 (both within the red ellipses) (see slide 69).
2. Both binary trees are LHs.
3. No heap is empty.
4. $18 < 21$
5. Recursively merge the heap with 21 with right subheap of the heap with 18.

March 10, 2021

Borahan Tümer, Ph.D.

90

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

91

Recursive Merging Algorithm

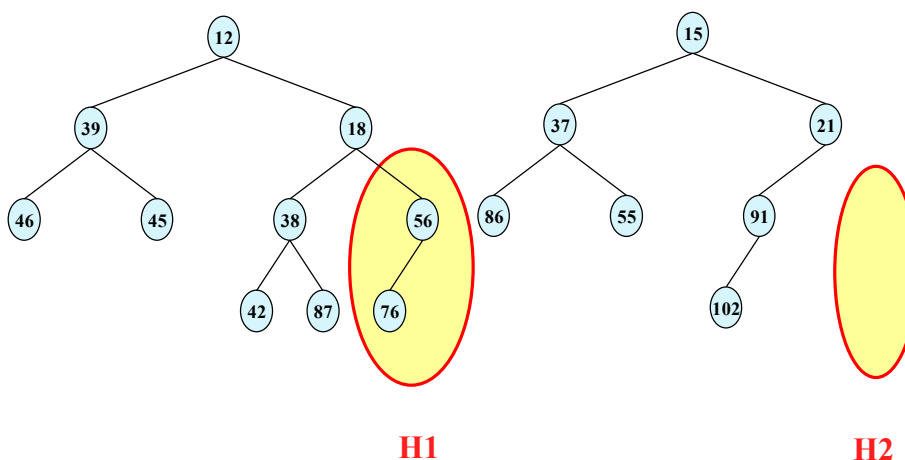
1. Input: two LHs H1 and H2 (both within the red ellipses) (see slide 71).
2. Both binary trees are LHs.
3. No heap is empty.
4. $21 < 56$
5. Recursively merge the heap with 56 with right subheap of the heap with 21.

March 10, 2021

Borahan Tümer, Ph.D.

92

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

93

Recursive Merging Algorithm

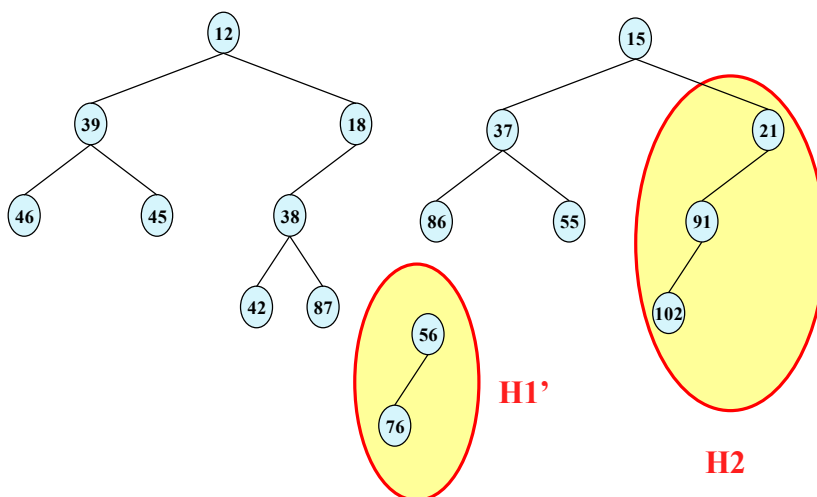
1. Input: two LHs H1 and H2 (both within the red ellipses) (see slide 73).
2. Both binary trees are LHs.
3. The right child of 21 is null. Hence, the resulting heap of merging the heap with root 56 (H1) and the right subheap of heap with root 21 (i.e., an empty heap) is H1 (the heap with root 56).
7. H1' is still a LH. No swaps necessary!
8. Skip;
9. The execution returns with H1' (see slide 75).

March 10, 2021

Borahan Tümer, Ph.D.

94

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

95

Recursive Merging Algorithm

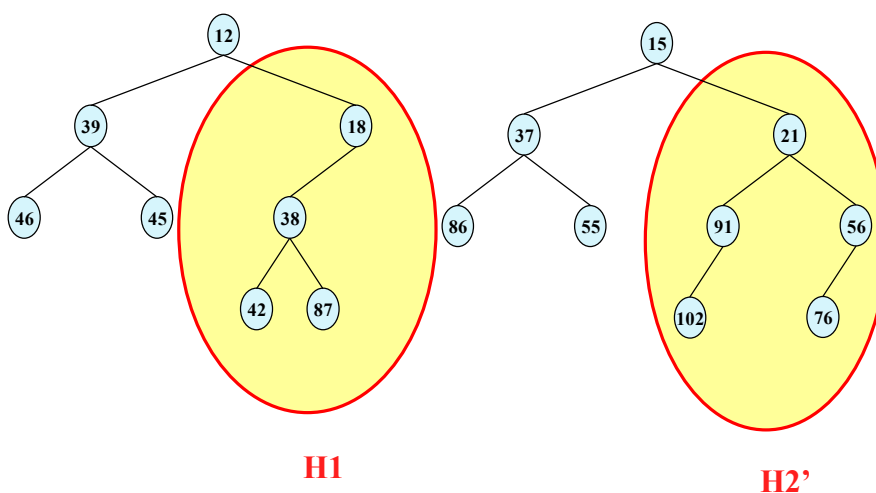
6. The return address of execution is step 6 of the instance of merge algorithm at which H1 was the heap with root 56 and H2 was the heap with 21 (see slide 71!). Now, the new H1 (H1') is the same heap; but it is the result of the last recursive merge. At this step, H1' is merged with the current H2 (see slide 75!)
7. H2' is still a LH. No swaps necessary!
8. Skip;
9. The execution returns with H2' (see slide 77).

March 10, 2021

Borahan Tümer, Ph.D.

96

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

97

Recursive Merging Algorithm

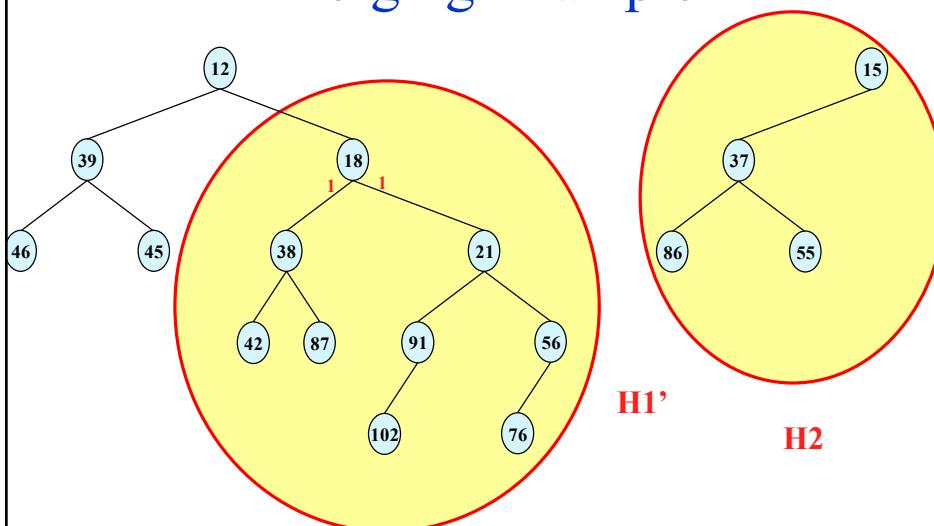
6. The return address of execution is step 6 of the instance of merge algorithm at which H1 was the heap with root 18 and H2 was the heap with 21 (see slide 69!). Now, H2' is merged with the current H1 (see slide 77!)
7. H1' is still a LH. No swaps necessary!
8. Skip;
9. The execution returns with H1' (see slide 79).

March 10, 2021

Borahan Tümer, Ph.D.

98

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

99

Recursive Merging Algorithm

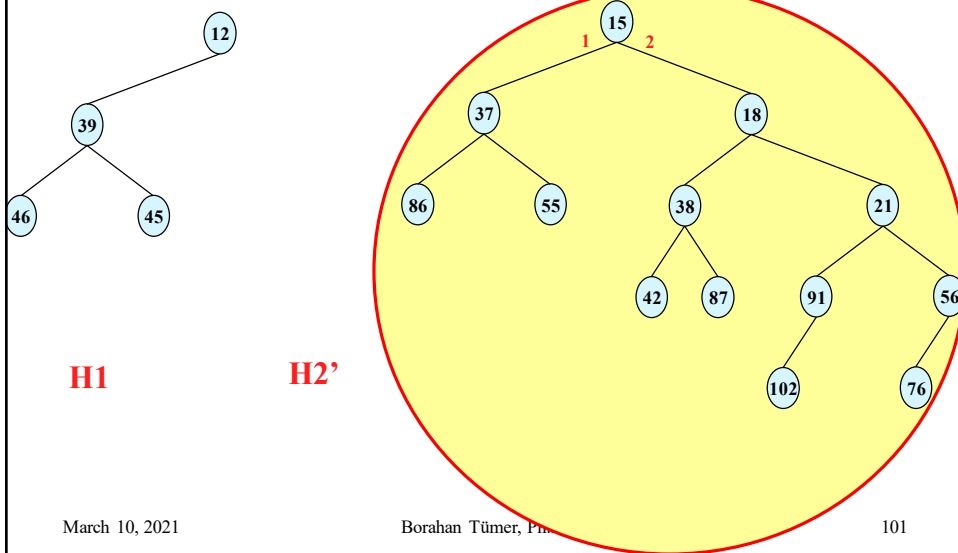
6. The return address of execution is step 6 of the instance of merge algorithm at which H1 was the heap with root 18 and H2 was the heap with 15 (see slide 67!). Now, H1' is merged with the current H2 (see slide 79!)
7. H2' is not a LH ($Npl(LC) < Npl(RC)$). (see slide 81!)
8. We swap LC and RC of 15!
9. Execution returns with H2'' (see slide 82).

March 10, 2021

Borahan Tümer, Ph.D.

100

Merging Example

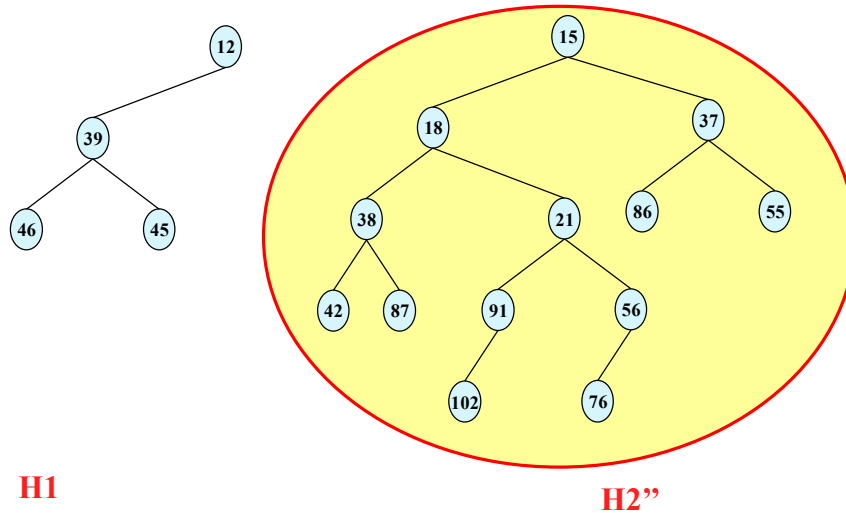


March 10, 2021

Borahan Tümer, Ph.D.

101

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

102

Recursive Merging Algorithm

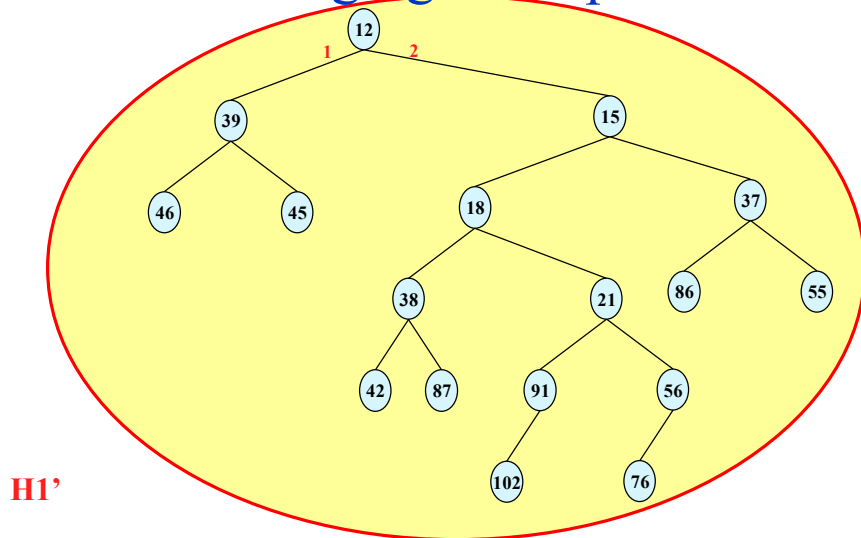
6. The return address of execution is step 6 of the instance of merge algorithm at which H1 was the heap with root 12 and H2 was the heap with 15 (see slide 65!). Now, H2'' is merged with the current H1 (see slide 84!)
7. H1' is not a LH ($Npl(LC) < Npl(RC)$).
8. We swap LC and RC of 12!
9. Execution returns with H1'' (see slide 85).

March 10, 2021

Borahan Tümer, Ph.D.

103

Merging Example

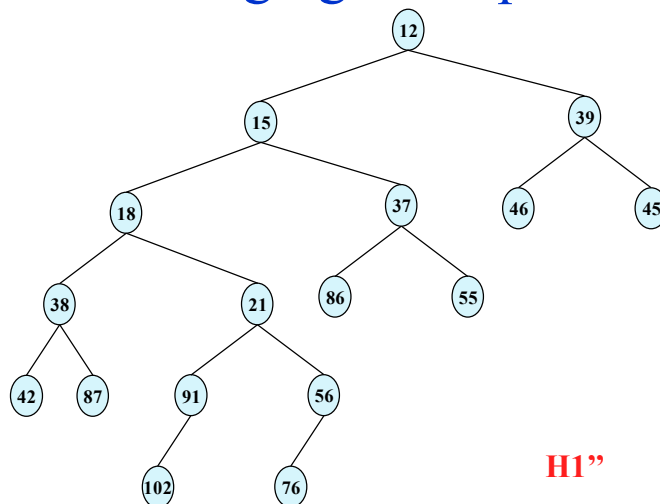


March 10, 2021

Borahan Tümer, Ph.D.

104

Merging Example



March 10, 2021

Borahan Tümer, Ph.D.

105

Non-Recursive Merging Algorithm: First Pass

- *Two passes*
- *First pass:*
 - arrange the nodes on the *right-most path* of both LHs in *sorted order*, keeping their respective left children;
 - create a new tree from the two LHs;
 - sort and arrange the nodes above,
 - make them the right path of the new tree.

March 10, 2021

Borahan Tümer, Ph.D.

106

Non-Recursive Merging Algorithm: Second Pass

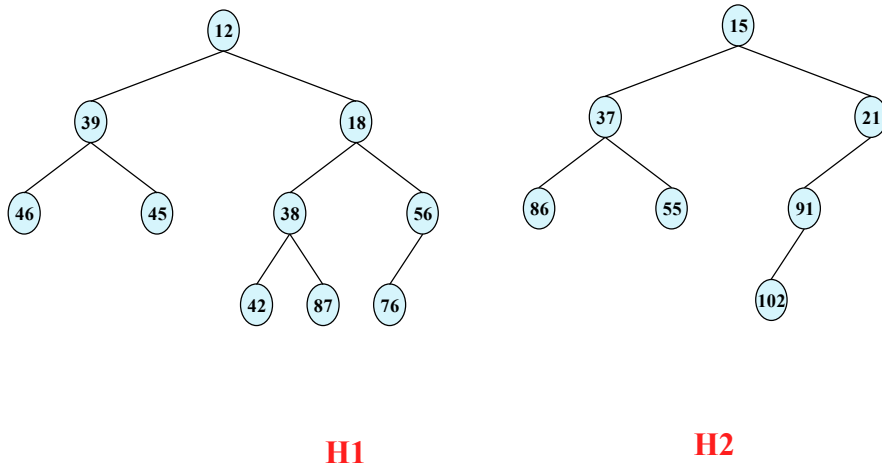
- Start a *bottom-up analysis* to
 - check and determine the nodes at which the leftist heap property is violated, and
 - perform a swap at these nodes.

March 10, 2021

Borahan Tümer, Ph.D.

107

Merging Example



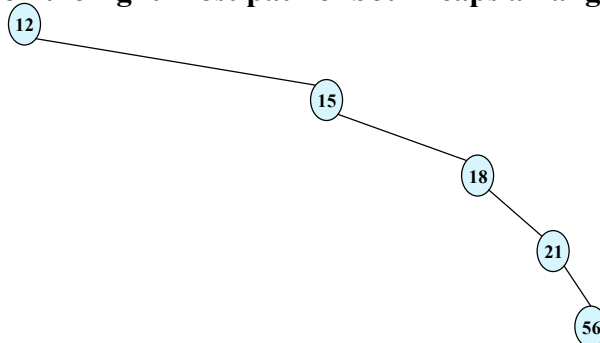
March 10, 2021

Borahan Tümer, Ph.D.

108

Merging Example: First Pass

- Nodes on the right-most path of both heaps arranged!



- A new tree made with the above as its right-most path!

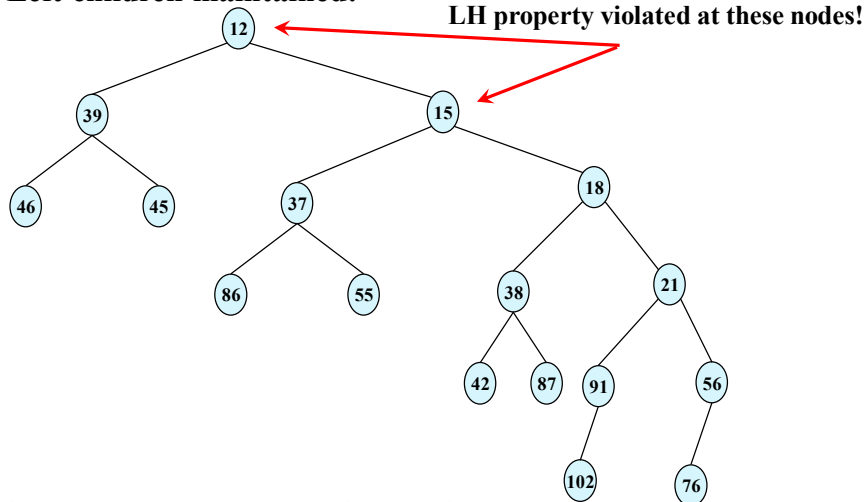
March 10, 2021

Borahan Tümer, Ph.D.

109

Merging Example: First Pass

- **Left children maintained!**

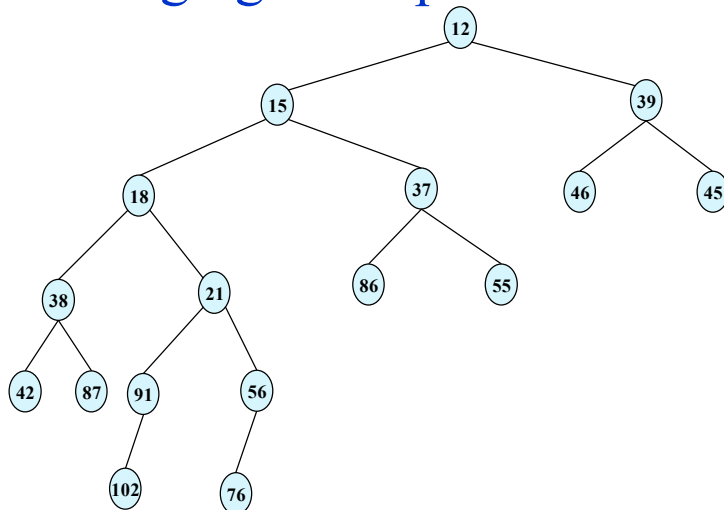


March 10, 2021

Borahan Tümer, Ph.D.

110

Merging Example: Second Pass



March 10, 2021

Borahan Tümer, Ph.D.

111

Time Analysis of Merging using LHs

Performing the recursive solution is proportional to the sum of the length of the right paths. The work done at each node visited on the right path is constant.

We have $O(\log(n))$ nodes on the right path in a LH with n nodes.

March 10, 2021

Borahan Tümer, Ph.D.

112

Binomial Heaps

March 10, 2021

Borahan Tümer, Ph.D.

113

Motivation for Binomial Heaps

- Leftist Heaps support
 - *merging, insertion, removal, and deleteMin*
- in $O(\log(n))$ time per operation.
- We know *binary heaps* have a constant (i.e., $O(1)$) insertion time.
- Question: May there be a data structure providing
 - $O(1)$ time for insertion, and
 - $O(\log(n))$ time for each other operation.
- This data structure is the so-called *binomial heaps (BHs) or binomial queues*.
- To study BHs we first need to discuss *binomial trees*.

March 10, 2021

Borahan Tümer, Ph.D.

114

Binomial Trees

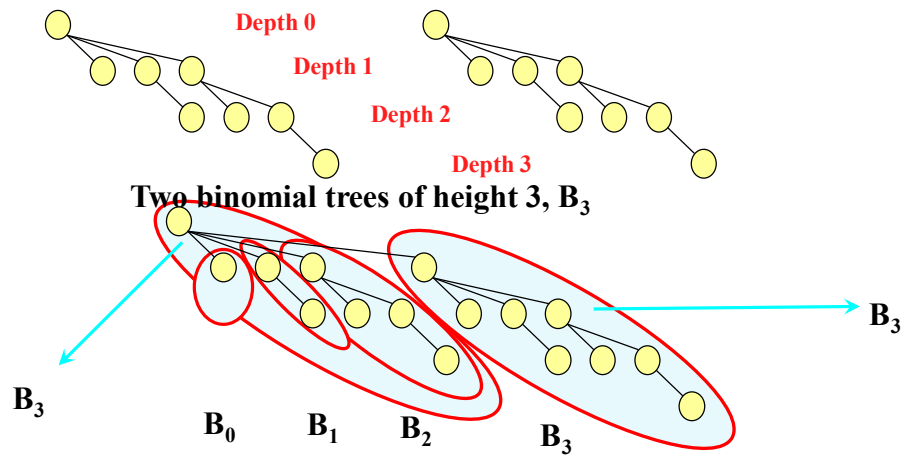
- A binomial tree B_k is an *ordered tree* (i.e., a rooted tree in which the children of each node are ordered; the order of the children matters) *defined recursively*.
- The binomial tree B_0 consists of a single node. A binomial tree B_k of *height* k is formed by attaching a B_{k-1} to the root of another B_{k-1} .
- In the next slide, we see two B_3 s combined to form a B_4 .

March 10, 2021

Borahan Tümer, Ph.D.

115

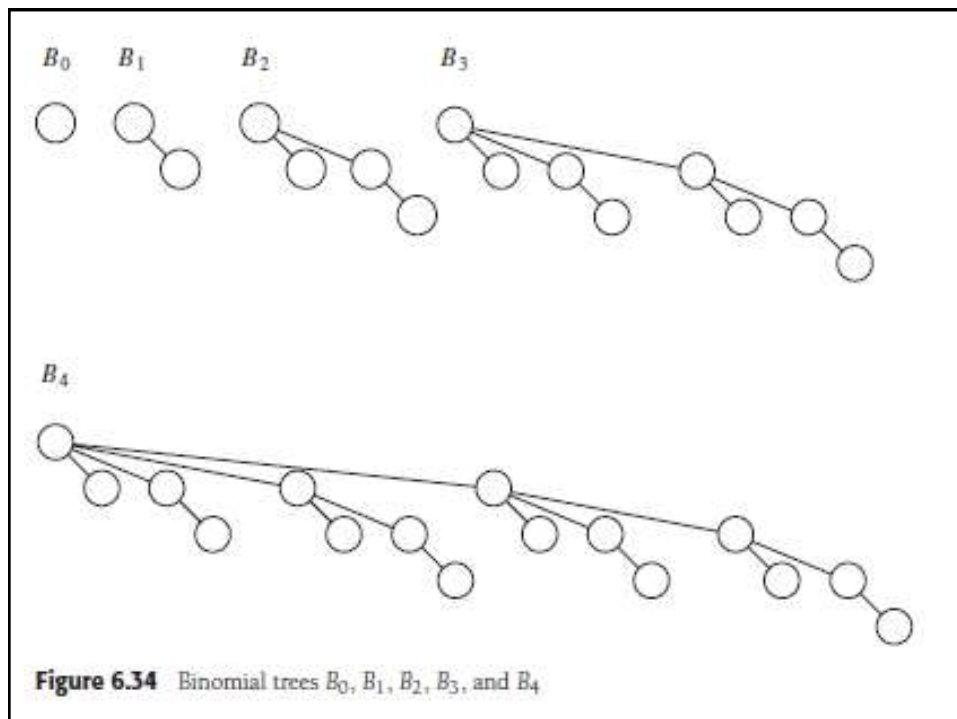
Binomial Trees



March 10, 2021

Borahan Tümer, Ph.D.

116



Binomial Trees

- A *binomial tree* B_k has
 - a height of k ;
 - $n = 2^k$ nodes (!!!);
 - $k+1$ depth levels ranging within $0, \dots, k$.
 - $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ nodes at *depth level* d .
 - a root and a $B_0, B_1, B_2, B_3, \dots, B_{k-1}$ connected to it in respective order (see slide 96!).

March 10, 2021

Borahan Tümer, Ph.D.

118

Binomial Heaps

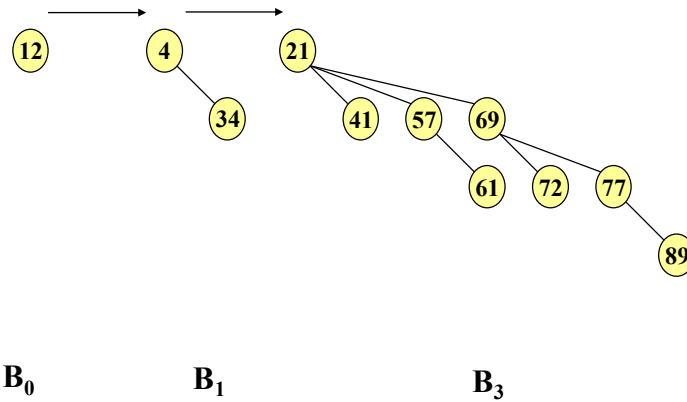
- *BHs* differ from other heap structures in that
 - a BH is *not a heap-ordered tree* but rather *a collection of heap ordered trees, a forest*.
 - each heap-ordered tree is of a constrained form known as a *binomial tree*.
- Each binomial tree in a BH obeys *min-heap* or *heap order* property.
- There is *at most one* B_k of *each height* k in a BH.
- In the next slide, we see an 11-node BH.

March 10, 2021

Borahan Tümer, Ph.D.

119

An 11-element Binomial Heap



March 10, 2021

Borahan Tümer, Ph.D.

120

Operations on Binomial Heaps

- *BH_Create()*
 - Creates the special header node for the BH.
- *BH_Find_Min()*
 - Finds the minimum key in the BH.
- *BH_Merge(H_1, H_2)*
 - Merges two BHs H_1 and H_2 .
- *BH_Insert(H, x)*
 - Inserts x into H .
- *BH_Delete-Min(H)*
 - Deletes the minimum key in H .
- *In preparing algorithms/pseudocode for these operations, [1] has been the main reference.*

March 10, 2021

Borahan Tümer, Ph.D.

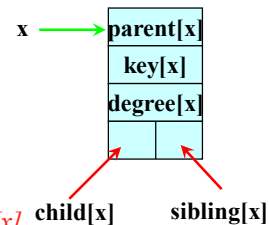
121

Operations on Binomial Heaps

- *Assumptions*

- Each node x in a BH contains

- a key field $key[x]$,
- a parent pointer, $p[x]$,
- a child pointer to its left-most child, $child[x]$,
- a pointer to its immediate right sibling, $sibling[x]$,
- a field holding the number of children, $degree[x]$.



March 10, 2021

Borahan Tümer, Ph.D.

122

Creating an Empty Tree

```
BH_Header * BH_Create()
{ // creates a special header node for a BH.
  BH_Header *BH_hdr;
  BH_hdr=(BH_Header *) malloc(sizeof(BH_Header));
  ... // here, proper values are assigned to special header
       fields.
  BH_hdr->first=NULL;
  return BH_hdr;
}
```

Running time: $\Theta(1)$

March 10, 2021

Borahan Tümer, Ph.D.

123

Finding Minimum Key in a BH

```

BH_Find_Min(BH_hdr)
{ // finds minimum key in the BH.
  y=NULL; x=BH_hdr->first; min= $\infty$ ;
  while (x != NULL)
    if (key[x]<min) { min=key[x]; y=x; }
    x = sibling[x];
  return y;
}

```

Running time: $O(\lg(n))$ Why?

March 10, 2021

Borahan Tümer, Ph.D.

124

Obtaining a B_k from two B_{k-1} s

```

Get_a_BT_k(y,z)
{ // obtains a  $B_k$  from two  $B_{k-1}$ s. Root of new B is z.
  p[y]=z;
  sibling[y]=child[z];
  child[z]=y;
  degree[z]++;
}

```

Running time: $\Theta(1)$

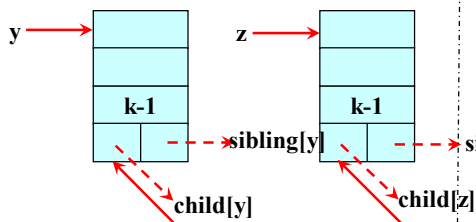
March 10, 2021

Borahan Tümer, Ph.D.

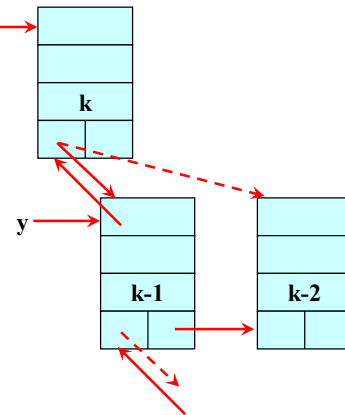
125

Get_a_BT_k(y,z) illustrated...

Before Get_a_BT_k(y,z)



After Get_a_BT_k(y,z)



Get_a_BT_k(y,z)

```
{
  p[y]=z;
  sibling[y]=child[z];
  child[z]=y;
  degree[z]++;
}
```

March 10, 2021

Borahan Tümer, Ph.D.

126

Merging two BHs

```
BH_Merge(BH1_hdr,BH2_hdr)
{ // merges two BH1 and BH2
  BH_hdr= BH_Create();
  BH_hdr->first=Merge_Root_Lists(); // Merges root lists of BH1 and BH2 into one sorted by
                                     // ascending degree.
  if (BH_hdr->first == NULL) return BH_hdr->first;
  q=NULL; x=BH_hdr->first; r=sibling[x];
  while (r !=NULL)
    if (degree[r]!=degree[x] || sibling[r]!=NULL &&
        degree[sibling[r]]==degree[x]) {q=x; x=r;}
    else if (key[x]<=key[r]){sibling[x]=sibling[r];Get_a_BT_k(r,x);}
    else { if (q == NULL) BH_hdr->first=r; else sibling[q]=r;
          Get_a_BT_k(x,r); x=r;
        }
    r=sibling[x];
  return BH_hdr;
}
```

Running time: $O(\lg(n))$

March 10, 2021

Borahan Tümer, Ph.D.

127

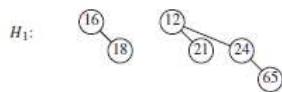


Figure 6.35 Binomial queue H_1 with six elements

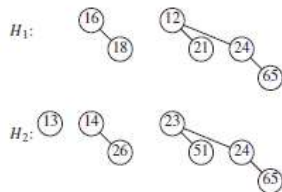


Figure 6.36 Two binomial queues H_1 and H_2

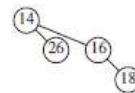


Figure 6.37 Merge of the two B_1 trees in H_1 and H_2

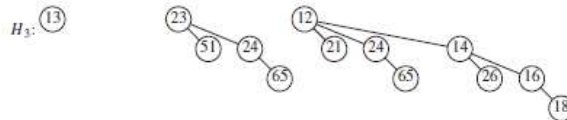


Figure 6.38 Binomial queue H_3 : the result of merging H_1 and H_2

March 10, 2021



Figure 6.39 After 1 is inserted



Figure 6.40 After 2 is inserted

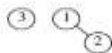


Figure 6.41 After 3 is inserted

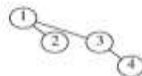


Figure 6.42 After 4 is inserted



Figure 6.43 After 5 is inserted

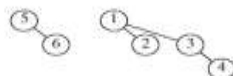


Figure 6.44 After 6 is inserted

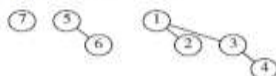
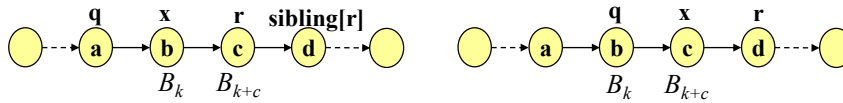


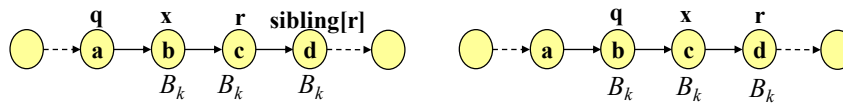
Figure 6.45 After 7 is inserted

Illustration of Various Cases in Merging

degree[r] \neq degree[x]



sibling[r] \neq NULL && degree[sibling[r]] = degree[x]



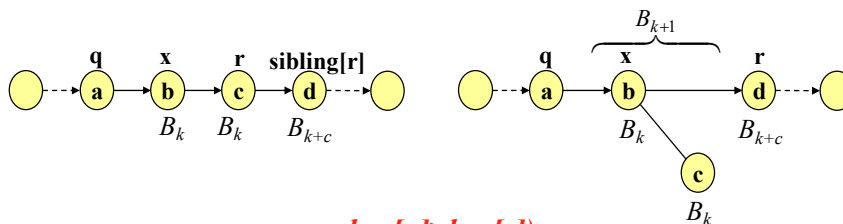
March 10, 2021

Borahan Tümer, Ph.D.

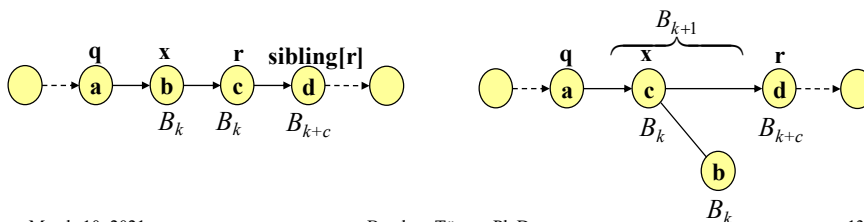
130

Illustration of Various Cases in Merging

key[x] \leq key[r]



key[x] > key[r]

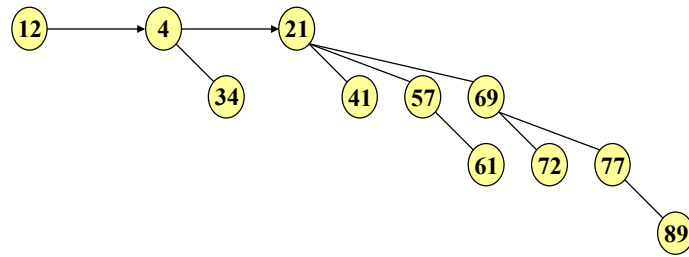


March 10, 2021

Borahan Tümer, Ph.D.

131

An 11-element BH

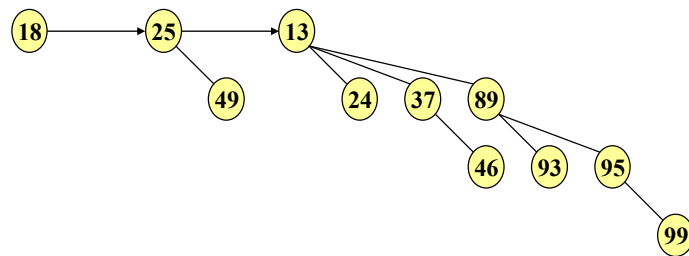
 B_0 B_1 B_3

March 10, 2021

Borahan Tümer, Ph.D.

132

Another 11-element BH

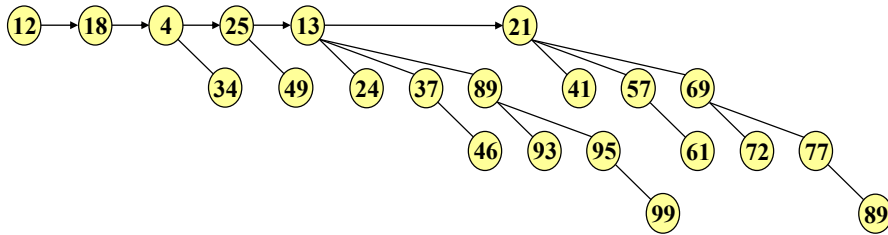
 B_0 B_1 B_3

March 10, 2021

Borahan Tümer, Ph.D.

133

Two BHs merged...

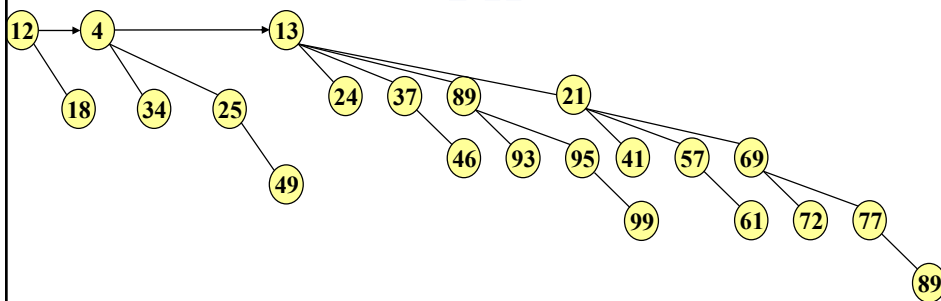


March 10, 2021

Borahan Tümer, Ph.D.

134

Two BHs merged... into a 22-element BH



March 10, 2021

Borahan Tümer, Ph.D.

135

Inserting a node into a BH

- Insertion of a node into a BH is the same as merging a single-node BH with another BH.

```
BH_Insert(BH_hdr,x)
{ // inserts x into BH.
  BH1_hdr=BH_Create();
  // makes a single-node (degree-0) BH out of x.
  p[x]=child[x]=sibling[x]=NULL; degree[x]=0;
  BH1_hdr->first=x;
  BH_hdr=BH_Merge(BH1_hdr,BH_hdr);
}
```

Running time: worst case: $O(\lg(n))$

March 10, 2021

Borahan Tümer, Ph.D.

136

Delete-Min in a BH

```
BH_Delete-Min(BH_hdr)
{ // deletes the minimum key in the BH.
  – Find the root x with minimum key //(wc:  $O(\lg(n))$ )
  – Remove x
  – BH1_hdr=BH_Create();
  – Establish a LL of x's children //(wc:  $O(\lg(n))$ )
  – BH1_hdr->first=pointer to node with degree=0
  – BH_hdr=BH_Merge(BH1_hdr,BH_hdr); //(wc:  $O(\lg(n))$ )
}
```

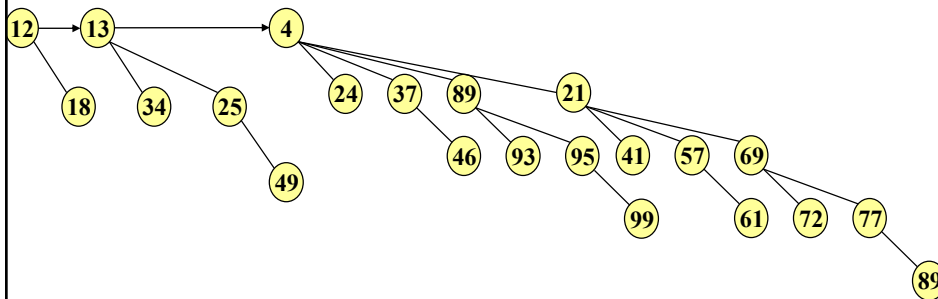
Running time: $O(\lg(n))$

March 10, 2021

Borahan Tümer, Ph.D.

137

A 22-element BH... Delete-min



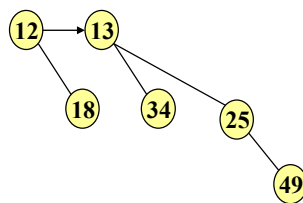
March 10, 2021

Borahan Tümer, Ph.D.

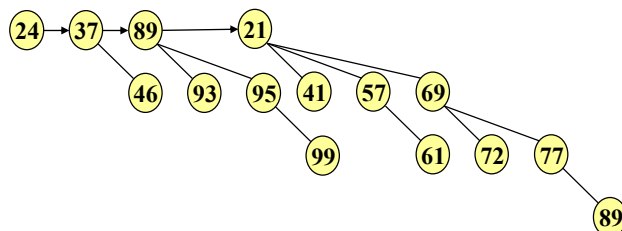
138

4 removed...

BH



BH1



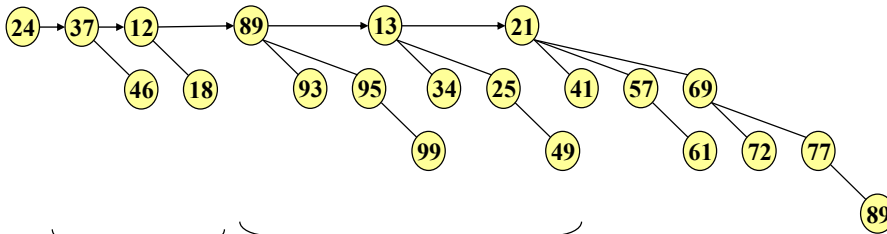
March 10, 2021

Borahan Tümer, Ph.D.

139

BH merged with BH1... 2 B₁s, 2B₂s

BH...



2 B₁s to merge
to a B₂

2 B₂s to merge to a B₃

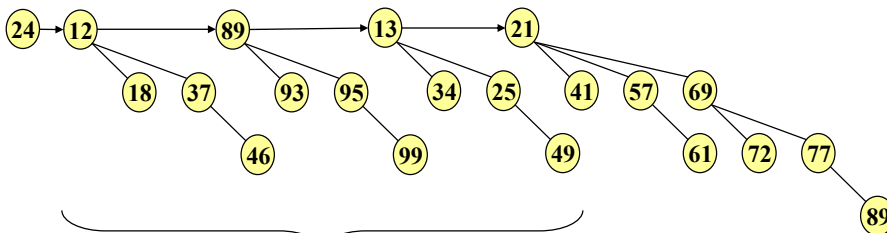
March 10, 2021

Borahan Tümer, Ph.D.

140

BH merged with BH1... 3B₂s

BH...



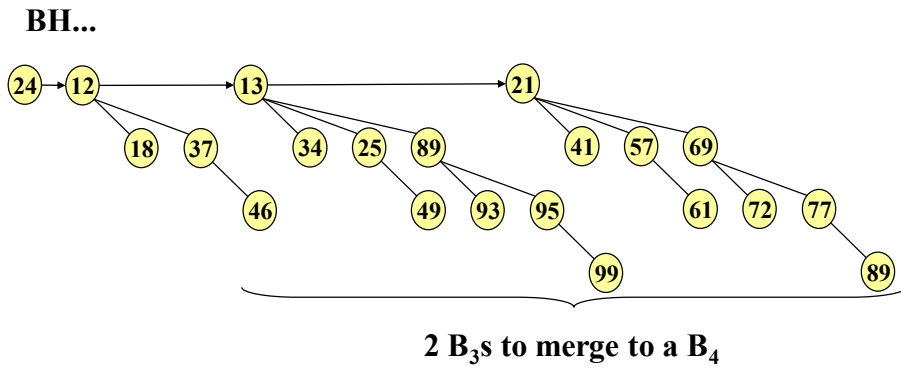
3B₂s... Last two to
merge to a B₃

March 10, 2021

Borahan Tümer, Ph.D.

141

BH merged with BH1... 2 B_3 s

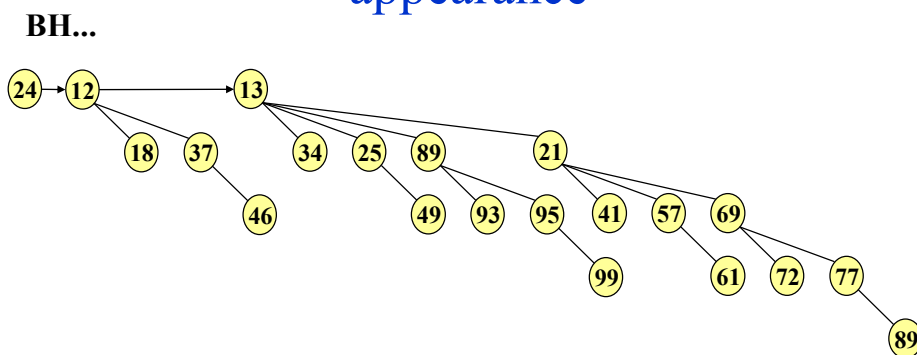


March 10, 2021

Borahan Tümer, Ph.D.

142

BH merged with BH1... Final appearance



March 10, 2021

Borahan Tümer, Ph.D.

143

Reference...

- [1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein,
“*Introduction to Algorithms*,” 2nd edition, MIT Press,
2003