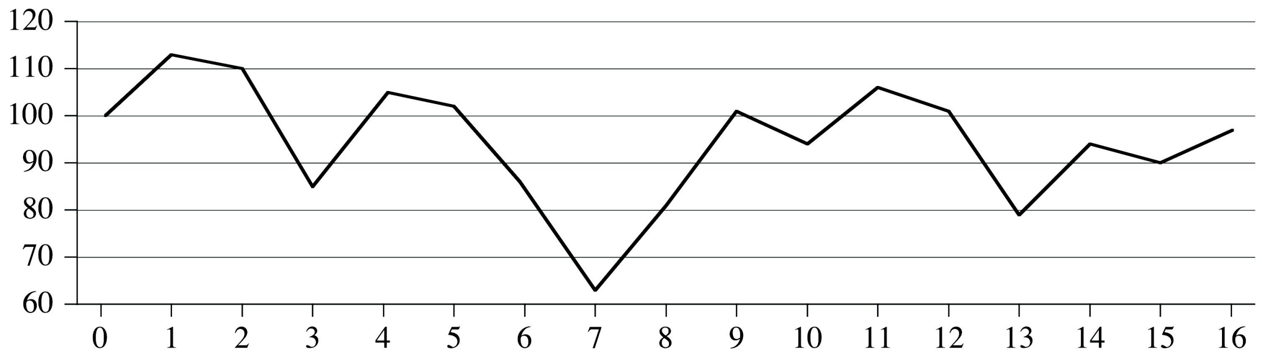
Plan: Heapsort (Ch 6)
Review
Divide and conquer algorithms

- merge sort
- binary search
- power of number
- Fibonacci sequence
- Strassen's algorithm

Topic today: Max subarray and begin order algorithms

Topic 1: Max subarray
Given an array A[1,..., n], find a continuous segment of A such that the sum of that segment is max of any continuous segment in the array.

Application of sub-array problem can be used as how to buy and sell stocks



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

Brute force:
check each sub-array (there are ${}_nC_2$ subarrays. → $\Theta(n^2)$

Divide and conquer:
Idea: divide the array into two halves. Find the max subarray in the first half and second half or the one crossing the middle.. Pick the largest one

Q:/ Any problem with it?

A:/ It is possible that the max subarray can be corssing the middle point.

It turns out that we can find the one crossing the middle with linear time → just expand to the left and right until the sum doesn't increase.

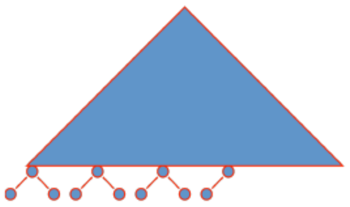
$T(n)=2T(n/2)+ \Theta(n) \rightarrow T(n)$ is $\Theta(n\lg(n))$

2. Heapsort

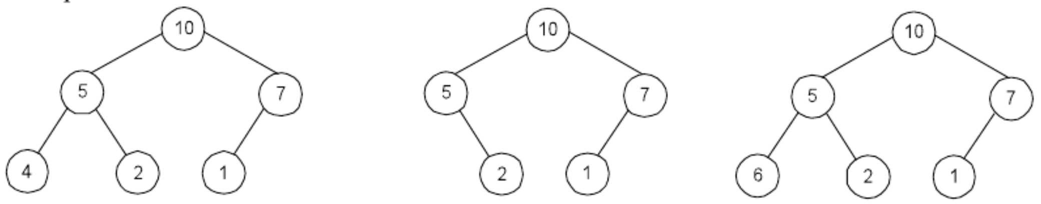
Definition of a heap

A *heap* is a binary tree with keys at its nodes (one key per node) such that:

- It is essentially complete, i.e., all its levels are full except possibly the last level, where only some rightmost leaves may be missing
- The key at each node is ≥ keys at its children



Example



Only the first one is a heap, the second one isn't because it misses one left child and the third one isn't either because it is not sorted vertically.

Note: Heap's elements are ordered top down (along any path down from its root), but they are not ordered left to right. Don't confuse heap with a binary search tree.

Properties

- Given *n,* there exists a unique binary tree with *n* nodes that is essentially complete, with *h* = floor(lg *n*)
- The root contains the largest key
- The subtree rooted at any node of a heap is also a heap
- A heap can be represented as an array in the top-down, left-right fashion
  Example
  n=6

0	1	2	3	4	5	6
	9	6	8	2	5	7

Properties

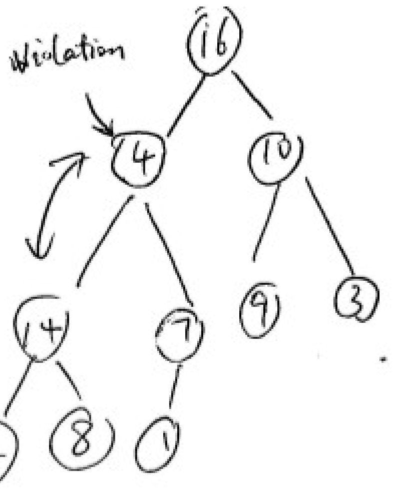
- first floor(n/2) items are parents and remaining items are leaves.
- for the ith item, if it's a parent, its children are in position 2i and 2i+1; if it is a leaf, then its parent is in position floor(i/2)

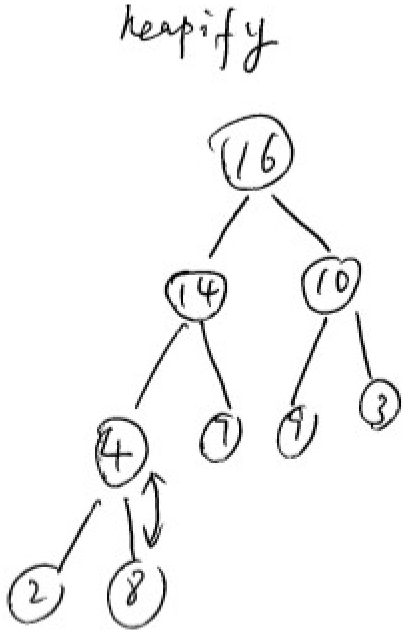
Sometimes, we think that the there is a heap size and there is an array size (basically not all array elements are part of the heap, only the first heap-size elements are part of the heap).

Operation 1: Heapify with one out of place element

Idea: keep exchanging it with its largest child until the heap condition holds

e.g.



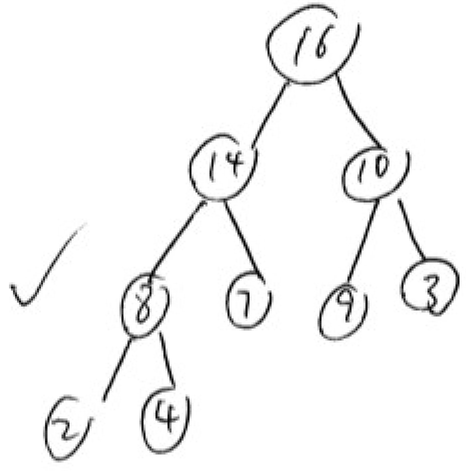


heapify efficiency:

$\le \lg n$

thus

$O(\lg n)$



How to build a heap structure

Step 0: Initialize the structure with keys in the order given

Step 1: heapify from the last parent down to the root.

Step 2: Repeat Step 1 for the preceding parental node

e.g.

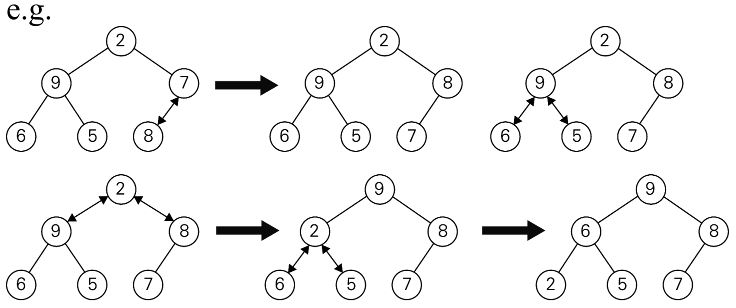


FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8

Using arrays, here are the results

STEPS	0	1	2	3	4	5	6
1		2	9	7	6	5	8
2		2	9	8	6	5	7
3		2	9	8	6	5	7
4		9	2	8	6	5	7
		9	6	8	2	5	7

Efficiency: looks like ϴ(nlgn) but through detailed analysis it is in fact ϴ(n)

How to insert a key into a heap
Step 1: Insert the new element at last position in heap.
Step 2: Compare it with its parent and, if it violates heap condition, exchange them
Step 3: Continue comparing the new element with nodes up the tree until the heap condition is satisfied.
e.g.
Insert 10 into the previous heap

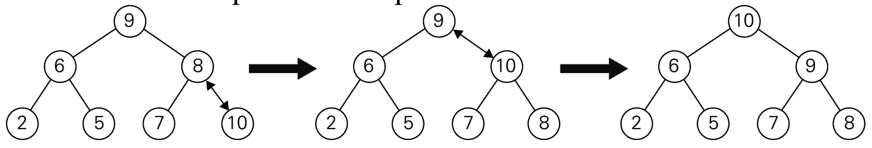


FIGURE 6.12 Inserting a key (10) into the heap constructed in Figure 6.11. The new key is sifted up via a swap with its parent until it is not larger than its parent (or is in the root).

How to remove a root of a heap
Q:/ Why root?
A:/ It is the basis of heap sort.

Step 1: swap the root with the last element
Step 2: remove the last element
Step 3: Swap the new root with its largest children until the condition of heap is satisfied.

e.g

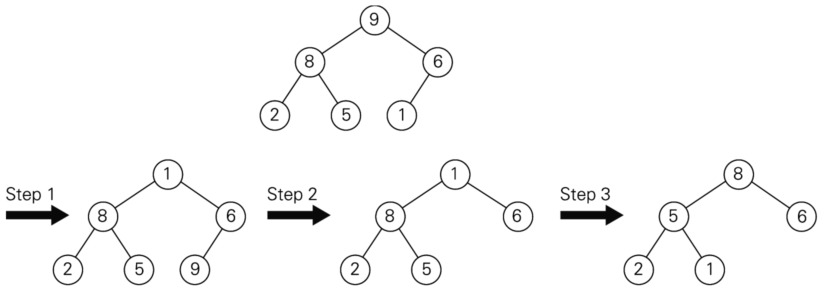


FIGURE 6.13 Deleting the root's key from a heap. The key to be deleted is swapped with the last key, after which the smaller tree is "heapified" by exchanging the new key at its root with the larger key at its children until the parental dominance requirement is satisfied.

Heapsort

Stage 1: Construct a heap for a given list of *n* keys

Stage 2: Repeat operation of root removal *n*-1 times:

- Exchange keys in the root and in the last (rightmost) leaf
- Decrease heap size by 1
- If necessary, swap new root with larger child until the heap condition holds

Baiscally the sorted part increases and the heap shrinks.

e.g.

Sort the list 2, 9, 7, 6, 5, 8 by heapsort

1. Build the heap

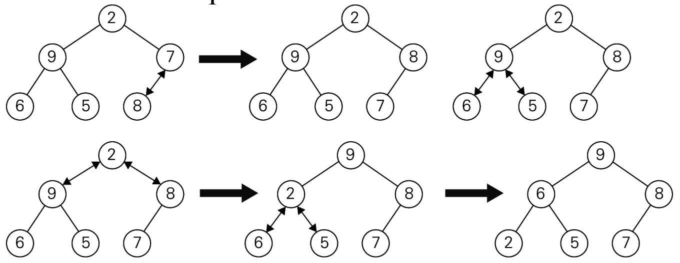


FIGURE 6.11 Bottom-up construction of a heap for the list 2, 9, 7, 6, 5, 8

2. Remove the roots 5 times

Stage 1 (heap construction)	Stage 2 (maximum deletions)
2 9 **7** 6 5 8	**9** 6 8 2 5 7
2 **9** 8 6 5 7	7 6 8 2 5 | **9**
2 9 8 6 5 7	**8** 6 7 2 5
9 **2** 8 6 5 7	5 6 7 2 | **8**
9 6 8 2 5 7	**7** 6 5 2
	2 6 5 | **7**
	6 2 5
	5 2 | **6**
	5 2
	2 | **5**
	2

FIGURE 6.14 Sorting the array 2, 9, 7, 6, 5, 8 by heapsort

Another example: 16 14 10 8 7 9 3 2 4 1

Efficiency of heap sort
Stage 1: Build heap for a given list of n keys
C(n) is O(n)

Stage 2: Repeat operation of root removal n-1 times (fix heap)
C(n) ≤2nlogn

Both worst-case and average-case efficiency: O(nlogn)