Part-D2

Heaps

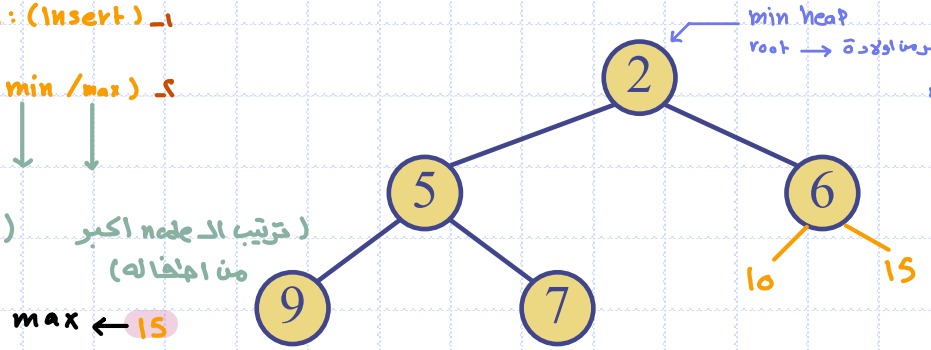
الفكرة انه الـ root
يا اكبر من اولاده او اصغر من اولادة حسب نوعها.

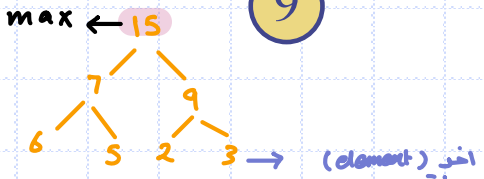
١- (insert) يكون : (left to right) (of the level)

٢- Priority (min / max)

(ترتيب الـ node اكبر من اطفاله)

(ترتيب الـ node اصغر من اطفاله) .





Recall Priority Queue ADT (§ 7.1.3)

- A priority queue stores a collection of entries
- Each **entry** is a pair (key, value)
  - Priority
- Main methods of the Priority Queue ADT
  - insert(k, x)
    inserts an entry with key k and value x
  - removeMin()
    removes and returns the entry with smallest key

- Additional methods
  - min()
    returns, but does not remove, an entry with smallest key
  - size(), isEmpty()
    - size() عدد الـ element
    - isEmpty() ترجع الـ true or false

- Applications:
  - Standby flyers
  - Auctions
  - Stock market

1 2 3 4
highest Priority ... lowest Priority

Recall Priority Queue Sorting (§ 7.1.4)

- We can use a priority queue to sort a set of comparable elements
  - Insert the elements with a series of insert operations
  - Remove the elements in sorted order with a series of removeMin operations
- The running time depends on the priority queue implementation:
  - Unsorted sequence gives selection-sort: $O(n^2)$ time
  - Sorted sequence gives insertion-sort: $O(n^2)$ time
- *Can we do better?*

Algorithm *PQ-Sort*(*S*, *C*)
  Input sequence *S*, comparator *C* for the elements of *S*
  Output sequence *S* sorted in increasing order according to *C*
  P ← priority queue with comparator *C*
  while ¬*S.isEmpty* ()
    e ← *S.remove* (*S. first* ())
    P.insertItem(*e*, *e*)
  while ¬*P.isEmpty*()
    e ← *P.removeMin*()
    S.insertLast(*e*)

the last node of a heap is the rightmost node of maximum depth.

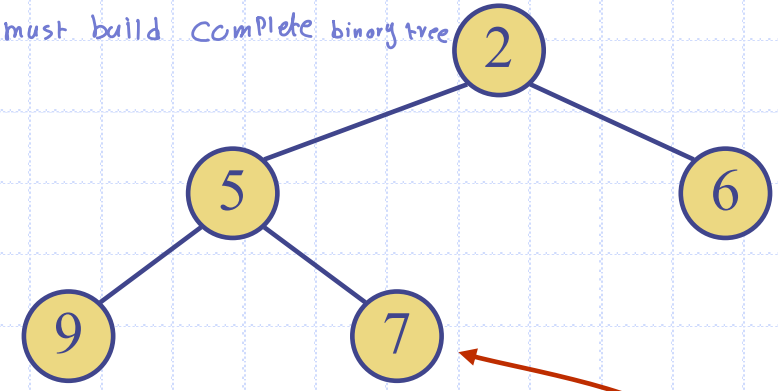
Heaps (§7.3)

first element should be removed + return
Max: remove first
min: remove first

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
  - Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
  - Complete Binary Tree: let h be the height of the heap
    - for $i = 0, \ldots, h-1$, there are 2^i nodes of depth i
    - at depth $h-1$, the internal nodes are to the left of the external nodes

- The last node of a heap is the rightmost node of depth h

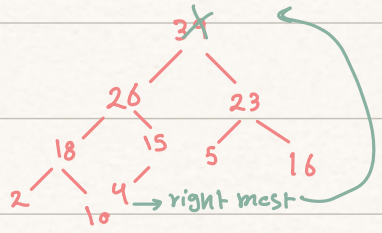
must build complete binary tree

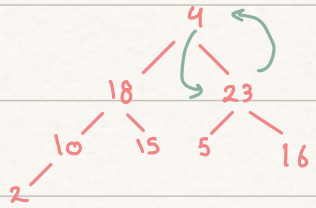


last node
موجود جال right

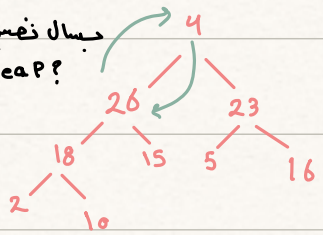
Q1. Illustrate the insertion of the following keys in a heap (max heap) (2, 5, 16, 4, 10, 23, 39, 18, 26, 15).

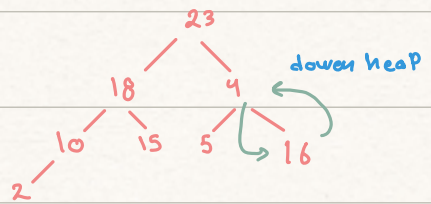
Q2. Illustrate the removal of the following keys from the above heap
39, 26, 23.

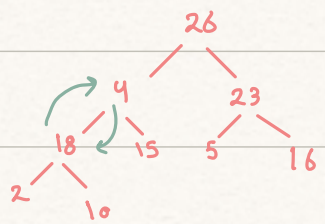


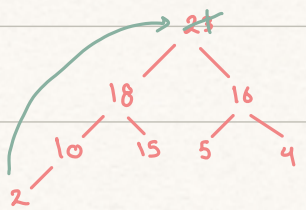


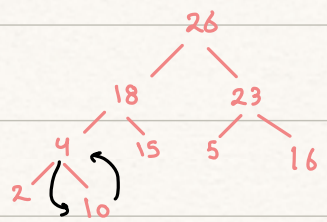
جبال نفس هل مي
max heap?
no.

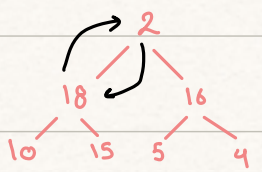


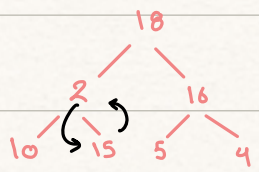


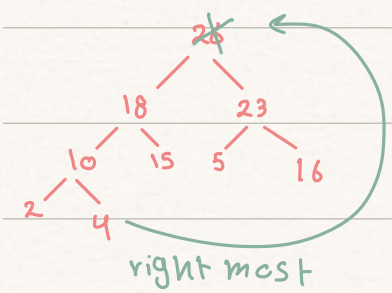


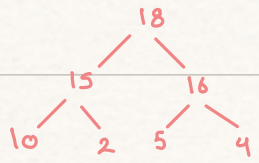




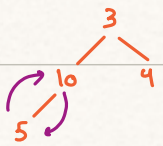


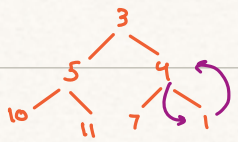


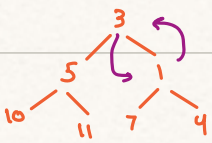


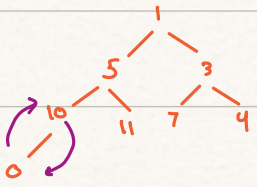


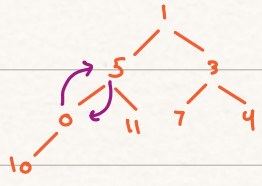
Q3. Illustrate the result of execution of the following integer keys to be inserted in a **priority queue** (min heap). Each node in the min heap stores an integer. 10, 4, 3, 5, 11, 7, 1, 0, 6, 9, 8

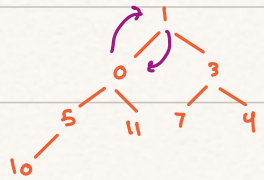


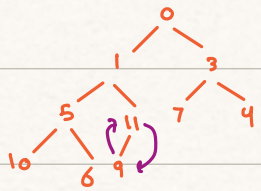


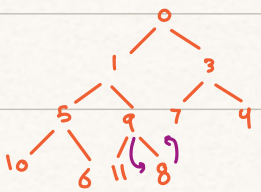




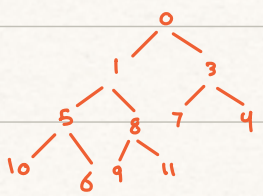








Find min heap

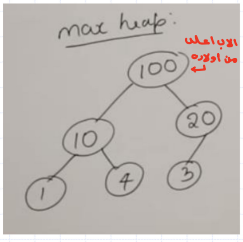


What is a Heap?

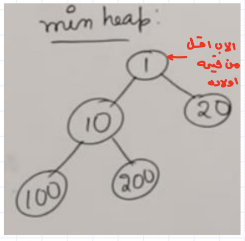
A heap is a binary tree that satisfies these special SHAPE and ORDER properties:

- SHAPE : **Its shape must be a *complete* binary tree:** a binary tree T with *h* levels is *complete* if all levels except possibly the last are completely full, and the last level has all its nodes to the left side. لازم كل الصلايتين ورا بعض والشاغرين ورا بعض
- ORDER: **each node in the heap, the value stored in that node is *smaller than or equal to the value in each of its children.*** اذا ماحكت شي ، min تكون by default

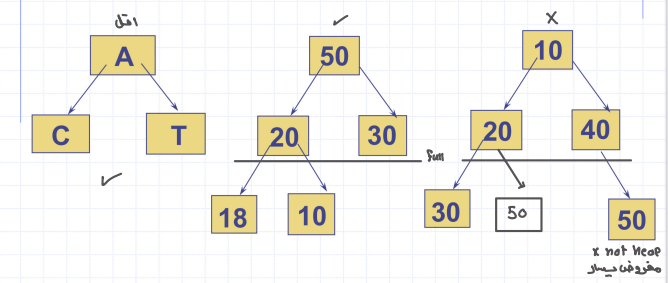
What is a Heap?



not heap كمان وحده شريطة شي، معناته node اذا

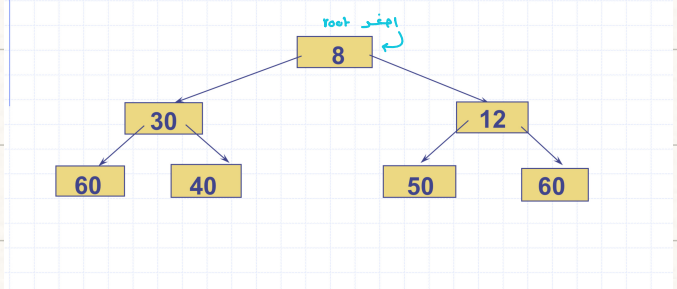


Are these all heaps?



CS 210

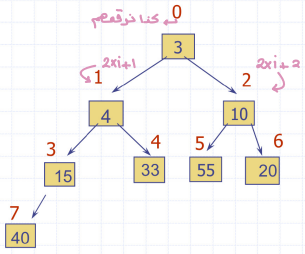
Is this a heap? yes it is



The smallest element in a heap is always found in the root node.

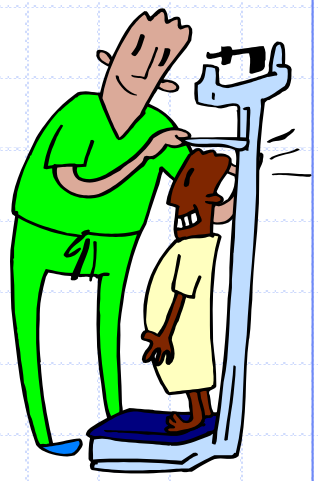
And use the numbers as array indexes to store the tree

[0]	3
[1]	4
[2]	10
[3]	15
[4]	33
[5]	55
[6]	20
[7]	40
[8]	



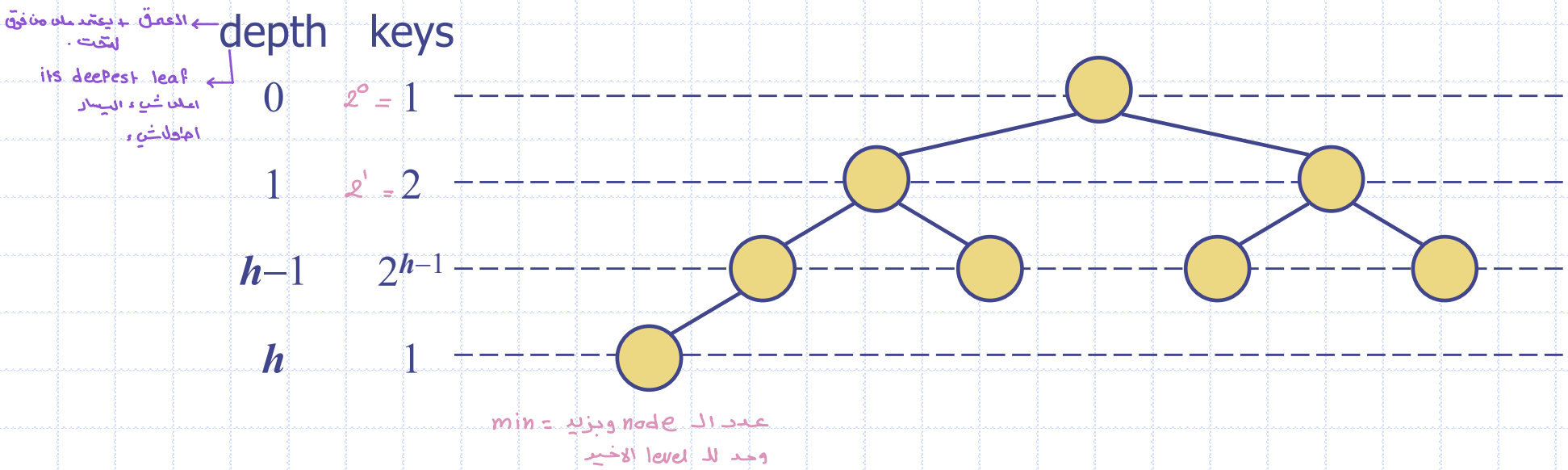
If the root of the tree is in position 0 of the list, then the left and right children of the node in position k are in positions 2k+1 and 2k+2 of the list, respectively. If these positions are beyond the end of the list, then these children do not exist.

Height of a Heap (§ 7.3.1)



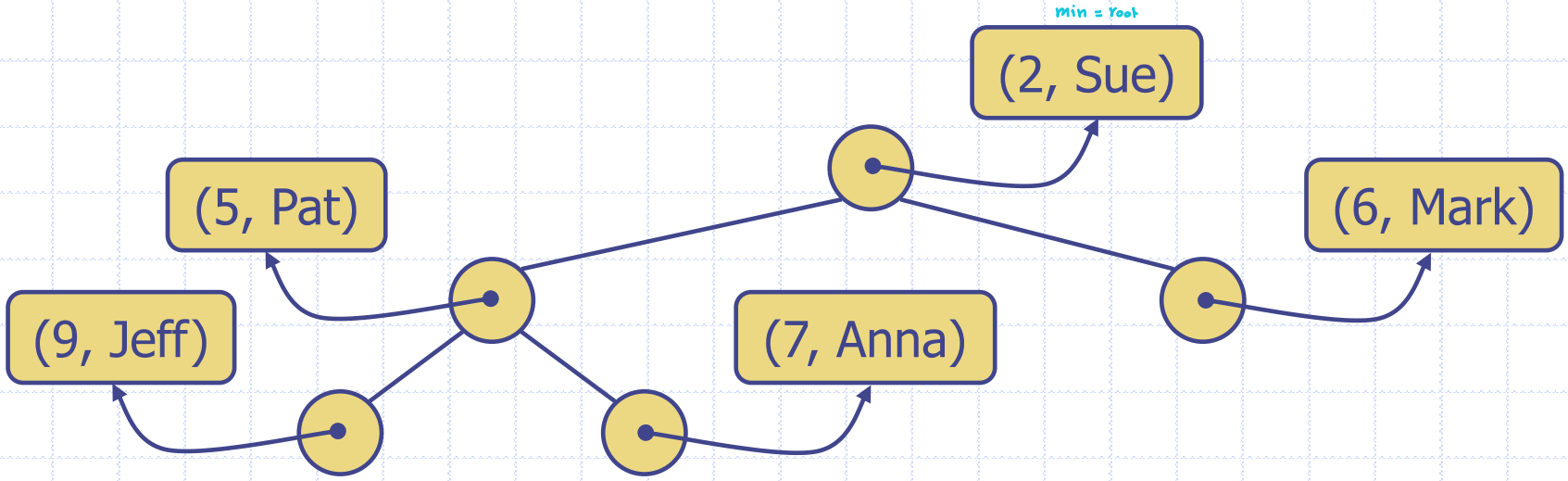
- Theorem: A heap storing n keys has height $O(\log n)$

  Proof: (we apply the complete binary tree property)
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$



Heaps and Priority Queues

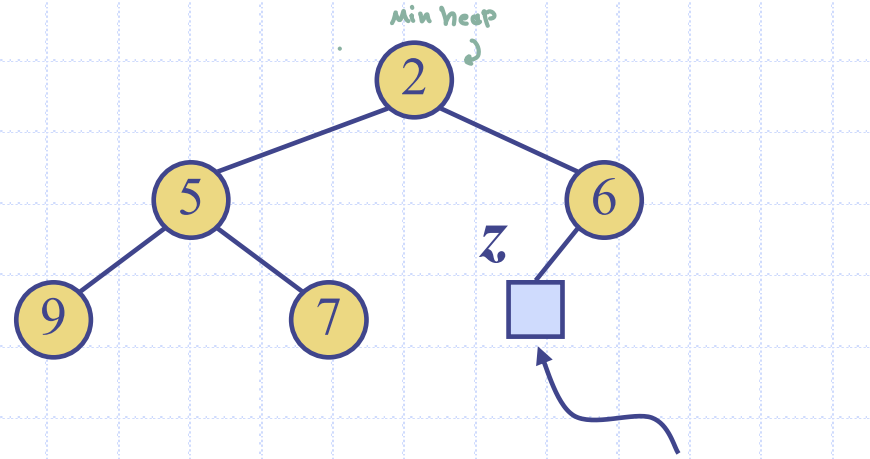
- We can use a heap to implement a priority queue
- We store a (key, element) item at each internal node
- We keep track of the position of the last node
- For simplicity, we show only the keys in the pictures

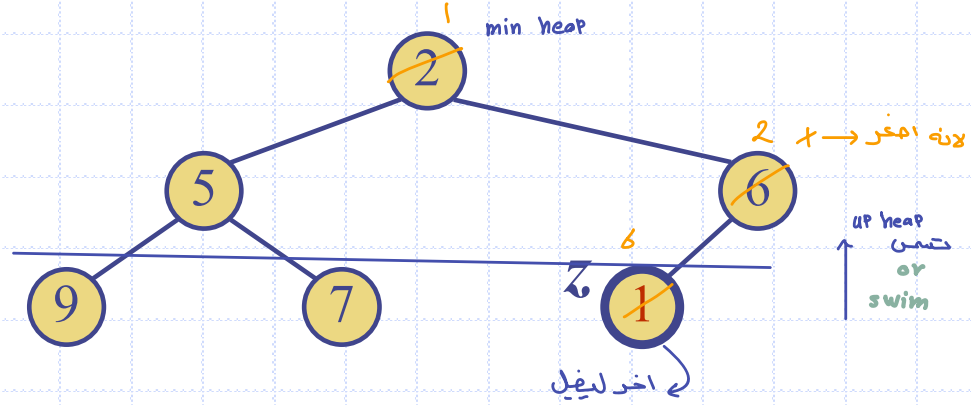


Insertion into a Heap (§ 7.3.3)

دائم بتجيب لاول فاضي Position
باخر (level) .

- Method insertItem of the priority queue ADT corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
  - Store k at z
  - Restore the heap-order property (discussed next)





بعدين نرتب على حسب موقعة

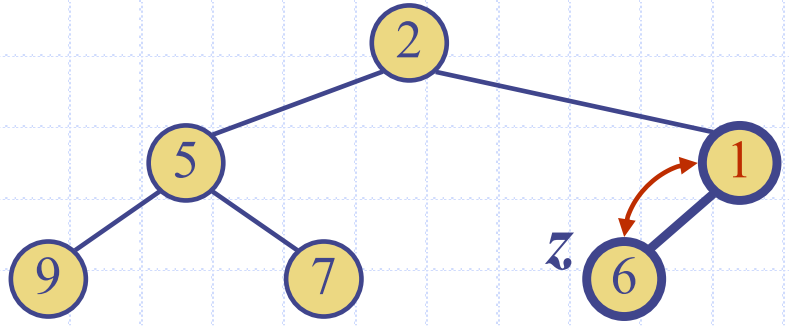
شو قيمه بنسبه للـ Parent تبعة هل بيبدل او لا .

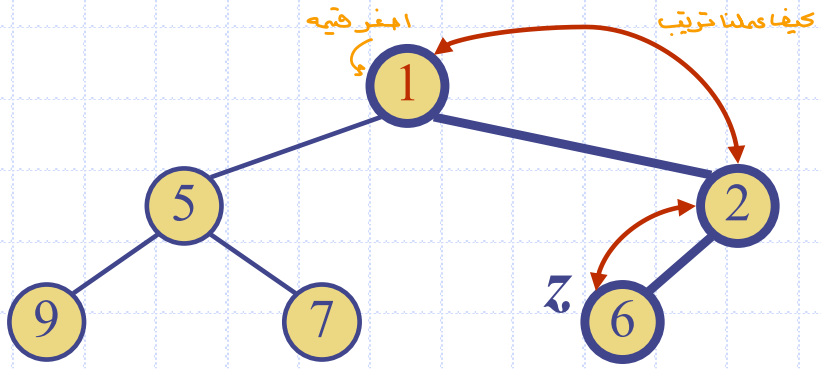
Upheap

من تحت لفوق

- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time

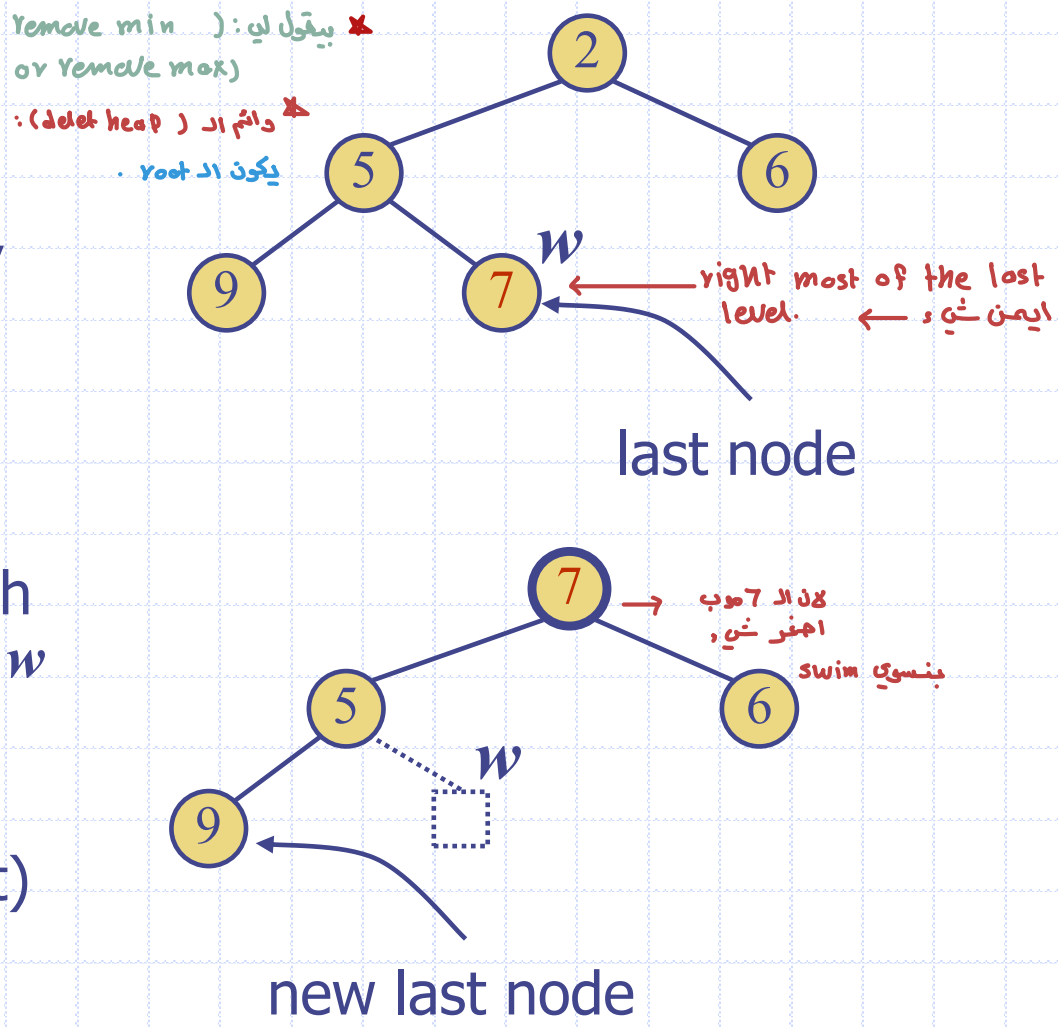
اهتم بالنص



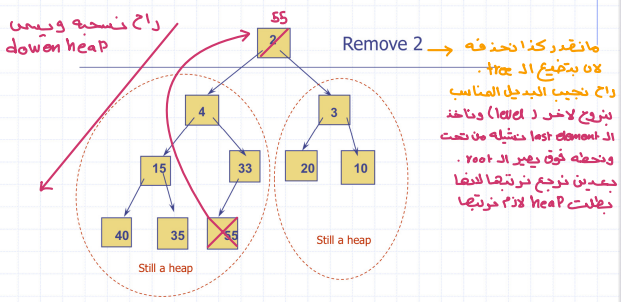


Removal from a Heap (§ 7.3.3)

- Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node w
  - Remove w
  - Restore the heap-order property (discussed next)



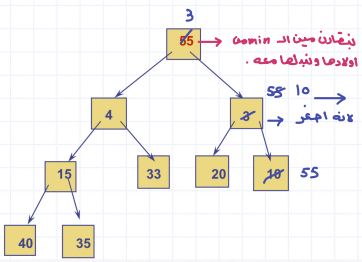
Say we want to remove the element with the smallest value from a heap: removeMin()



راح نسحبه ويمين down heap

ماننقدر كذا نحذفه لان بتضيع الـ tree . راح نجيب البديل المناسب بنروح لاخر (level) وناخذ الـ last element اما نشيله من تحت ونحطه فوق يعبر الـ root . بعدين نرجع نرتبها لانها بطلت heap لازم نرتبها

Solution: DownHeap.

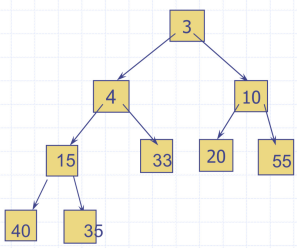


بنقارن مين الـ domin اولادها وبنبدلها معه .

بنقارنها باولادها

55 لان اصغر ١٠

Finally, a restored heap

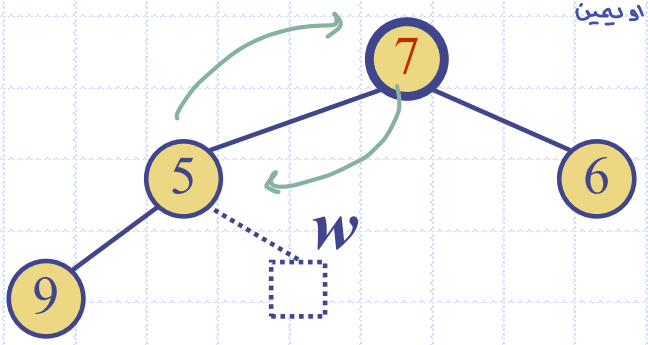


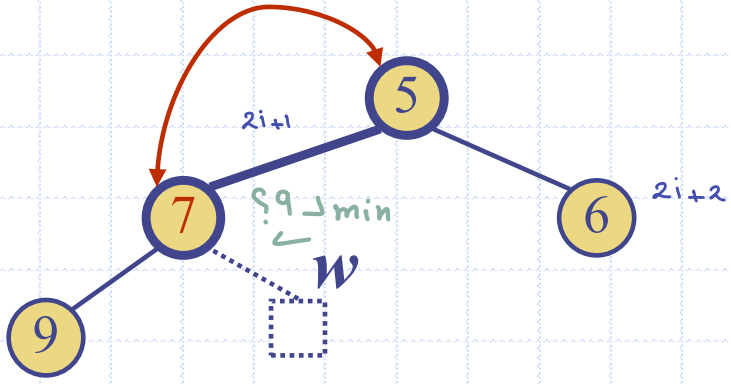
Downheap (remove)

جنزل بنزل اللين ما منوصل لاخر level
بدون ما نلاقي min ثاني .

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Upheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time

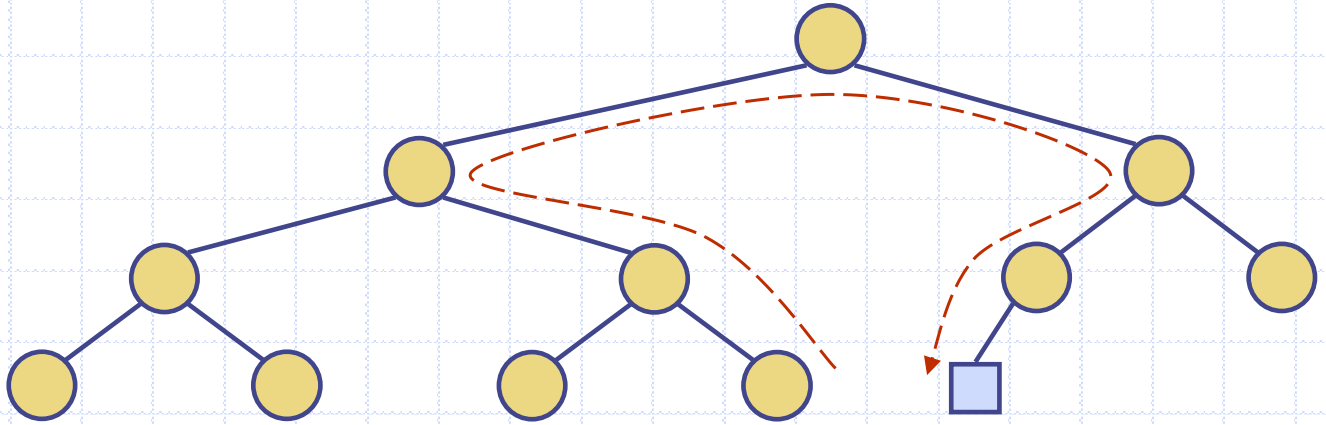
كلا منه بختار يا يسار او يمين
+ شغلي بالنص



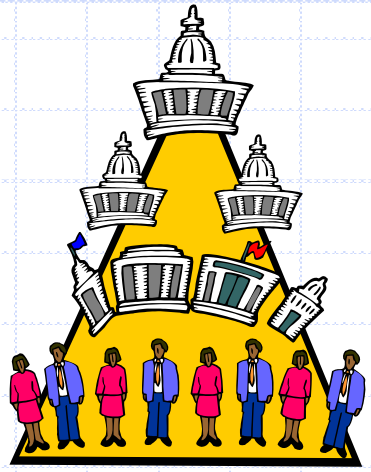


Updating the Last Node

- The insertion node can be found by traversing a path of $O(\log n)$ nodes
  - Go up until a left child or the root is reached
  - If a left child is reached, go to the right child
  - Go down left until a leaf is reached
- Similar algorithm for updating the last node after a removal



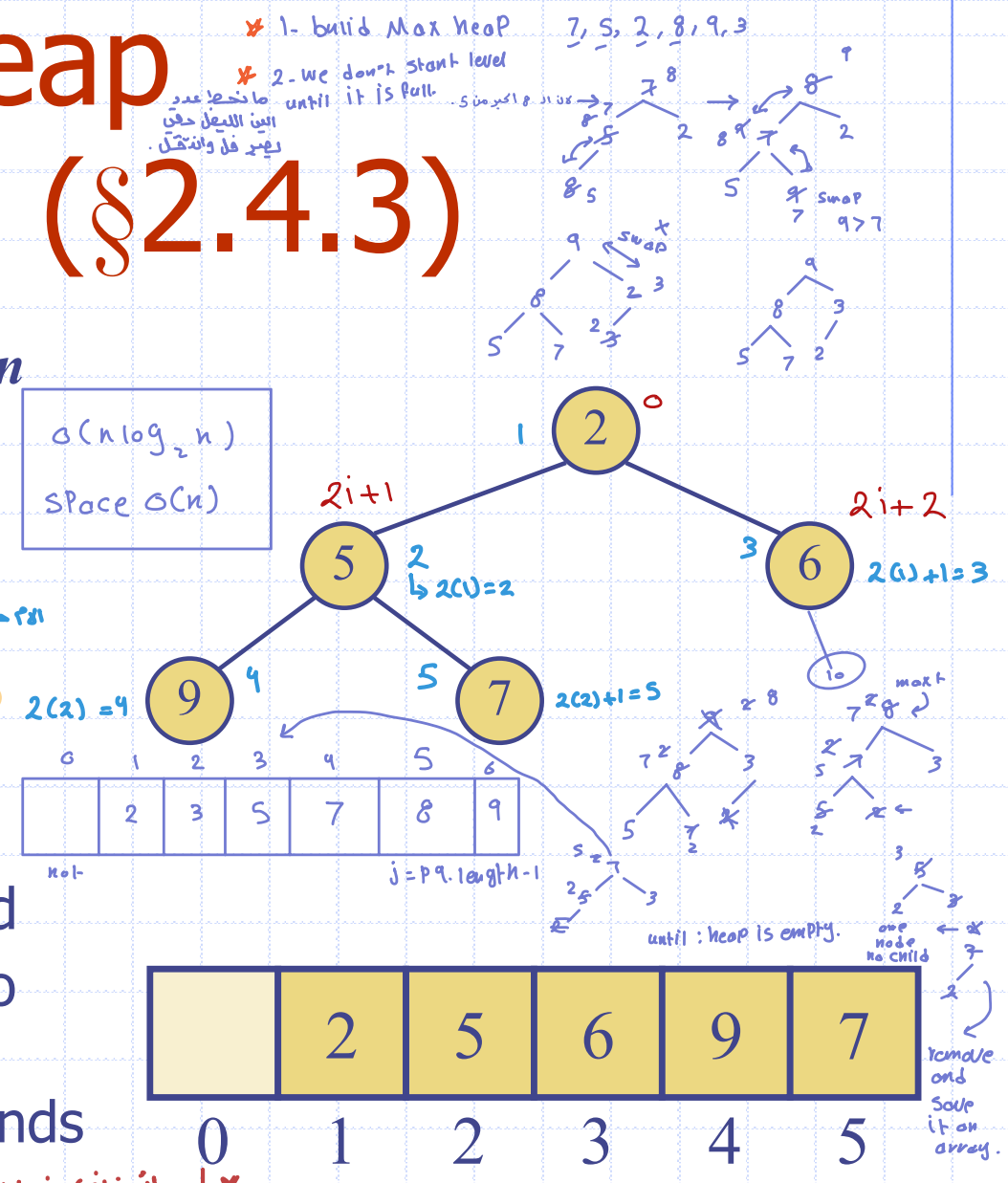
Heap-Sort (§2.4.4)



- Consider a priority queue with n items implemented by means of a heap
  - the space used is $O(n)$
  - methods insert and removeMin take $O(\log n)$ time
  - methods size, isEmpty, and min take time $O(1)$ time
- Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time
- The resulting algorithm is called heap-sort
- Heap-sort is much faster than quadratic sorting algorithms, such as insertion-sort and selection-sort

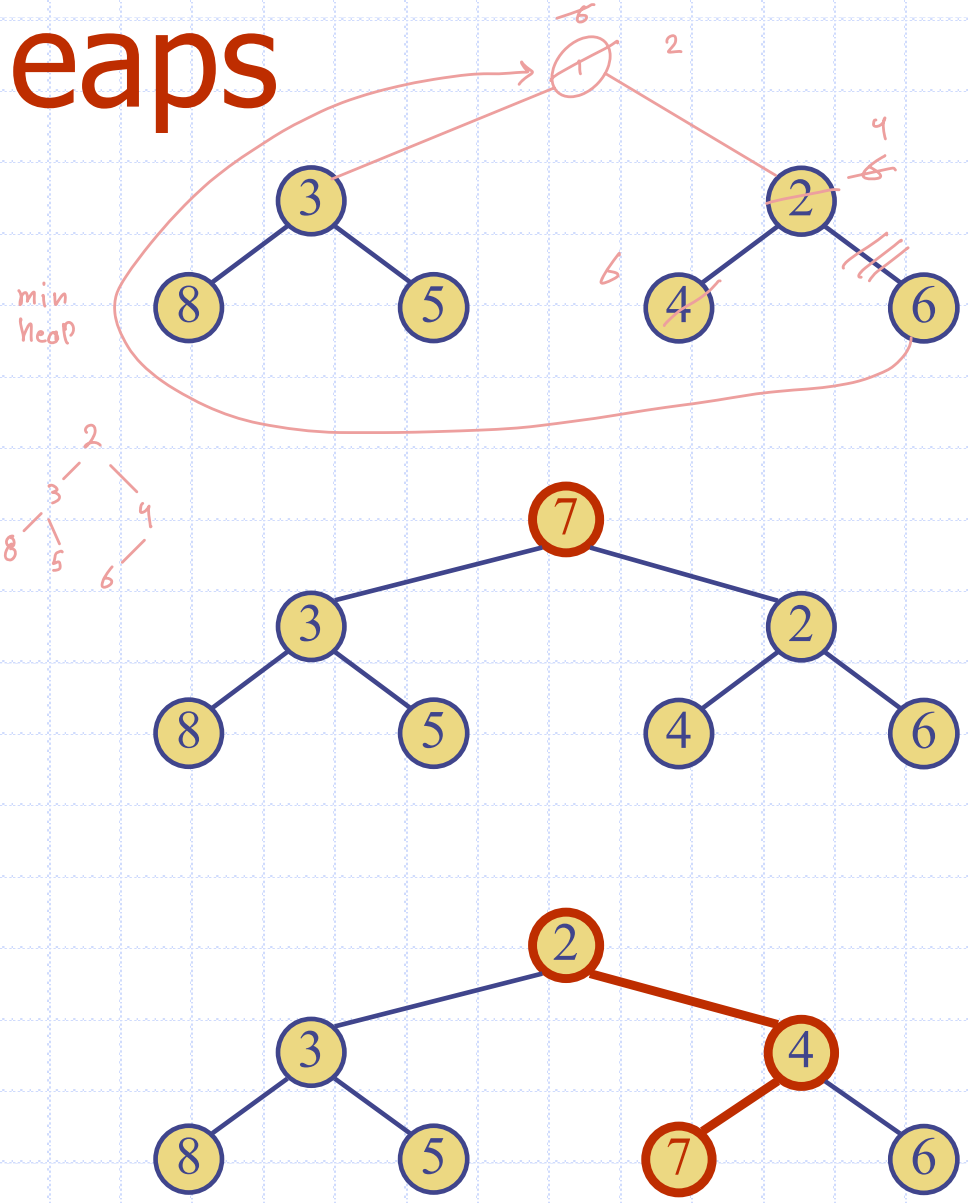
Vector-based Heap Implementation (§2.4.3)

- We can represent a heap with n keys by means of an array of length $n + 1$
- For the node at rank i
  - the left child is at rank $2i$
  - the right child is at rank $2i + 1$
- Links between nodes are not explicitly stored
- The cell of at rank 0 is not used
- Operation insert corresponds to inserting at rank $n + 1$
- Operation removeMin corresponds to removing at rank n
- Yields in-place heap-sort



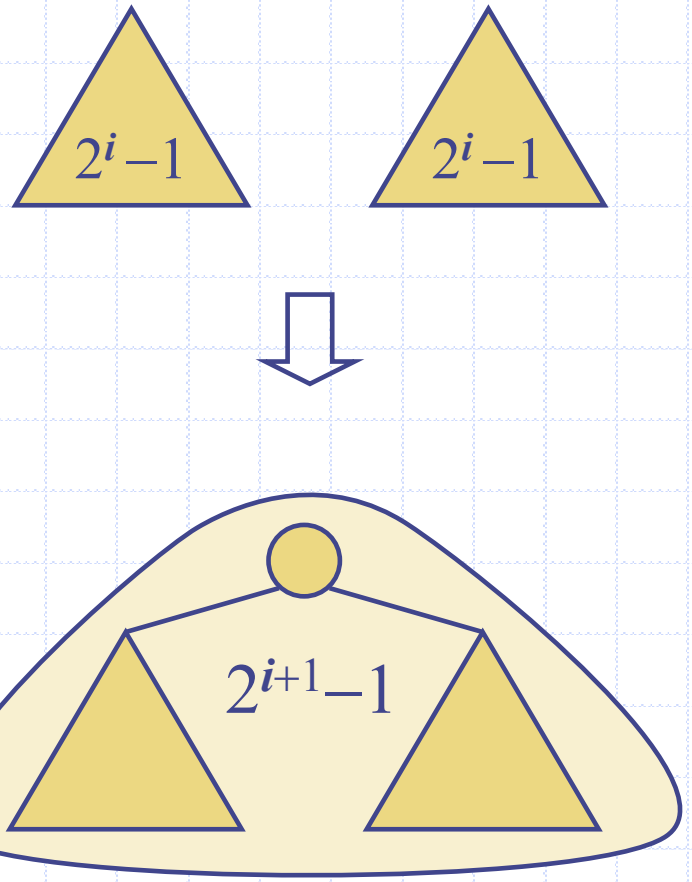
Merging Two Heaps

- We are given two two heaps and a key k
- We create a new heap with the root node storing k and with the two heaps as subtrees
- We perform downheap to restore the heap-order property

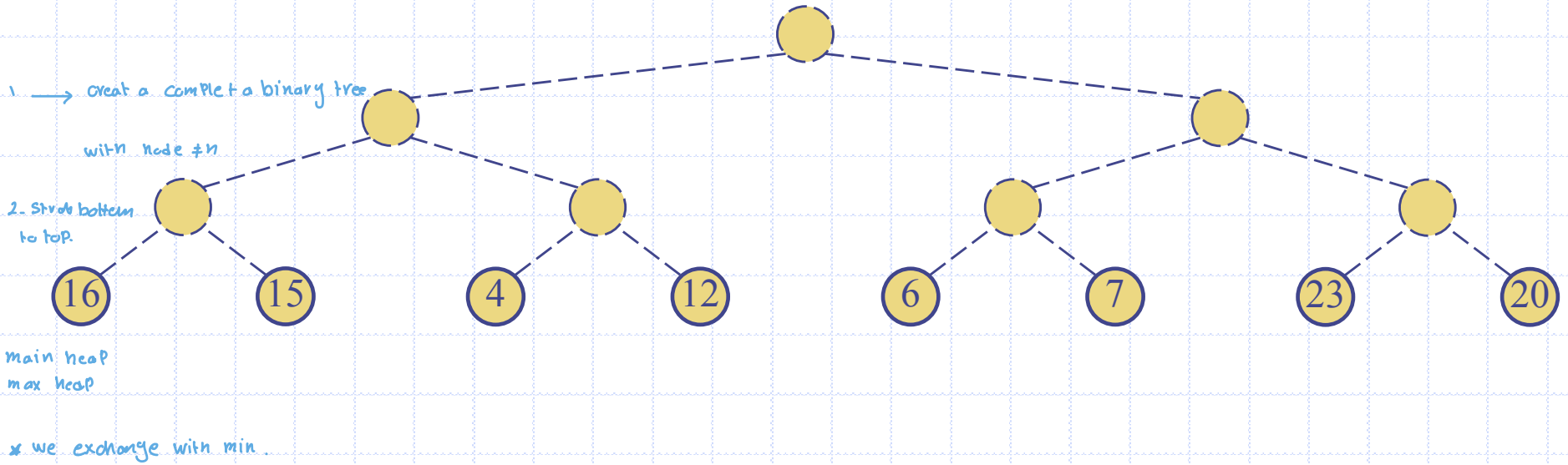


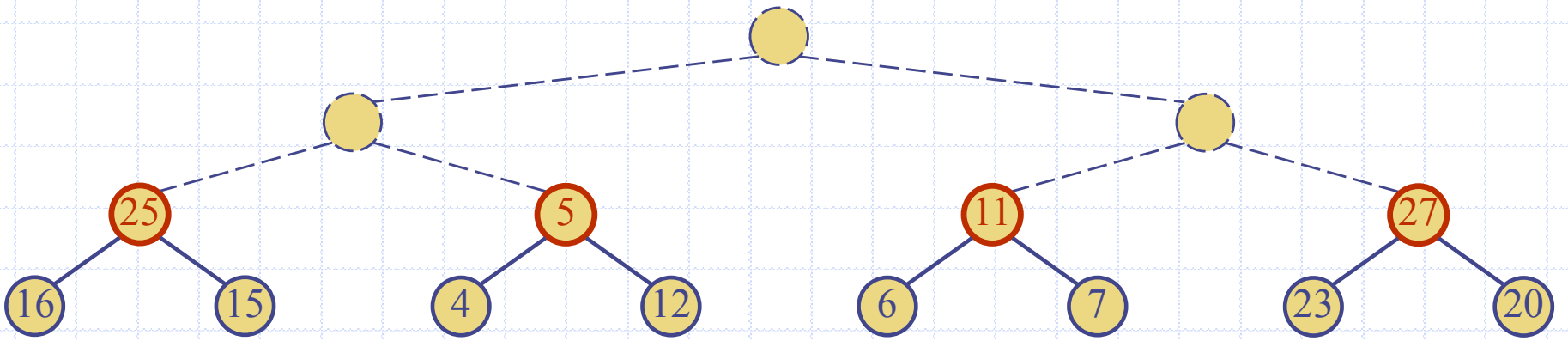
Bottom-up Heap Construction (§2.4.3)

- We can construct a heap storing n given keys in using a bottom-up construction with $\log n$ phases
- In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys

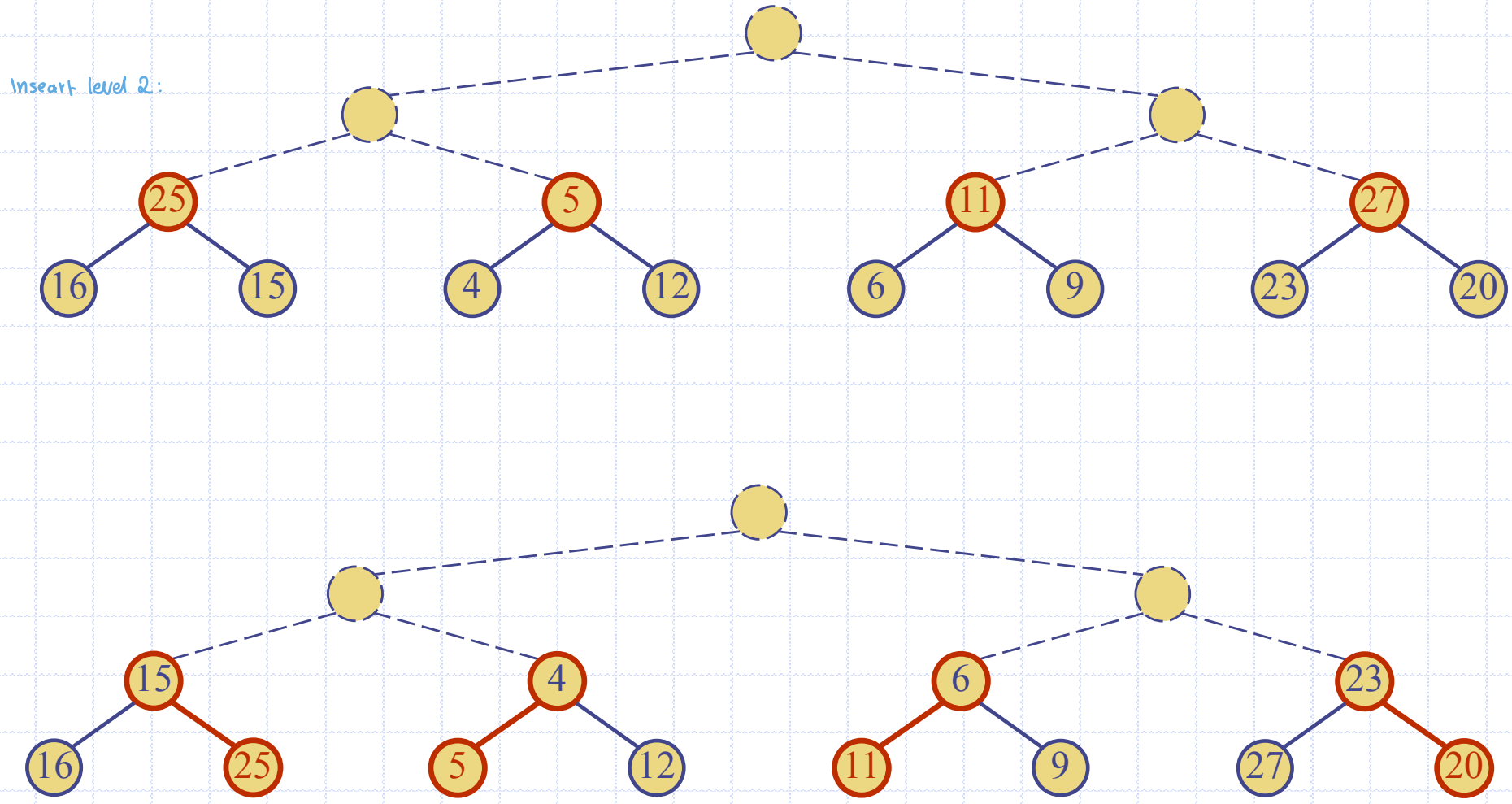


Example

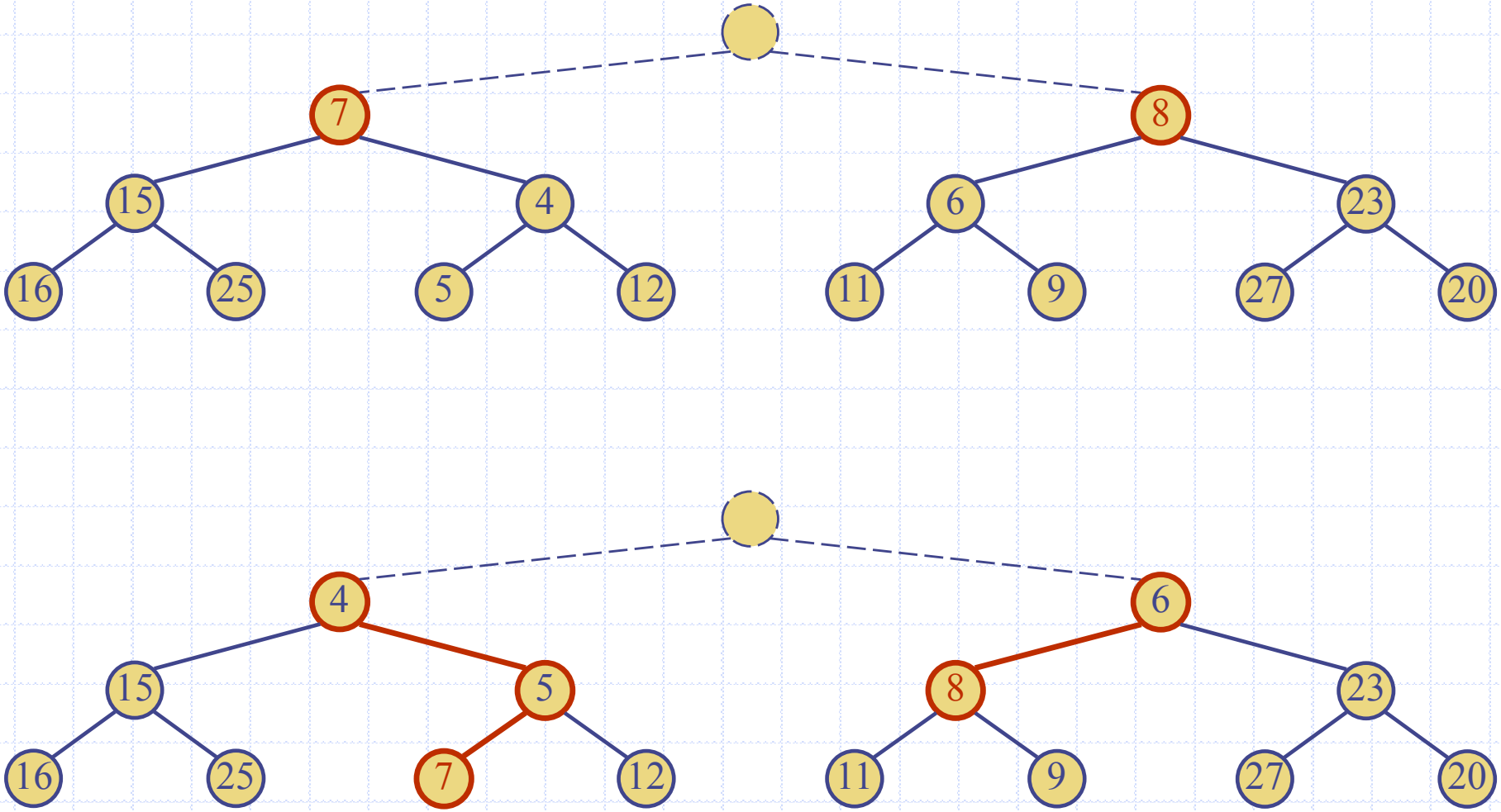

1 ⟶ creat a complete a binary tree
with node ≠n
2- strat bottom
to top.
16
15
4
12
6
7
23
20
main heap
max heap
* we exchange with min.


25
5
11
27
16
15
4
12
6
7
23
20

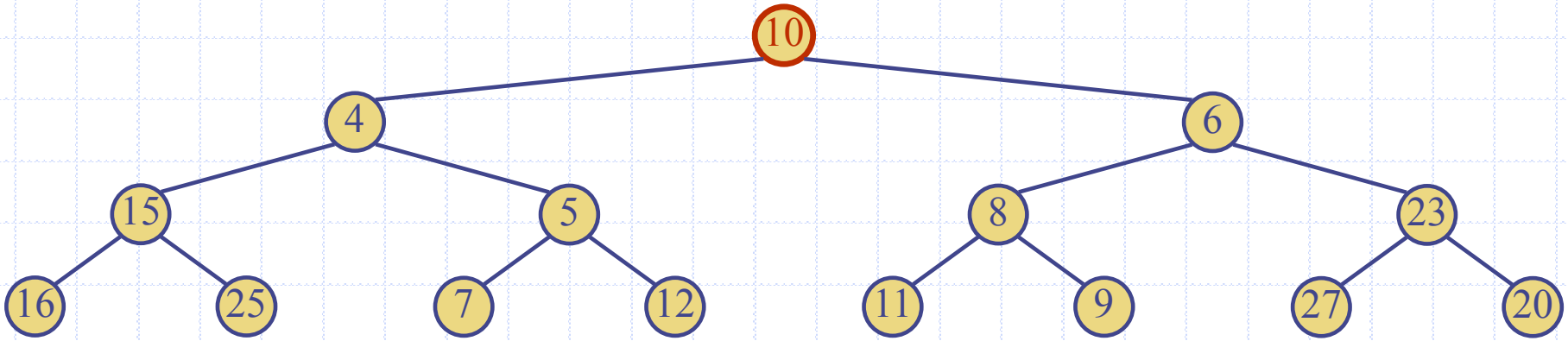
Example (contd.)



Example (contd.)



Example (end)

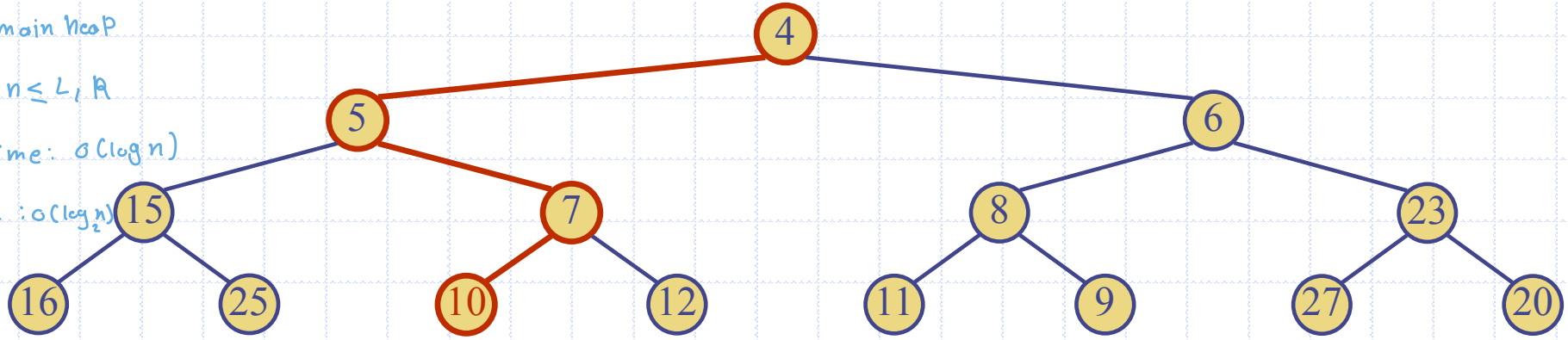


main heap

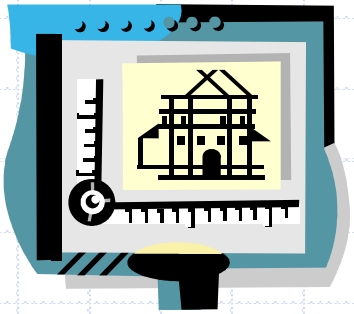
$n \leq L, R$

time: $O(\log n)$

run: $O(\log_2 n)$



Analysis



- We visualize the worst-case time of a downheap with a proxy path that goes first right and then repeatedly goes left until the bottom of the heap (this path may differ from the actual downheap path)
- Since each node is traversed by at most two proxy paths, the total number of nodes of the proxy paths is $O(n)$
- Thus, bottom-up heap construction runs in $O(n)$ time
- Bottom-up heap construction is faster than n successive insertions and speeds up the first phase of heap-sort