The University of Auckland

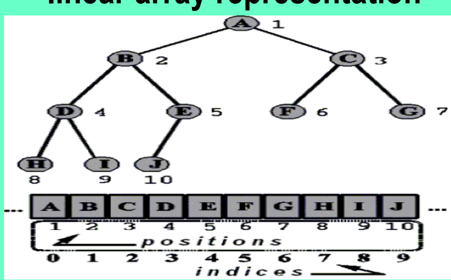
Algorithm HeapSort

- J. W. J. Williams (1964): a special binary tree called **heap** to obtain an $O(n \log n)$ worst-case sorting
- Basic steps:
  - Convert an array into a heap in linear time $O(n)$
  - Sort the heap in $O(n \log n)$ time by deleting n times the maximum item because each deletion takes the logarithmic time $O(\log n)$

Lecture 8 COMPSCI.220.FS.T - 2004 1

The University of Auckland

Complete Binary Tree: linear array representation



Lecture 8 COMPSCI.220.FS.T - 2004 2

The University of Auckland

Complete Binary Tree

- A complete binary tree of the height h contains between 2^h and $2^{h+1}-1$ nodes
- A complete binary tree with the n nodes has the height $\lfloor \log_2 n \rfloor$
- Node positions are specified by the level-order traversal (the root position is 1)
- If the node is in the position p then:
  - the parent node is in the position $\lfloor p/2 \rfloor$
  - the left child is in the position $2p$
  - the right child is in the position $2p + 1$

Lecture 8 COMPSCI.220.FS.T - 2004 3

The University of Auckland

Binary Heap

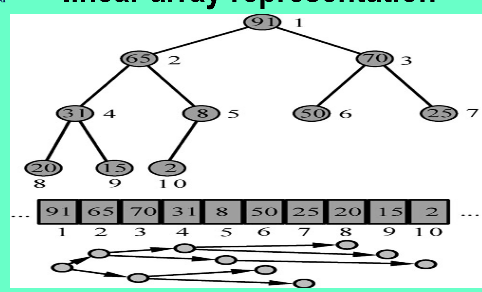
- A heap consists of a complete binary tree of height h with numerical keys in the nodes
- The defining feature of a heap:

  the key of each parent node is **greater than** or **equal to** the key of any child node
- The root of the heap has the maximum key

Lecture 8 COMPSCI.220.FS.T - 2004 4

The University of Auckland

Complete Binary Tree: linear array representation



Lecture 8 COMPSCI.220.FS.T - 2004 5

The University of Auckland

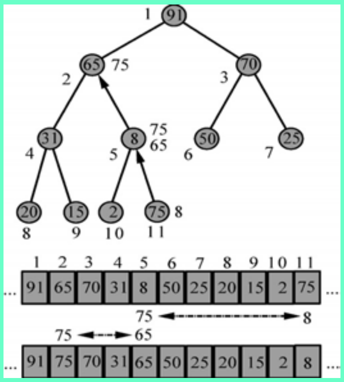
Binary Heap: insert a new key

- Heap of k keys → heap of $k + 1$ keys
- Logarithmic time $O(\log k)$ to insert a new key:
  - Create a new leaf position $k + 1$ in the heap
  - **Bubble** (or **percolate**) the new key up by swapping it with the parent if the latter one is smaller than the new key

Lecture 8 COMPSCI.220.FS.T - 2004 6

The University of Auckland

Binary Heap: an example of inserting a key



Lecture 8 COMPSCI.220.FS.T - 2004 7

The University of Auckland

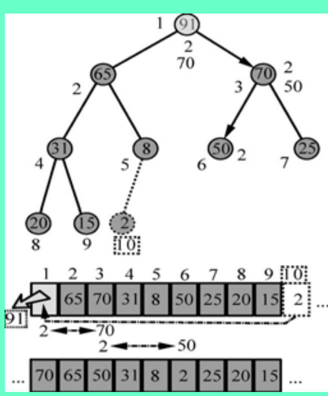
Binary Heap: delete the maximum key

- Heap of k keys → heap of $k - 1$ keys
- Logarithmic time $O(\log k)$ to delete the root (or maximum) key:
  - Remove the root key
  - Delete the leaf position k and move its key into the root
  - Bubble (percolate) the root key down by swapping with the largest child if the latter one is greater

Lecture 8 COMPSCI.220.FS.T - 2004 8

The University of Auckland

Binary Heap: an example of deleting the maximum key

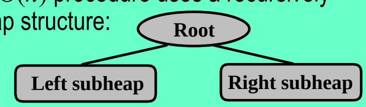


Lecture 8 COMPSCI.220.FS.T - 2004 9

The University of Auckland

Linear Time Heap Construction

- n insertions take $O(n \log n)$ time.
- Alternative $O(n)$ procedure uses a recursively defined heap structure:

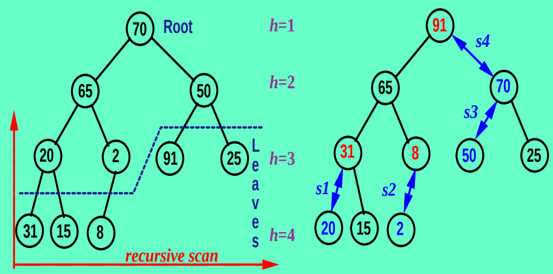


  - form recursively the left and right subheaps
  - percolate the root down to establish the heap order everywhere

Lecture 8 COMPSCI.220.FS.T - 2004 10

The University of Auckland

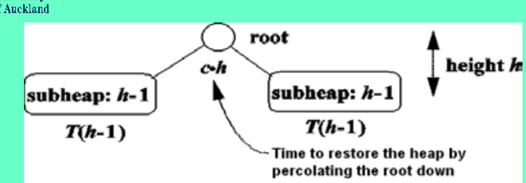
Heapifying Recursion



Lecture 8 COMPSCI.220.FS.T - 2004 11

The University of Auckland

Time to build a heap



$$T(h) = 2\,T(h-1) + c \cdot h; \qquad T(0) = 0$$

$$\rightarrow \quad T(h) = c \cdot \left(2^{h+1} - h - 1\right)$$

Lecture 8 COMPSCI.220.FS.T - 2004 12

Linear time heap construction: non-recursive procedure

- Nodes are percolated down in reverse level order
- When the node p is processed, its descendants will have been already processed.
- Leaves need not to be percolated down.
- Worst-case time $\mathrm{T}(h)$ for building a heap of height h:

  $\mathrm{T}(h) = 2\mathrm{T}(h-1) + ch \rightarrow \mathrm{T}(h) = \mathrm{O}(2^h)$
  - Form two subheaps of height $h-1$
  - Percolate the root down a path of length at most h

Lecture 8 COMPSCI.220.FS.T - 2004 13

Time to build a heap

- A heap of the height h has $n = 2^{h-1}\ldots 2^h - 1$ nodes so that the height of the heap with n items: $h = \lceil \log_2 n \rceil$
- Thus, $\mathrm{T}(h) = \mathrm{O}(2^h)$ yields the linear time $\mathrm{T}(n) = \mathrm{O}(n)$

$$\int_0^N (N-x)e^x dx = e^N - N - 1$$

$$\int_0^N xe^x dx = (N-1)e^N + 1$$

Two integral relationships helping to derive the above (see Slide 12) and the like discrete formulas

Lecture 8 COMPSCI.220.FS.T - 2004 14

Time to build a heap

$$\mathrm{T}(h) = 2\,\mathrm{T}(h-1) + c \cdot h$$

$$2\,\mathrm{T}(h-1) = 2^2\,\mathrm{T}(h-2) + 2c \cdot (h-1)$$

$$\ldots \quad \ldots \quad \ldots$$

$$2^{h-2}\,\mathrm{T}(2) = 2^{h-1}\,\mathrm{T}(1) + 2^{h-2} c \cdot 2$$

$$2^{h-1}\,\mathrm{T}(1) = 2^h\,\mathrm{T}(0) + 2^{h-1} c \cdot 1 = 2^{h-1} c \cdot 1$$

$$\mathrm{T}(h) = c \cdot \left(1 \cdot 2^{h-1} + 2 \cdot 2^{h-2} + \ldots + (h-2) \cdot 2^2 + (h-1) \cdot 2^1 + h \cdot 2^0\right)$$

$$= c \cdot \left(2^{h+1} - h - 1\right)$$

Lecture 8 COMPSCI.220.FS.T - 2004 15

Steps of HeapSort

$^p/_i$	$^1/_0$	$^2/_1$	$^3/_2$	$^4/_3$	$^5/_4$	$^6/_5$	$^7/_6$	$^8/_7$	$^9/_8$	$^{10}/_9$
a	70	65	50	20	2	91	25	31	15	8
HEAPIFY					8					2
				31				20		
			91			50				
	91		70							
h	**91**	**65**	**70**	**31**	**8**	**50**	**25**	**20**	**15**	**2**

Lecture 8 COMPSCI.220.FS.T - 2004 16

Steps of HeapSort

a_1	2	65	70	31	8	50	25	20	15	**91**
Restore the heap (R.h.)	**70**		**2**							
			50			**2**				
H_9	70	65	50	31	8	2	25	20	15	
a_2	**15**	65	50	31	8	2	25	20	**70**	**91**
R.h.	**65**	**15**								
		31		**15**						
				20				**15**		
h_8	65	31	50	20	8	2	25	15		

Lecture 8 COMPSCI.220.FS.T - 2004 17

Steps of HeapSort

a_3	**15**	31	50	20	8	2	25	**65**	**70**	**91**
R.h.	**50**		**15**							
			25				**15**			
h_7	50	31	25	20	8	2	15			
a_4	**15**	31	25	20	8	2	**50**	**65**	**70**	**91**
R.h.	**31**	**15**								
		20		**15**						
h_6	31	20	25	15	8	2				

Lecture 8 COMPSCI.220.FS.T - 2004 18

The University of Auckland

Steps of HeapSort

a_5	2	20	25	15	8	**31**	**50**	**65**	**70**	**81**
R. h.	25		2							
h_5	25	20	2	15	8					
a_6	8	20	2	15	**25**	**31**	**50**	**65**	**70**	**81**
R. h.	20	8								
		15		8						
h_4	20	15	2	8						

Lecture 8 COMPSCI.220.FS.T - 2004 19

The University of Auckland

Steps of HeapSort

a_7	8	15	2	**20**	**25**	**31**	**50**	**65**	**70**	**81**
R. h.	15	8								
h_3	15	8	2							
a_8	2	8	**15**	**20**	**25**	**31**	**50**	**65**	**70**	**81**
R. h.	8	2								
h_2	8	2								
a_9	2	**8**	**15**	**20**	**25**	**31**	**50**	**65**	**70**	**81**

sorted array

Lecture 8 COMPSCI.220.FS.T - 2004 20