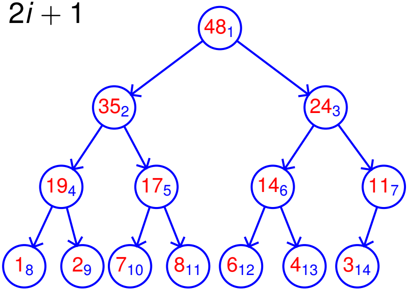
Table of Contents

Section outline

1. **Binary Heaps**
   - Structure of binary heaps
   - Top-down construction of binary heap (min-heap)
   - Analysis of top-down construction
   - Bottom-up construction of binary heap (max-heap)
   - Analysis of bottom-up binary heap construction
   - Operations on binary heaps

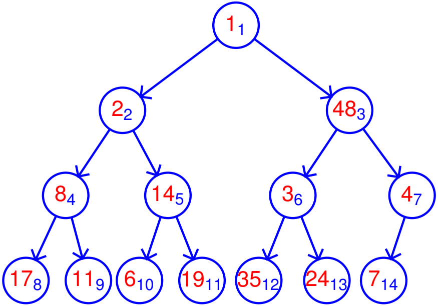
Structure of binary heaps

- Must be a complete rooted binary tree
- For a max-heap (min-heap) the key in the root node must be larger (smaller) than either children
- Each of the two children should be a max-heap (min-heap)
- Developed JWJ Williams
- BFS numbering of the nodes are shown in blue
- A node numbered n, has parent at $\left\lfloor \frac{i}{2} \right\rfloor$, left child at $2i$ and right child at $2i+1$



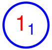
- Can be implemented using an array, indexing nodes by their BFS numbering

Construction schemes



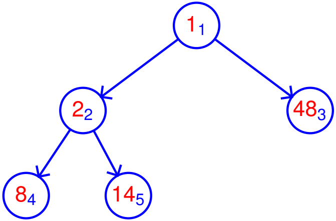
- Nodes are initially present in the array in arbitrary order, shown in the complete binary tree, for convenience
- Adjoining structure is not a heap
- The heap can be constructed either top-down or bottom-up
- Top-down construction better suited when construction must proceed as nodes become available

Top-down construction of binary heap (min-heap)



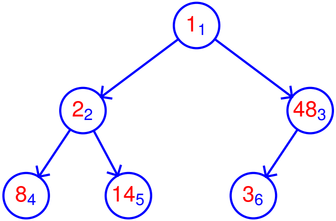
- Nodes are inserted in the heap one by one
- New node is added after last node and moved up the tree, as required
- Tree containing only ‘1’ is heap

Top-down construction of binary heap (min-heap)



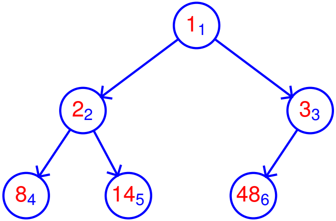
- Nodes are inserted in the heap one by one
- New node is added after last node and moved up the tree, as required
- Insertion of these keys, in order, do not disturb the heap property

Top-down construction of binary heap (min-heap)



- Nodes are inserted in the heap one by one
- New node is added after last node and moved up the tree, as required
- Insertion of ‘3’ disturbs the min-heap property, so adjustments will be needed

Top-down construction (contd.)

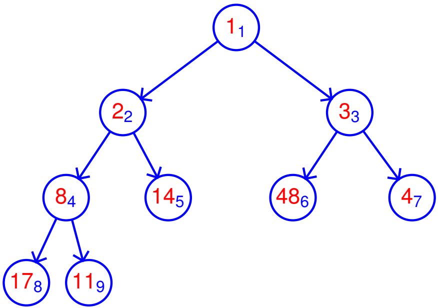


- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

Top-down construction (contd.)

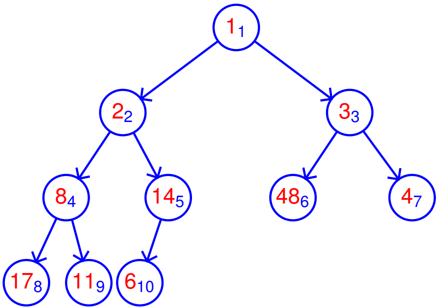


Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Top-down construction (contd.)

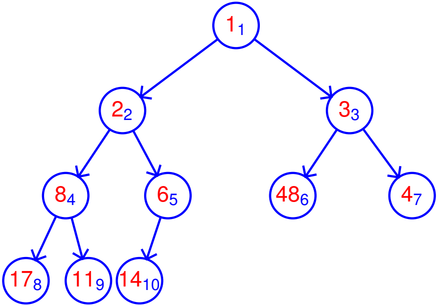


- ‘3’ and ‘48’ (parent of ‘3’) are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of ‘6’ disturbs the min-heap, so interchange with ‘14’
- ‘19’ is properly inserted, but ‘35’ disturbs the heap
- ‘35’ and ‘48’ are interchanged
- Insertion of ‘24’ disturbs the heap, interchanged with ‘35’
- Insertion of ‘7’ does not disturb the heap

Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

Top-down construction (contd.)

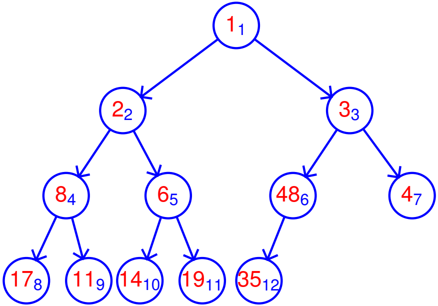


Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Top-down construction (contd.)

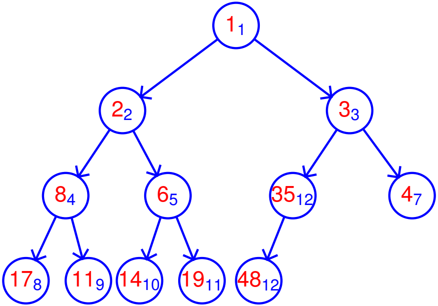


- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

Top-down construction (contd.)

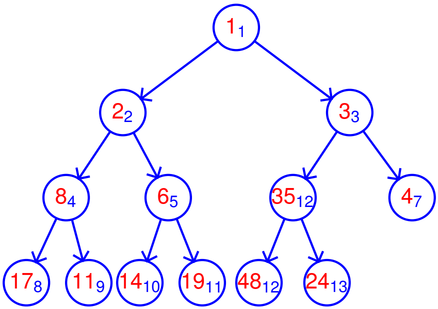


Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Top-down construction (contd.)

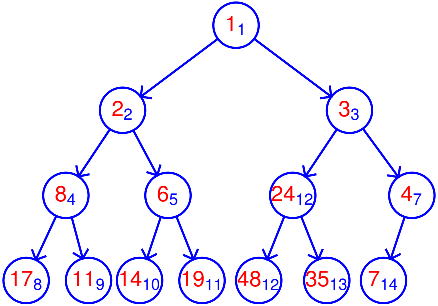


Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Top-down construction (contd.)

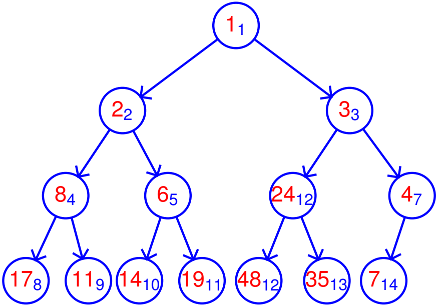


Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

- ‘3’ and ‘48’ (parent of ‘3’) are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of ‘6’ disturbs the min-heap, so interchange with ‘14’
- ‘19’ is properly inserted, but ‘35’ disturbs the heap
- ‘35’ and ‘48’ are interchanged
- Insertion of ‘24’ disturbs the heap, interchanged with ‘35’
- Insertion of ‘7’ does not disturb the heap

Top-down construction (contd.)

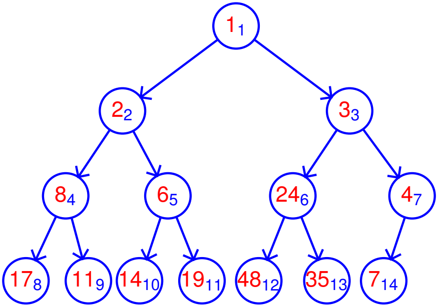


- '3' and '48' (parent of '3') are interchanged
- Insertion of these keys, in order, do not disturb the heap property
- Insertion of '6' disturbs the min-heap, so interchange with '14'
- '19' is properly inserted, but '35' disturbs the heap
- '35' and '48' are interchanged
- Insertion of '24' disturbs the heap, interchanged with '35'
- Insertion of '7' does not disturb the heap

Percolation

The process of smaller (*lighter*) keys going up (*bubbling up*) and larger larger (*heavier*) keys going down (*settling down*) is called *percolation*

Analysis of top-down construction



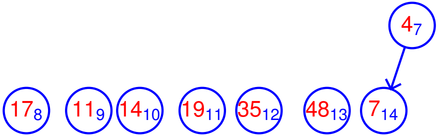
- The number of nodes in the tree after inserting k-th node is k
- This node may have to rise through $\lg k$ levels
- Total cost of building the heap this way:

$$\sum_{k=1}^{n} \lg k = \lg n! \in O(n \lg n)$$

Bottom-up construction of binary heap (max-heap)

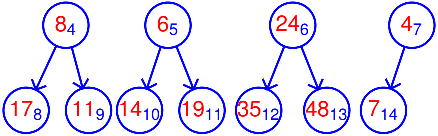
- The leaf-level nodes (at the end of the array) are all individual heaps

Bottom-up construction of binary heap (max-heap)



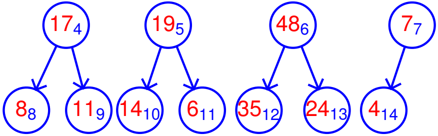
- The leaf-level nodes (at the end of the array) are all individual heaps
- The first internal node is at $\left\lfloor \frac{n}{2} \right\rfloor$
- Incorporation of '4' disturbs the heap property, correction by way of interchanging with larger child key needed

Bottom-up construction of binary heap (max-heap)



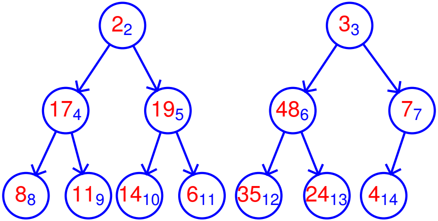
- The leaf-level nodes (at the end of the array) are all individual heaps
- The first internal node is at $\left\lfloor \frac{n}{2} \right\rfloor$
- Incorporation of ‘4’ disturbs the heap property, correction by way of interchanging with larger child key needed
- Similar problem with the incorporation of ‘24’, ‘6’, ‘8’, therefore, interchanges with larger child key are needed

Bottom-up construction of binary heap (contd.)



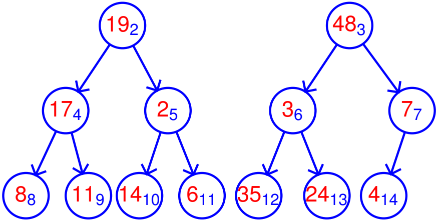
- Keys ‘4’, ‘24’, ‘6’, ‘8’ have been interchanged (at most once, being at the penultimate level) to restore the heap property of the individual heaps

Bottom-up construction of binary heap (contd.)



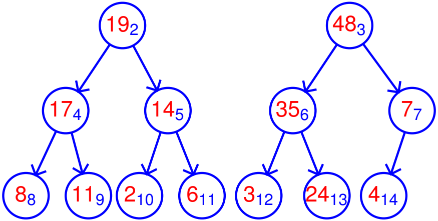
- Keys ‘4’, ‘24’, ‘6’, ‘8’ have been interchanged (at most once, being at the penultimate level) to restore the heap property of the individual heaps
- Now keys ‘3’ and then ‘2’ are introduced, interchanges (now at most twice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



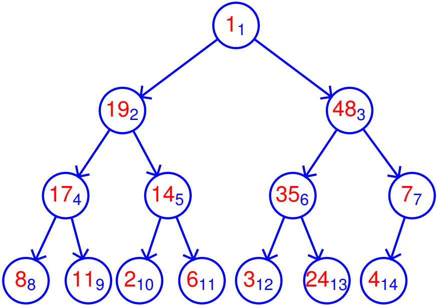
- Keys '4', '24', '6', '8' have been interchanged (at most once, being at the penultimate level) to restore the heap property of the individual heaps
- Now keys '3' and then '2' are introduced, interchanges (now at most twice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



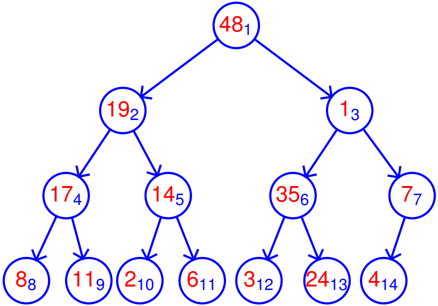
- Keys '4', '24', '6', '8' have been interchanged (at most once, being at the penultimate level) to restore the heap property of the individual heaps
- Now keys '3' and then '2' are introduced, interchanges (now at most twice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



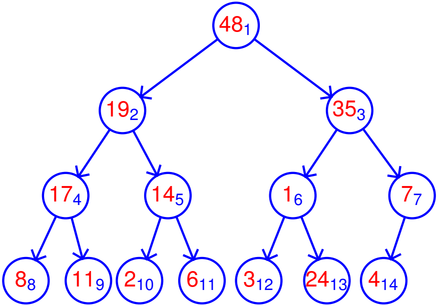
- Now key ‘1’ is introduced at the root, interchanges (now at most thrice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



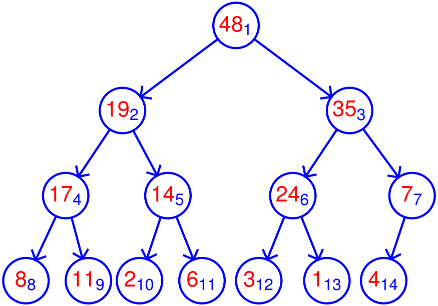
- Now key ‘1’ is introduced at the root, interchanges (now at most thrice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



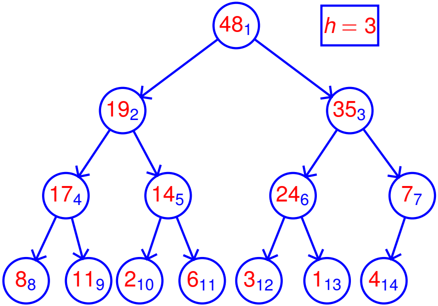
- Now key '1' is introduced at the root, interchanges (now at most thrice) will then be needed to restore min-heap property

Bottom-up construction of binary heap (contd.)



- Now key '1' is introduced at the root, interchanges (now at most thrice) will then be needed to restore min-heap property

Analysis of bottom-up binary heap construction



$\boxed{d = 0, (h-d) = 3, 2^d = 1}$

cost for level: $(h-d) \cdot 2^d = 3 \times 1 = 0$

$\boxed{d = 1, (h-d) = 2, 2^d = 2}$

cost for level: $(h-d) \cdot 2^d = 2 \times 2 = 4$

$\boxed{d = 2, (h-d) = 1, 2^d = 4}$

cost for level: $(h-d) \cdot 2^d = 1 \times 4 = 4$

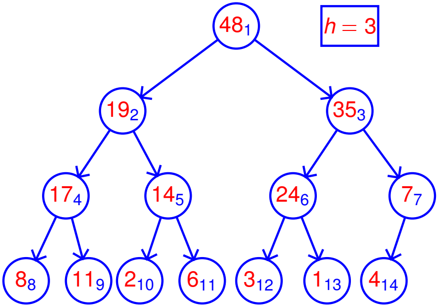
$\boxed{d = 3, (h-d) = 0, 2^d = 8}$

cost for level: $(h-d) \cdot 2^d = 0 \times 8 = 0$

- Apart from simple operations, the main contribution to the cost comes from the percolation of nodes
- Cost of bottom-up construction:

$$\sum_{d=0}^{h} (h-d)2^d = h \sum_{d=0}^{h} 2^d - \sum_{d=0}^{h} d2^d = h(2^{h+1} - 1) - \sum_{d=0}^{h} d2^d$$

Analysis of bottom-up binary heap construction



$\boxed{h = 3}$

$\boxed{d = 0, (h - d) = 3, 2^d = 1}$

cost for level: $(h - d) \cdot 2^d = 3 \times 1 = 0$

$\boxed{d = 1, (h - d) = 2, 2^d = 2}$

cost for level: $(h - d) \cdot 2^d = 2 \times 2 = 4$

$\boxed{d = 2, (h - d) = 1, 2^d = 4}$

cost for level: $(h - d) \cdot 2^d = 1 \times 4 = 4$

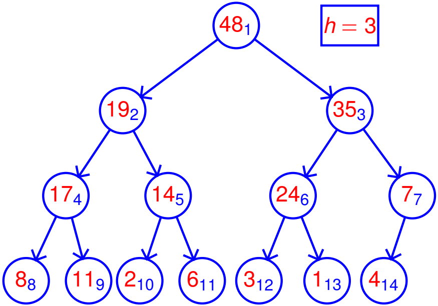
$\boxed{d = 3, (h - d) = 0, 2^d = 8}$

cost for level: $(h - d) \cdot 2^d = 0 \times 8 = 0$

- Apart from simple operations, the main contribution to the cost comes from the percolation of nodes
- Cost of bottom-up construction:

$$\sum_{d=0}^{h} (h-d)2^d = h\sum_{d=0}^{h} 2^d - \sum_{d=0}^{h} d2^d = h(2^{h+1} - 1) - \sum_{d=0}^{h} d2^d$$

Analysis of bottom-up binary heap construction



$\boxed{h = 3}$

$\boxed{d = 0, (h - d) = 3, 2^d = 1}$

cost for level: $(h - d) \cdot 2^d = 3 \times 1 = 0$

$\boxed{d = 1, (h - d) = 2, 2^d = 2}$

cost for level: $(h - d) \cdot 2^d = 2 \times 2 = 4$

$\boxed{d = 2, (h - d) = 1, 2^d = 4}$

cost for level: $(h - d) \cdot 2^d = 1 \times 4 = 4$

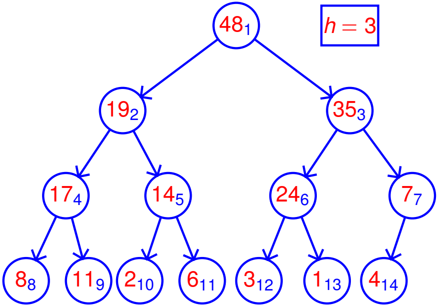
$\boxed{d = 3, (h - d) = 0, 2^d = 8}$

cost for level: $(h - d) \cdot 2^d = 0 \times 8 = 0$

- Apart from simple operations, the main contribution to the cost comes from the percolation of nodes
- Cost of bottom-up construction:

$$\sum_{d=0}^{h} (h - d)2^d = h \sum_{d=0}^{h} 2^d - \sum_{d=0}^{h} d2^d = h(2^{h+1} - 1) - \sum_{d=0}^{h} d2^d$$

Analysis of bottom-up binary heap construction



$\boxed{d = 0, (h-d) = 3, 2^d = 1}$

cost for level: $(h-d) \cdot 2^d = 3 \times 1 = 0$

$\boxed{d = 1, (h-d) = 2, 2^d = 2}$

cost for level: $(h-d) \cdot 2^d = 2 \times 2 = 4$

$\boxed{d = 2, (h-d) = 1, 2^d = 4}$

cost for level: $(h-d) \cdot 2^d = 1 \times 4 = 4$

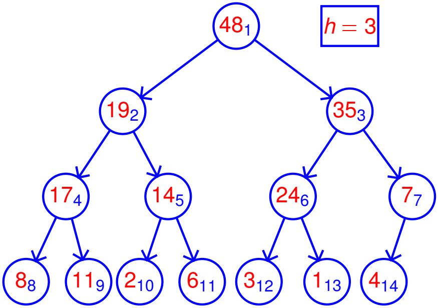
$\boxed{d = 3, (h-d) = 0, 2^d = 8}$

cost for level: $(h-d) \cdot 2^d = 0 \times 8 = 0$

- Apart from simple operations, the main contribution to the cost comes from the percolation of nodes
- Cost of bottom-up construction:

$$\sum_{d=0}^{h} (h-d)2^d = h\sum_{d=0}^{h} 2^d - \sum_{d=0}^{h} d2^d = h(2^{h+1}-1) - \sum_{d=0}^{h} d2^d$$

Analysis of bottom-up binary heap construction



$\boxed{h = 3}$

$\boxed{d = 0, (h - d) = 3, 2^d = 1}$

cost for level: $(h - d) \cdot 2^d = 3 \times 1 = 0$

$\boxed{d = 1, (h - d) = 2, 2^d = 2}$

cost for level: $(h - d) \cdot 2^d = 2 \times 2 = 4$

$\boxed{d = 2, (h - d) = 1, 2^d = 4}$

cost for level: $(h - d) \cdot 2^d = 1 \times 4 = 4$

$\boxed{d = 3, (h - d) = 0, 2^d = 8}$

cost for level: $(h - d) \cdot 2^d = 0 \times 8 = 0$

- Apart from simple operations, the main contribution to the cost comes from the percolation of nodes
- Cost of bottom-up construction:

$$\sum_{d=0}^{h} (h - d)2^d = h \sum_{d=0}^{h} 2^d - \sum_{d=0}^{h} d2^d = h(2^{h+1} - 1) - \sum_{d=0}^{h} d2^d$$

Analysis of binary heap construction (contd.)

- Consider $f(x) = \sum_{d=0}^{h} 2^d x^d = \frac{(2x)^{h+1}-1}{2x-1} = g(x)$
- $f'(x) = 0 + \sum_{d=1}^{h} d2^d x^{d-1}$
- $f'(x)|_{x=1} = \sum_{d=0}^{h} d2^d$
- $g'(x) = \frac{2(h+1)(2x-1)(2x)^h - 2((2x)^{h+1}-1)}{(2x-1)^2}$
- $g'(x)|_{x=1} = 2(h+1)2^h - 2(2^{h+1} - 1) = h2^{h+1} - 2^{h+1} + 2$
- Now, $h(2^{h+1} - 1) - \sum_{d=0}^{h} d2^d = h2^{h+1} - h - h2^{h+1} + 2^{h+1} - 2$

  $= (2^{h+1} - 1) - (h+1) = n - \lg n \in O(n)$
- Thus, a heap is constructed in linear time (asymtotically), in the number of keys

Operations on binary heaps

heapify Making a heap from a complete binary tree rooted at index i (indices starting from 1, such that sub-trees rooted at index positions $2i$ and $2i+1$ are already heaps – time needed is proportional to height of node: $O(\lg n - \lg i)$ time, n being the total number of keys in the array

```
heapifyMax(keyTyp A[], int i, int n) {
  if (2*i >= n) return; // leaf, so done
  int mIdx = 2*i == n ? 2*i ? // only one child
   A[2*i] > A[2*i+1] ? 2*i : 2*i+1;
  keyTyp tky = A[i]; A[i]=A[mIdx]; A[mIdx]=tky;
  heapifyMax(A, mIdx, n); // carry on
}
```

Operations on binary heaps

heapify Making a heap from a complete binary tree rooted at index i (indices starting from 1, such that sub-trees rooted at index positions $2i$ and $2i+1$ are already heaps – time needed is proportional to height of node: $O(\lg n - \lg i)$ time, n being the total number of keys in the array

```
heapifyMax(keyTyp A[], int i, int n) {
  if (2*i >= n) return; // leaf, so done
  int mIdx = 2*i == n ? 2*i ? // only one child
    A[2*i] > A[2*i+1] ? 2*i : 2*i+1;
  keyTyp tky = A[i]; A[i]=A[mIdx]; A[mIdx]=tky;
  heapifyMax(A, mIdx, n); // carry on
}
```

Operations on binary heaps (contd.)

buildHeap Constructing a heap from elements in an array using heapify() for bottom-up constuction – can be done in linear time

```
buildHeap(keyTyp A[], int n) {
  int i = n/2; // index of parent of last leaf
  while (i) // as root has index of 1
    heapify(A, i--, n);
}
```

Operations on binary heaps (contd.)

buildHeap Constructing a heap from elements in an array using heapify() for bottom-up constuction – can be done in linear time

```
buildHeap(keyTyp A[], int n) {
  int i = n/2; // index of parent of last leaf
  while (i) // as root has index of 1
    heapify(A, i--, n);
}
```

Operations on binary heaps (contd.)

insert A new element is added to a heap, at the end of the array and then the heap is adjusted via percolation – can be done in $O(\lg n)$ time

findM The minimum or the maximum key is to be found, this element is always located at the top of the heap – can be done in $O(1)$ time

xtractM The minimum or the maximum key is to be removed from the heap. This requires the last element to replace the min/max element and then the heap is adjusted via percolation – can be done in $O(\lg n)$ time

changeKey The key value associated with an entry is changed, this requires adjustment of the heap via percolation – can be done in $O(\lg n)$ time

Section outline

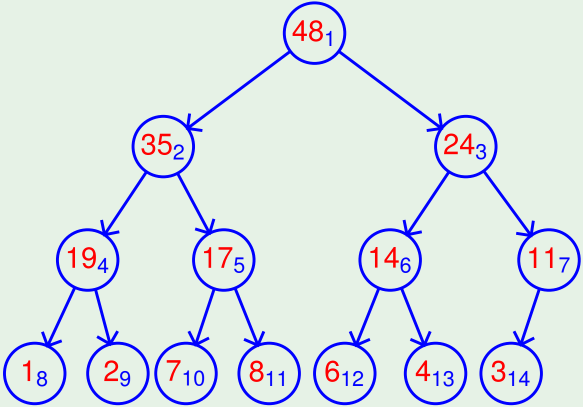
2. **Heap sort**
   - Heap sort mechanism
   - Heap sort example

Heap sort mechanism

1. First make a heap out of the keys stored in the array
2. After that proceed like selection sort
   i. Extract the maximum element, saving it at the position of the right most leaf, before extraction
   ii. Repeat this process until the heap is empty
3. Time complexity: $\sum_{i=n}^{2} \lg_2 i = \Theta(n \lg n)$
4. Invented by JWJ Williams in 1964

Heap sort example

Example

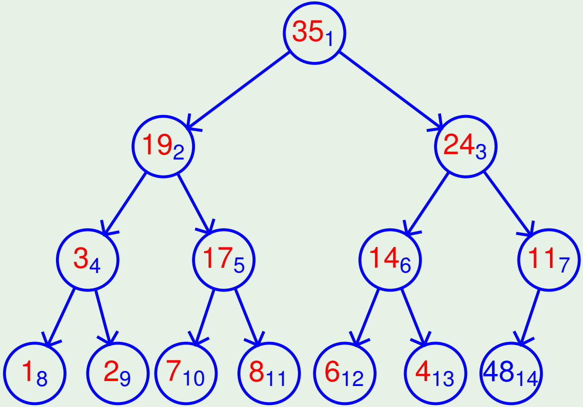


- Elements in the array are formed into a heap using buildHeap()
- Iniital heap
- Layout of the heap in the array

48	35	24	19	17	14	11	1	2	7	8	6	4	3

Heap sort example (contd.)

Example

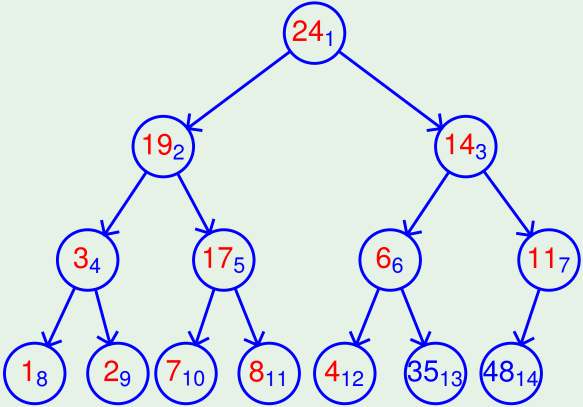


- xtractM()
- Items in the array after xtractM()

35	19	24	3	17	14	11	1	2	7	8	6	4	48

Heap sort example (contd.)

Example

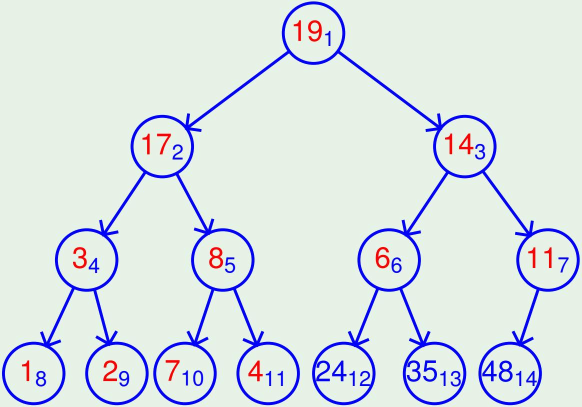


- xtractM()
- Items in the array after xtractM()

24	19	14	3	17	6	11	1	2	7	8	4	35	48

Heap sort example (contd.)

Example

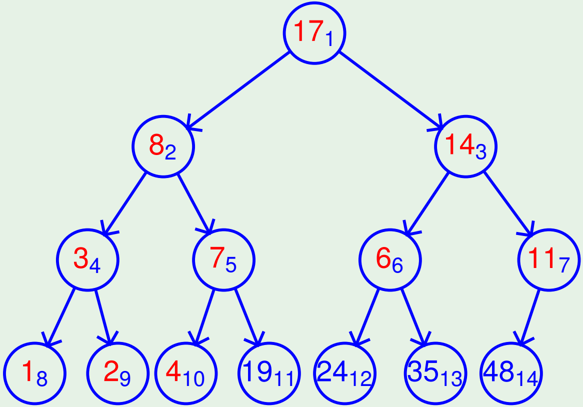


- xtractM()
- Items in the array after xtractM()

19	17	14	3	8	6	11	1	2	7	4	24	35	48

Heap sort example (contd.)

Example

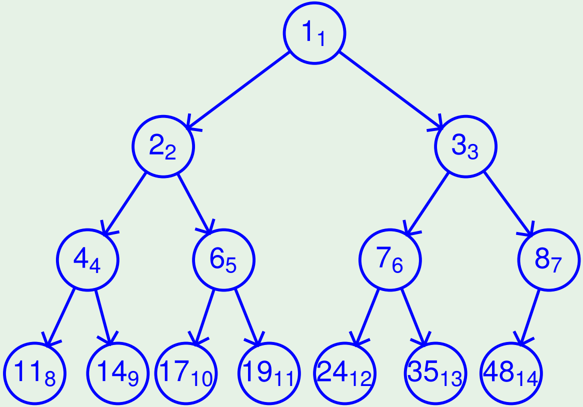


- xtractM()
- Items in the array after xtractM()

17	8	14	3	7	6	11	1	2	4	19	24	35	48

Heap sort example (contd.)

Example



- After extracting all elements from the heap
- Items in the array are sorted in ascending order
- Min heap leads to sorting in descending order

1	2	3	4	6	7	8	11	14	17	19	24	35	48