Heapsort

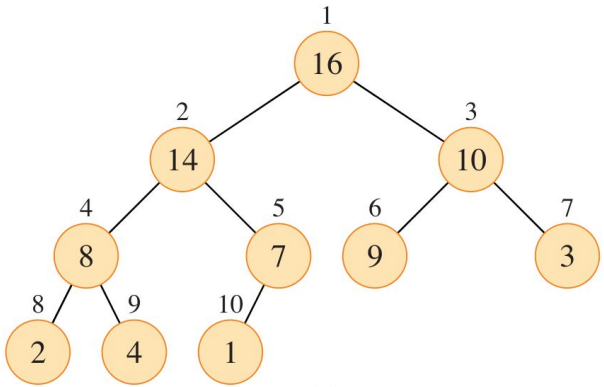
Paramita Koley

Heapsort

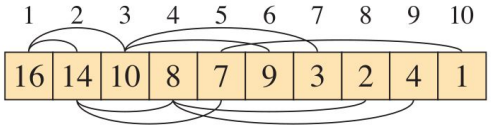
- In-place like insertion sort
- Runs in O(n log n) like merge sort
- Combines better attributes of both algorithms

Heap

(Binary) heap data structure is an array object which can be viewed as a nearly complete binary tree



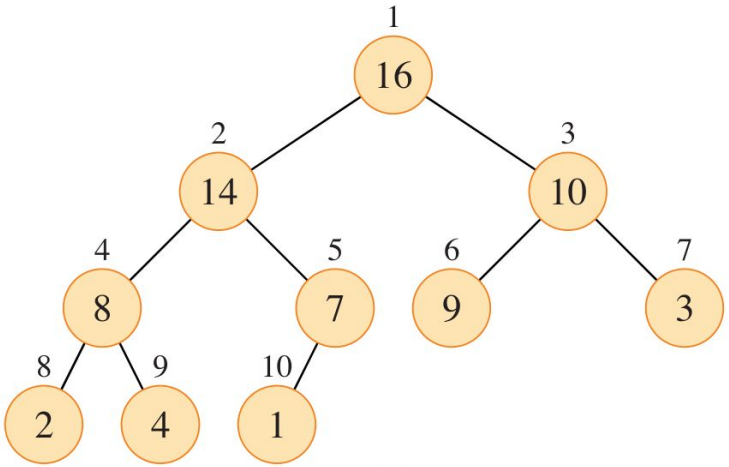
(a)



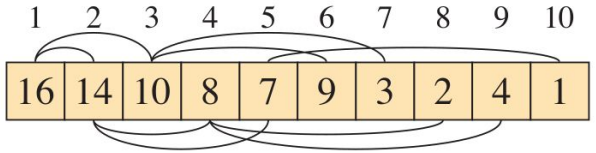
(b)

Attributes: Heap_size

Root of the tree = A[1]



(a)



(b)

```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

Root: A[1]

Max-heap and min-heap

Max-heap

$$A[\text{PARENT}(i)] \geq A[i] \,,$$

Used in heap-sort

Min-heap

$$A[\text{PARENT}(i)] \leq A[i] \,.$$

Used in priority queue

Heap data structure

- Height of a node: number of edges on the longest simple downward path from that node to a leaf
- Height of a heap: height of the root of the heap

What is height of a heap with n elements? $\Theta(\log n)$

Questions

1. Maximum and minimum numbers of elements in a heap of height h?
2. Show that in any subtree of a max-heap, the root of the subtree contains the largest value occurring anywhere in that subtree
3. Where in a max-heap might the smallest element reside, assuming that all elements are distinct?
4. At which levels in a max-heap might the kth largest element reside at max, assuming that all elements are distinct?
5. Is an array that is sorted increasingly a min-heap or max-heap or none?
6. Show that for a nearly complete binary tree, the leaves are A [floor(n/2) + 1: n].

Build heap from an array

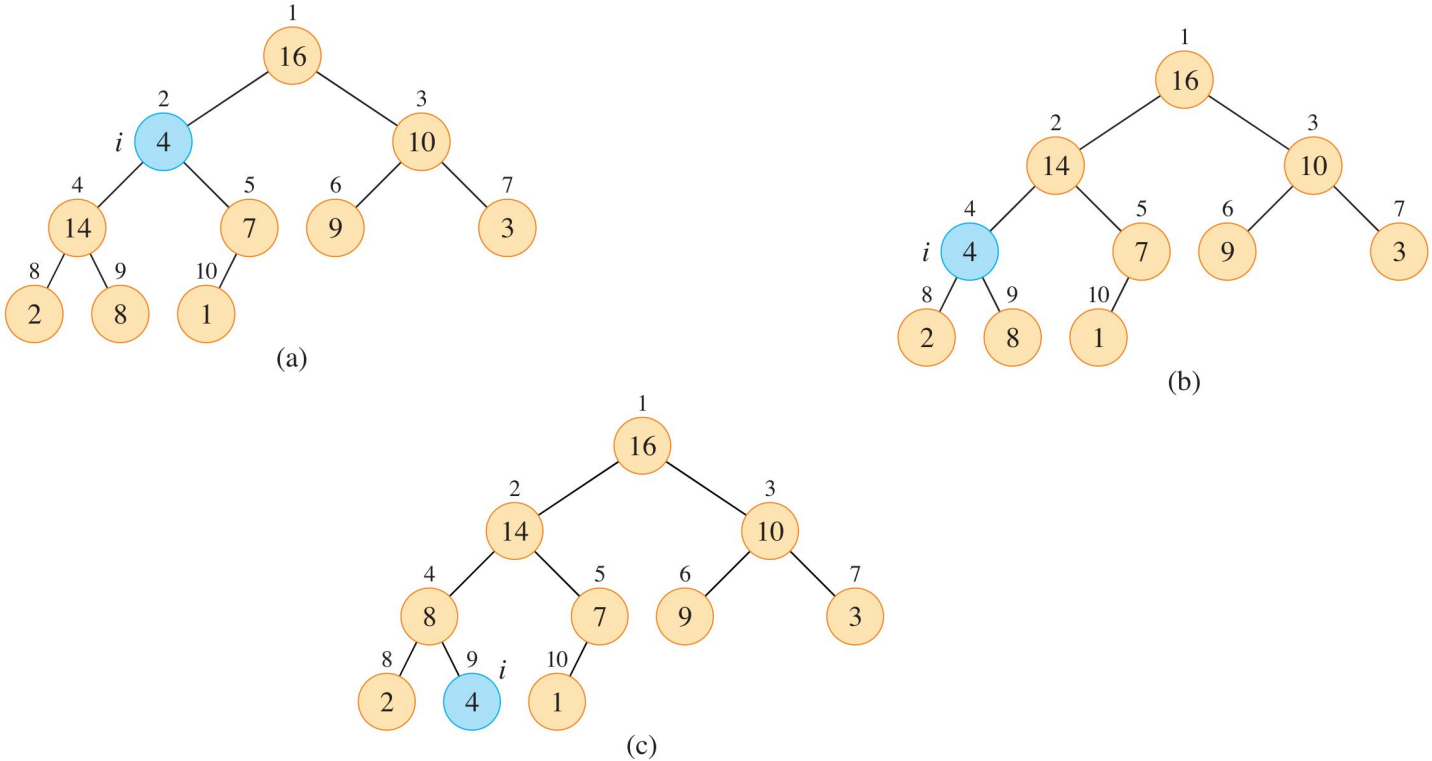
- Max-heapify

  Input: Array A, heap size n, index i

  Assumes left and right subtrees are max-heaps but A [i] can be smaller than any of its child

  Returns a max-heap rooted at i

Max-heapify


1
16
2
i
4
3
10
4
14
5
7
6
9
7
3
8
2
9
8
10
1
(a)
1
16
2
14
3
10
4
i
4
5
7
6
9
7
3
8
2
9
8
10
1
(b)
1
16
2
14
3
10
4
8
5
7
6
9
7
3
8
2
9
i
4
10
1
(c)

Max-heapify

```
MAX-HEAPIFY(A, i)
 1  l = LEFT(i)
 2  r = RIGHT(i)
 3  if l ≤ A.heap-size and A[l] > A[i]
 4       largest = l
 5  else largest = i
 6  if r ≤ A.heap-size and A[r] > A[largest]
 7       largest = r
 8  if largest ≠ i
 9       exchange A[i] with A[largest]
10       MAX-HEAPIFY(A, largest)
```

Run-time complexity

$\Theta(1)$ for initial operations

$T(2n/3)$ for recursion on subtree (Why?)

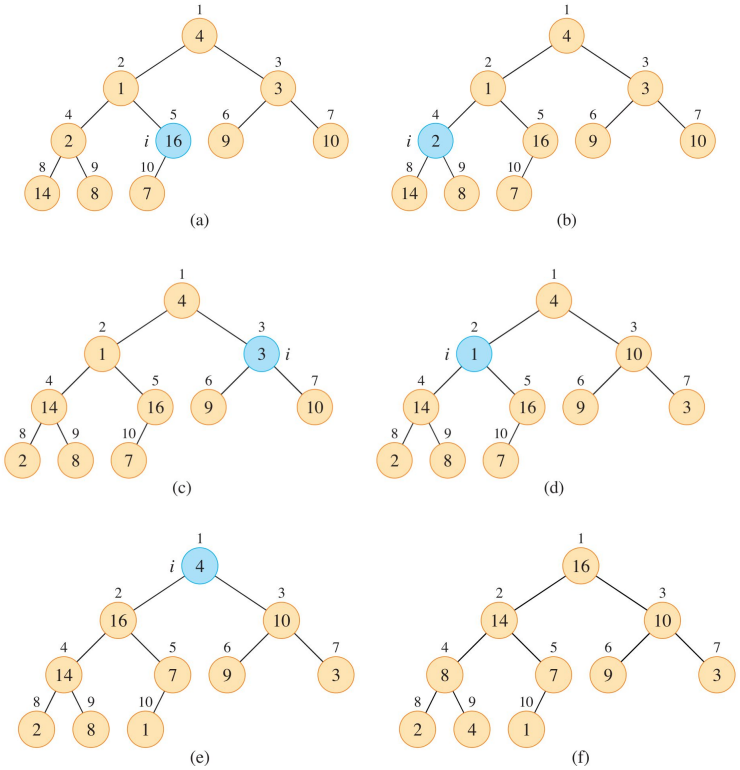
Final recurrence looks like $T(n) \leq T(2n/3) + \Theta(1)$

Solution: $T(n) = O(\lg n)$

Questions

1. A = <27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>. Max-heapify (A, 3)
2. Design min-heapify subroutine along with run-time complexity
3. Effect of calling max-heapify (A, i) for i > A.heap-size/2?
4. Worst-case for max-heapify. Create a max-heap that causes worst-case run-time and explain the run-time.

Building a heap (Build-max-heap)




```
BUILD-MAX-HEAP(A, n)
1  A.heap-size = n
2  for i = ⌊n/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```

Run time of building a heap

Max-heapify = O(log n)

Number of nodes = O(n)

Total cost = O(n log n)

Run time of building a heap

Max-heapify for a node of height h = O(h)

Number of nodes with height h = $\lceil n/2^{h+1} \rceil$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} .$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} .$$

$$\sum_{k=0}^{\infty} k x^k = \frac{x}{(1 - x)^2}$$

for $|x| < 1$.

$$\begin{aligned}
\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil ch
&\le \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{n}{2^h} ch \\
&= cn \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \\
&\le cn \sum_{h=0}^{\infty} \frac{h}{2^h} \\
&\le cn \cdot \frac{1/2}{(1 - 1/2)^2} \\
&= O(n) .
\end{aligned}$$

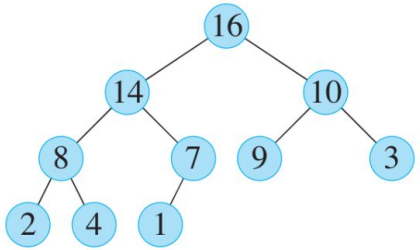
Questions

Why does loop index i in build-max-heap decrease from floor(n/2) to 1 rather than increase from 1 to floor(n/2)?

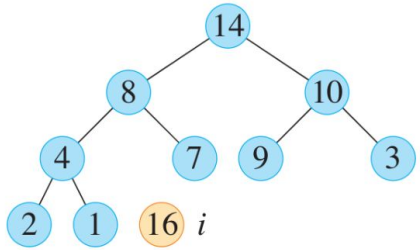
Heapsort Algorithm

```
HEAPSORT(A, n)
1  BUILD-MAX-HEAP(A, n)
2  for i = n downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

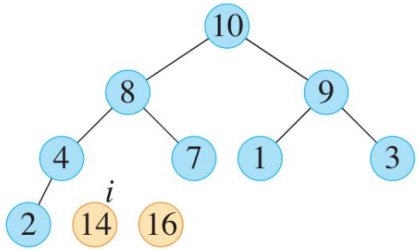
Heap sort



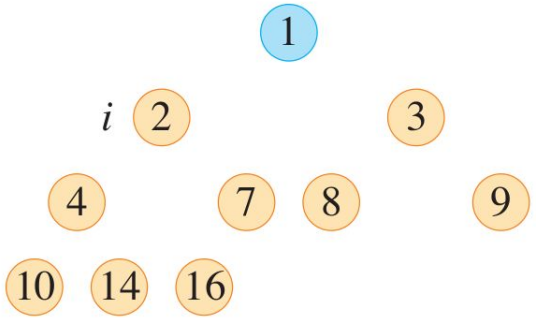
(a)



(b)



(c)



(j)