Data Structures:
Sorting: Heap Sort, Radix Sort

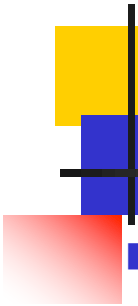
Won Kim
(Lecture by Youngmin Oh)
Spring 2022

Heap Sort

Heap Sort

- Use of a Heap
  - Max Heap or Min Heap
- “Min-Max” Sort
- Supplementary Reading
  - http://en.wikipedia.org/wiki/Heapsort

A Heap



Priority Queue

- A queue in which every element is associated with a “priority”
- Stack
  - Each inserted element has a higher priority
- Queue
  - Each inserted element has a lower priority

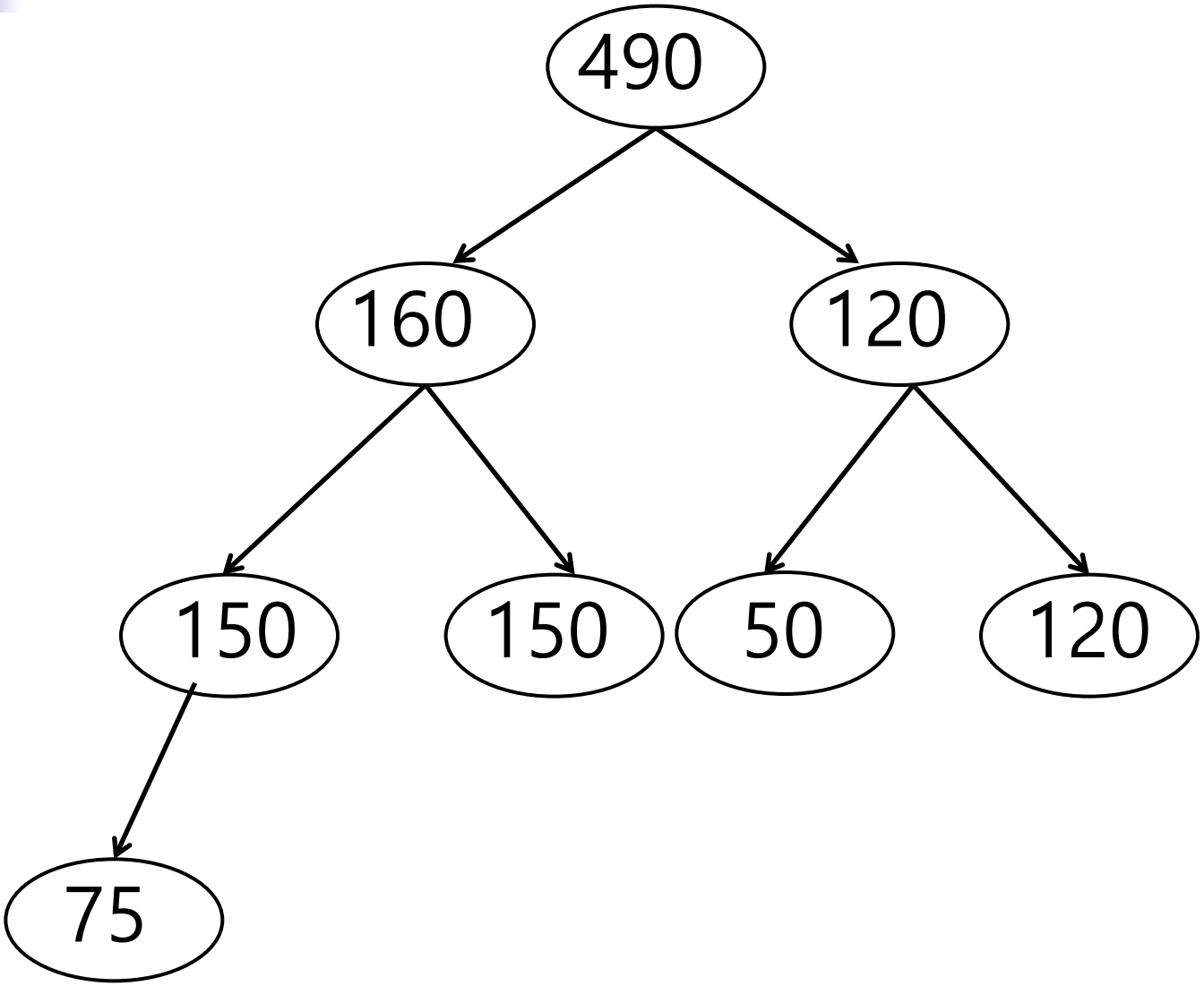
Priority Queue

- Implementations
  - (naive and inefficient) unsorted array
  - multiple queues (one for each priority)
  - (usually) heap
- Operations
  - insert (an element with a priority)
  - remove the highest priority element
    - Pop (get max, get front)
  - peek (just determine the highest priority element)

Max Heap

- "Complete Binary Tree + Max Tree"
- Max Tree
  - The key in each node is not smaller than any key in its left and right subtrees
  - Not a binary search tree
- Operations
  - Insert a new node
  - Delete the largest node
  - "No search"
- Insertion and deletion require restructuring to maintain the max heap properties.
- May be used to implement a priority queue or heap sort

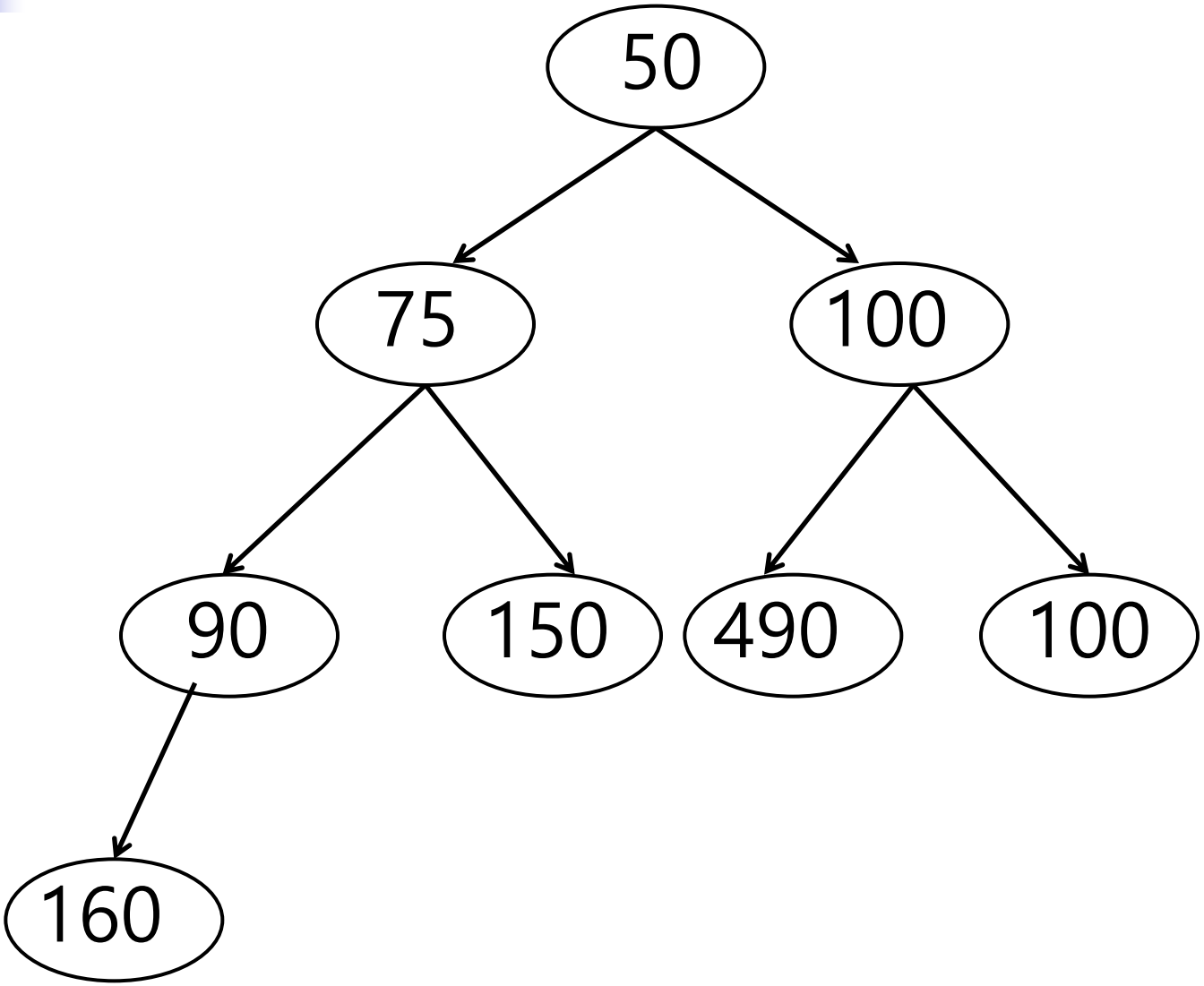
Max Heap: Example



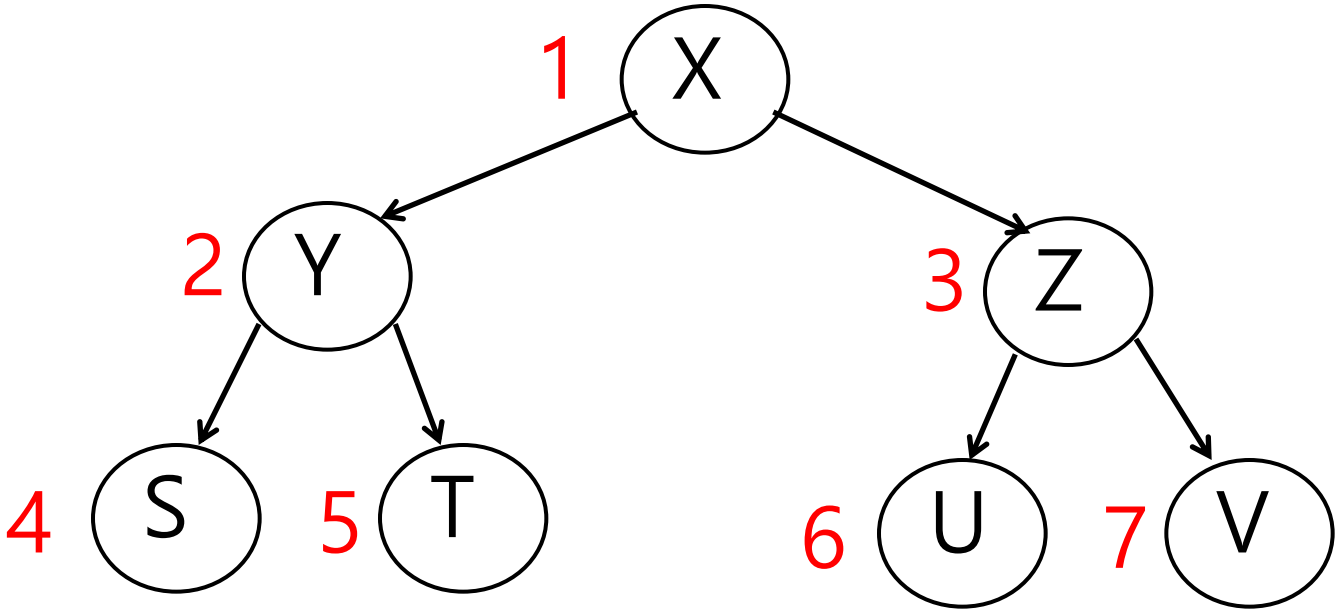
Min Heap

- Complete Binary Tree + Min Tree
- Min Tree
  - The key in each node is not larger than any key in its left and right subtrees
- Operations
  - Insert a new node
  - Delete the smallest node

Min Heap: Example



Implementing a Heap Using an Array



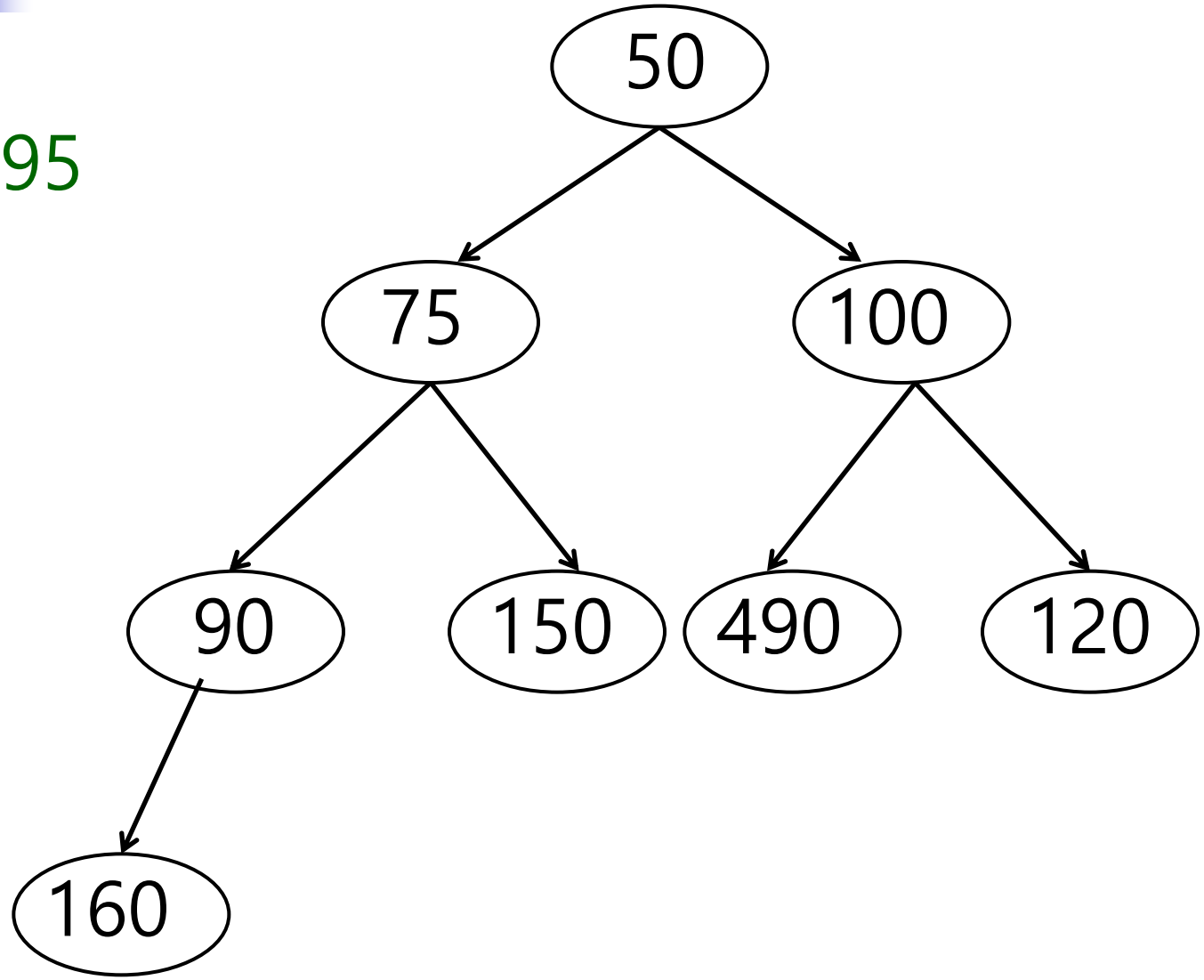
1	X
2	Y
3	Z
4	S
5	T
6	U
7	V

Computing the Locations of Nodes

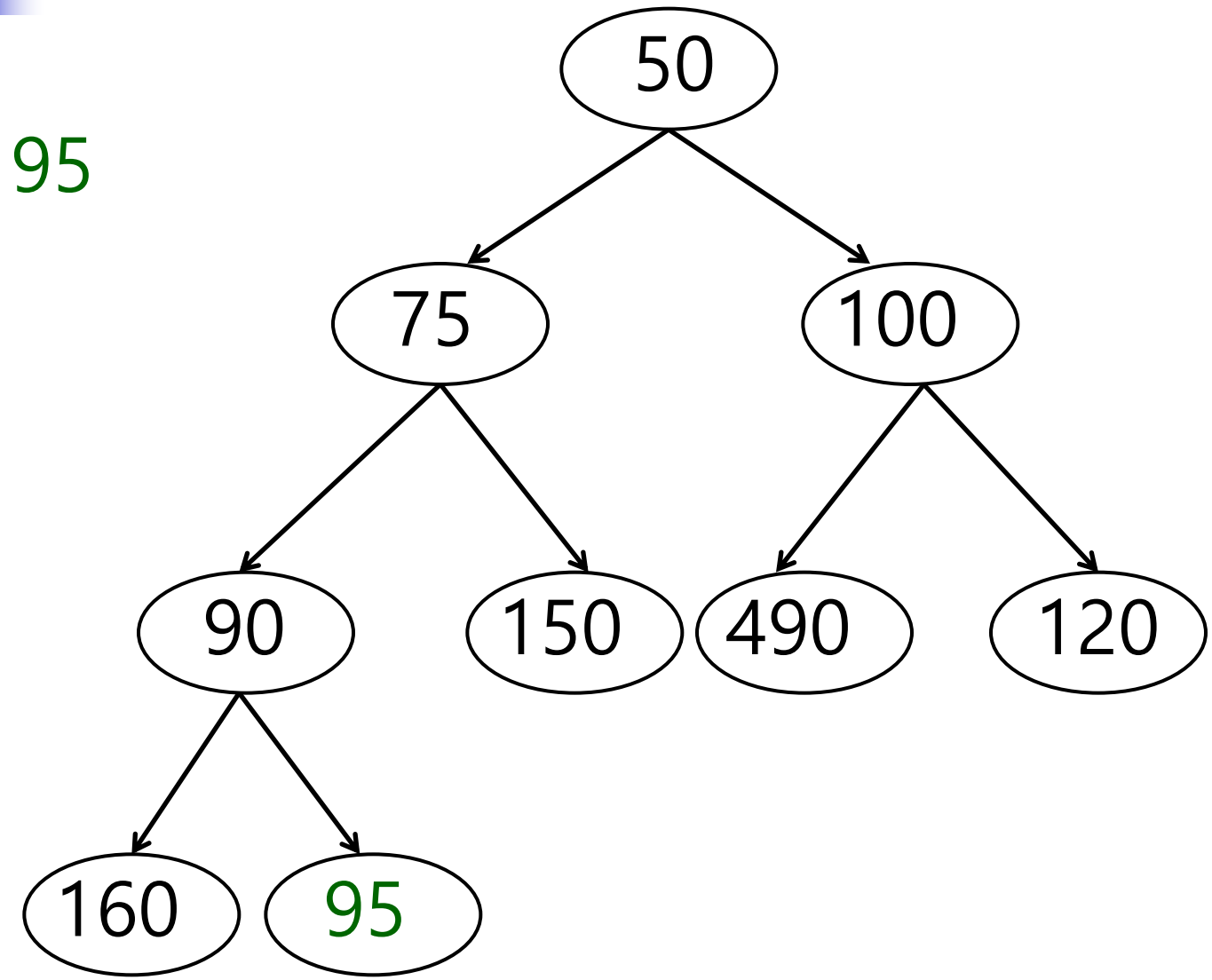
- For a node with array index i on a full binary tree with n nodes
  - Parent of i = $\lceil i/2 \rceil$
  - Left child of i = 2i
  - Right child of i = 2i + 1

Inserting into a Min Heap: Example 1

95

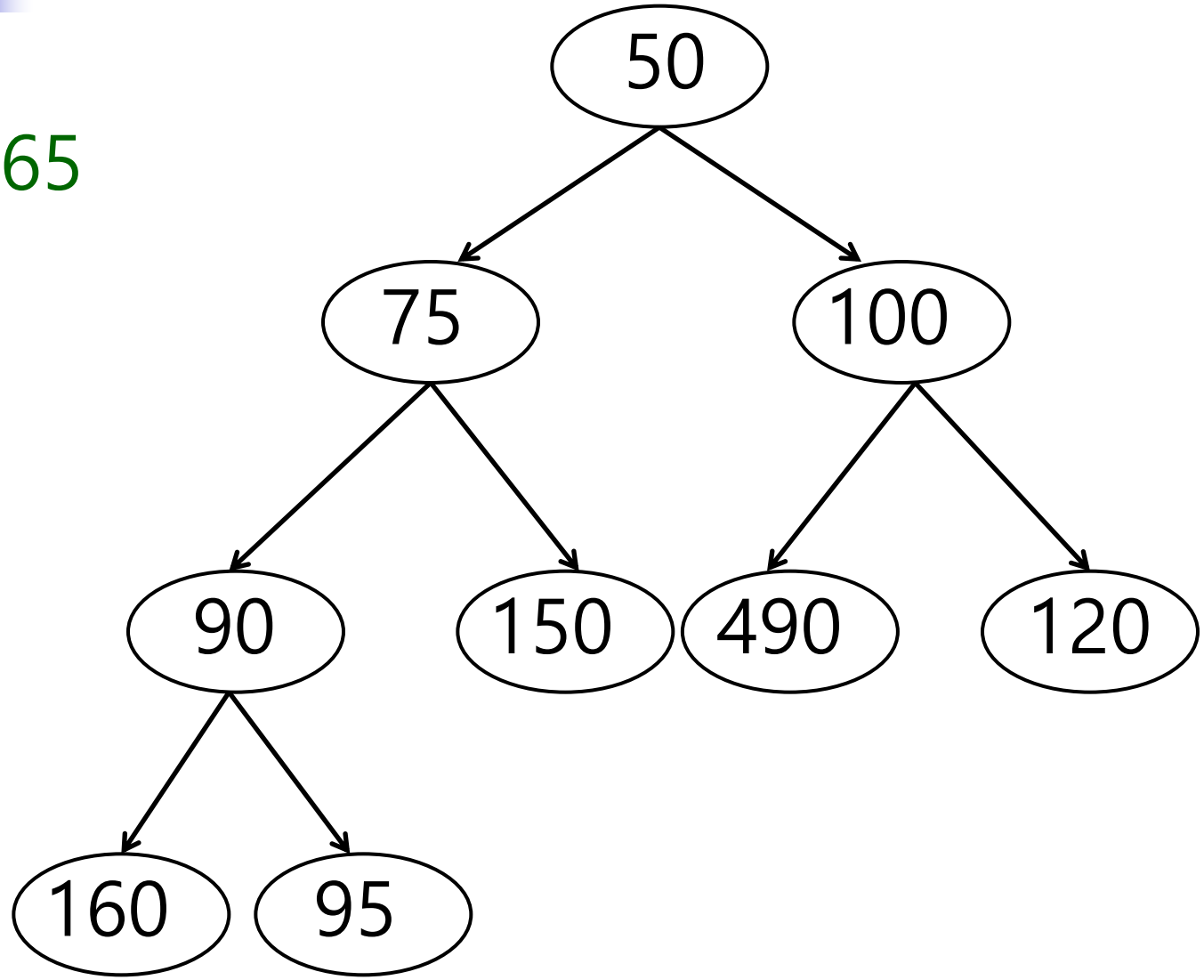


Inserting into a Min Heap: Example 1 (cont'd)



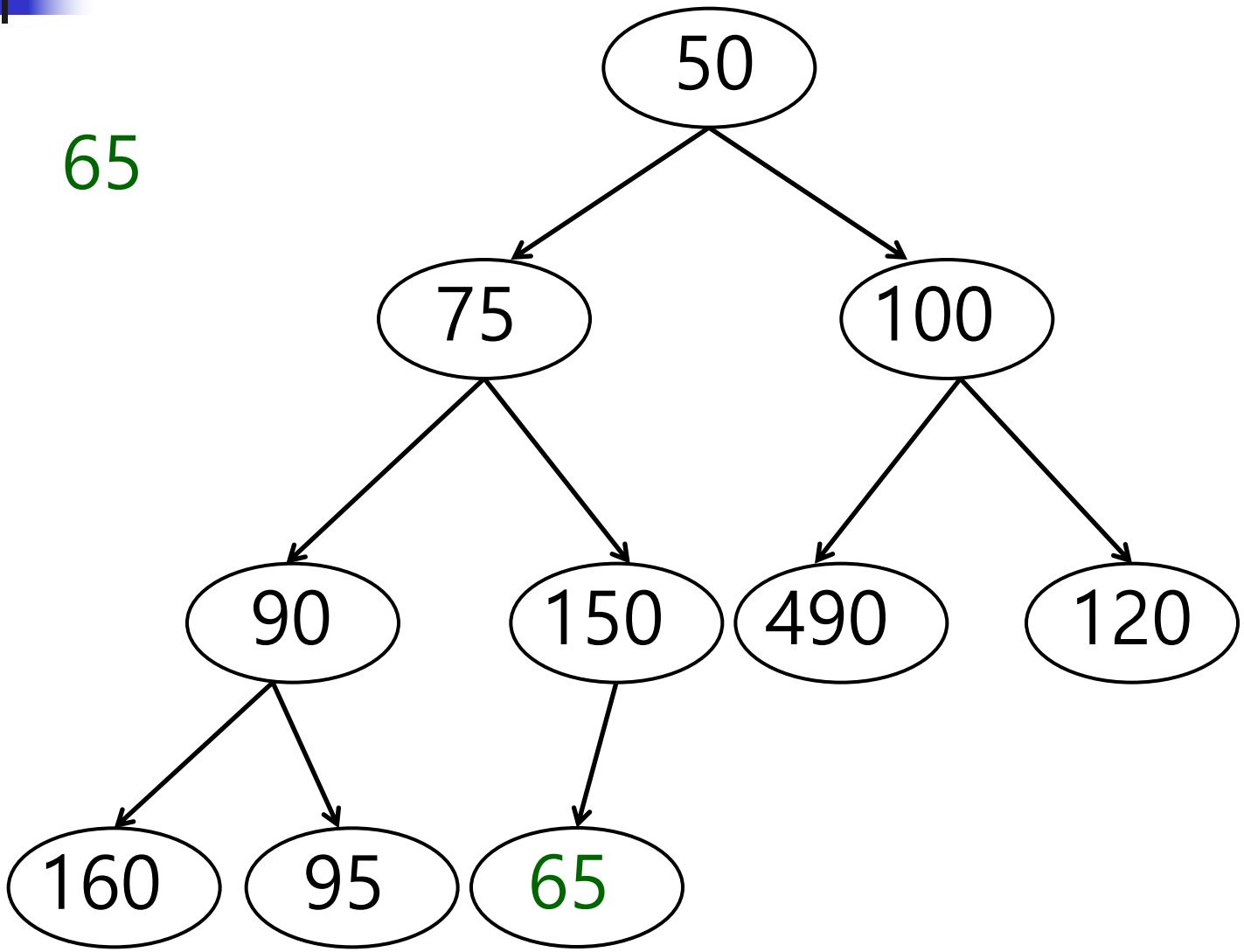
Inserting into a Min Heap: Example 2

65

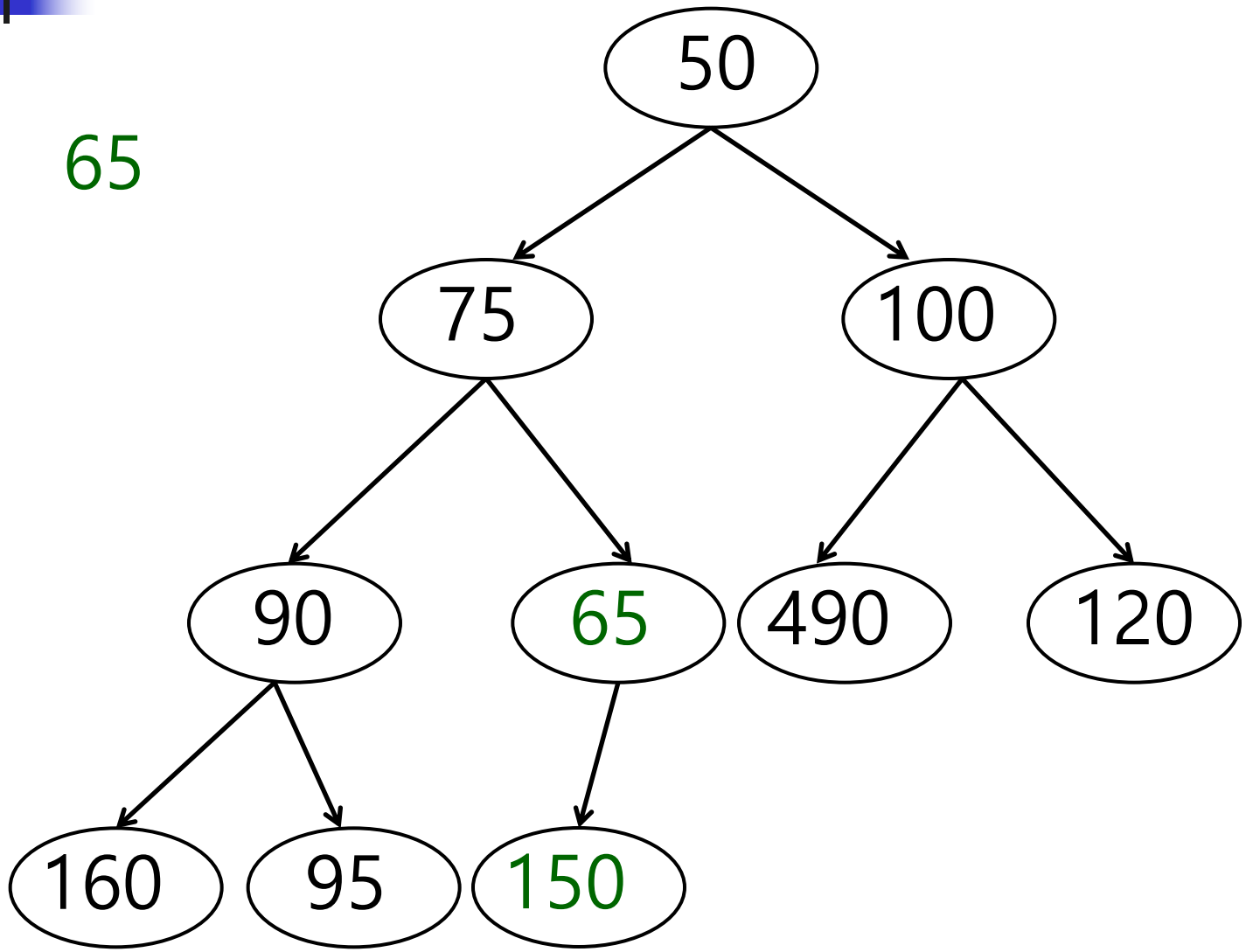


Inserting into a Min Heap: Example 2 (cont'd)

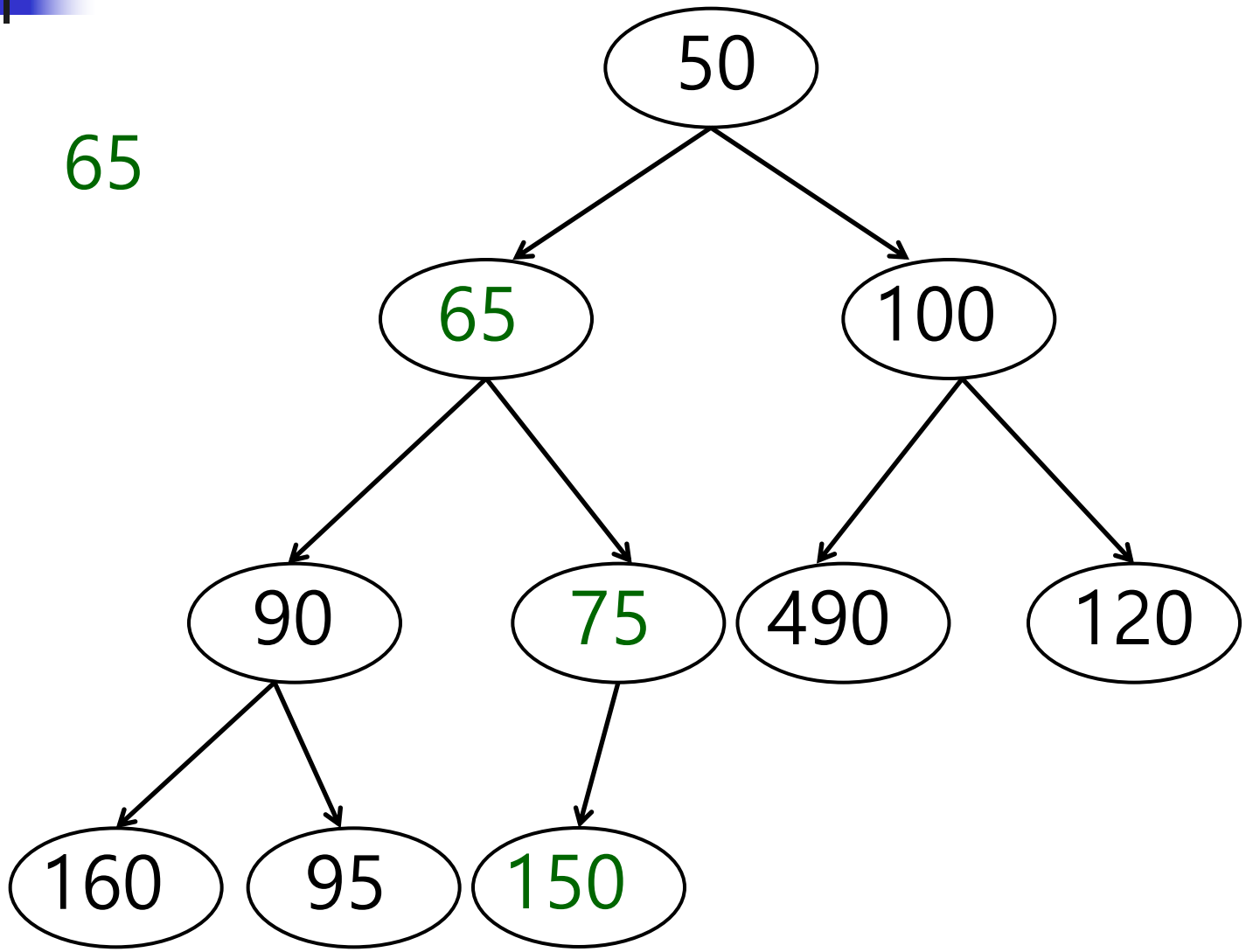
65



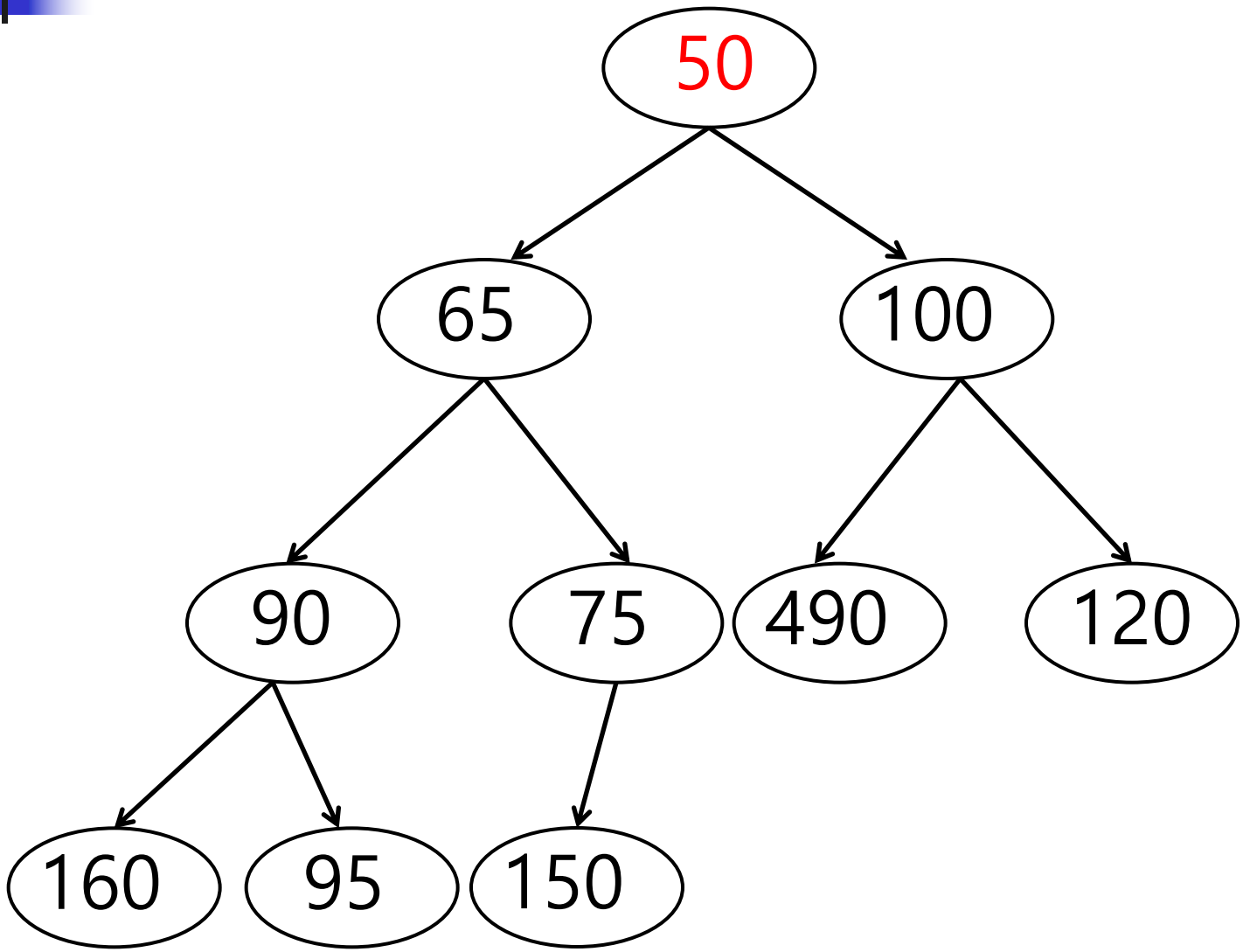
Inserting into a Min Heap: Example 2 (cont'd)



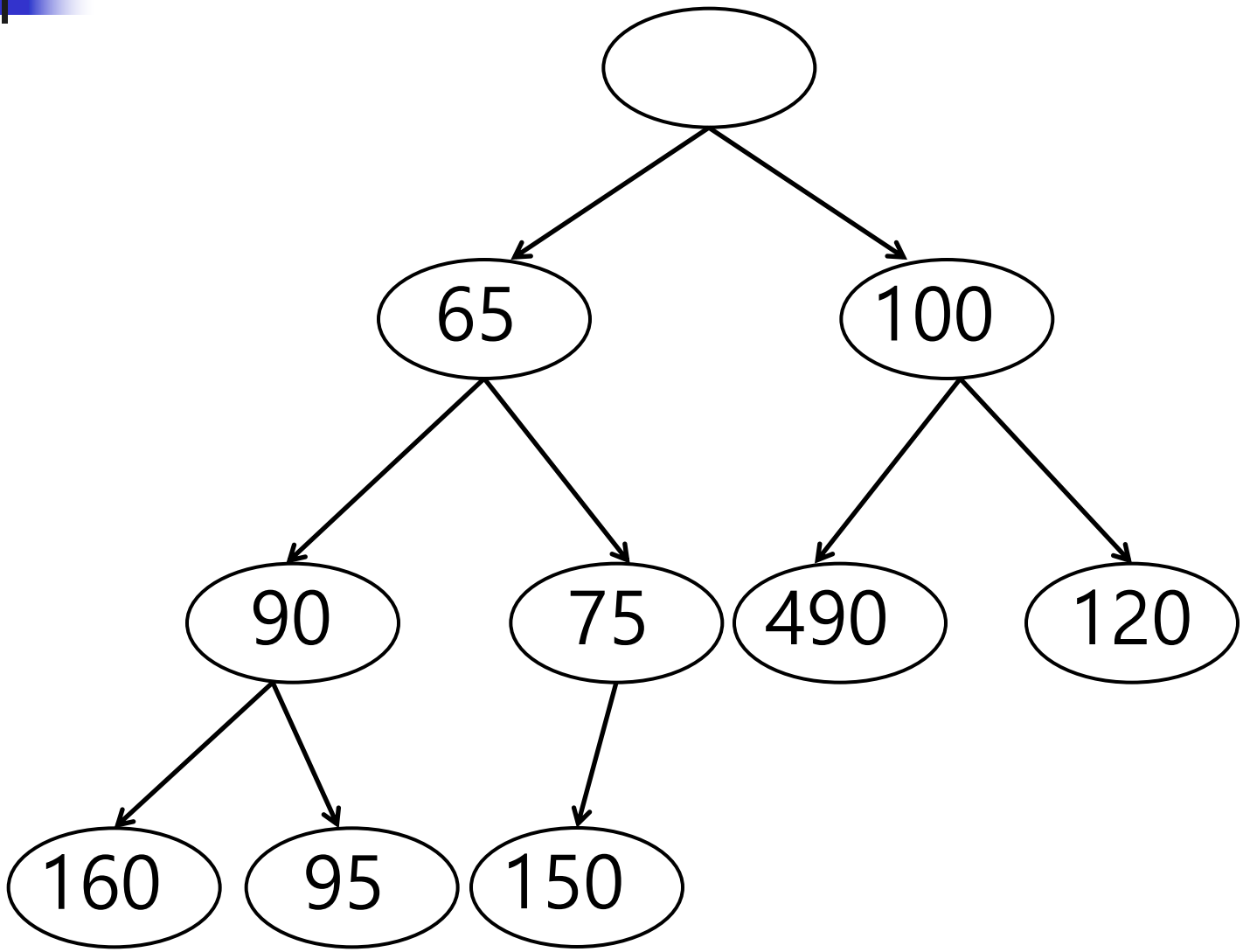
Inserting into a Min Heap: Example 2 (cont'd)



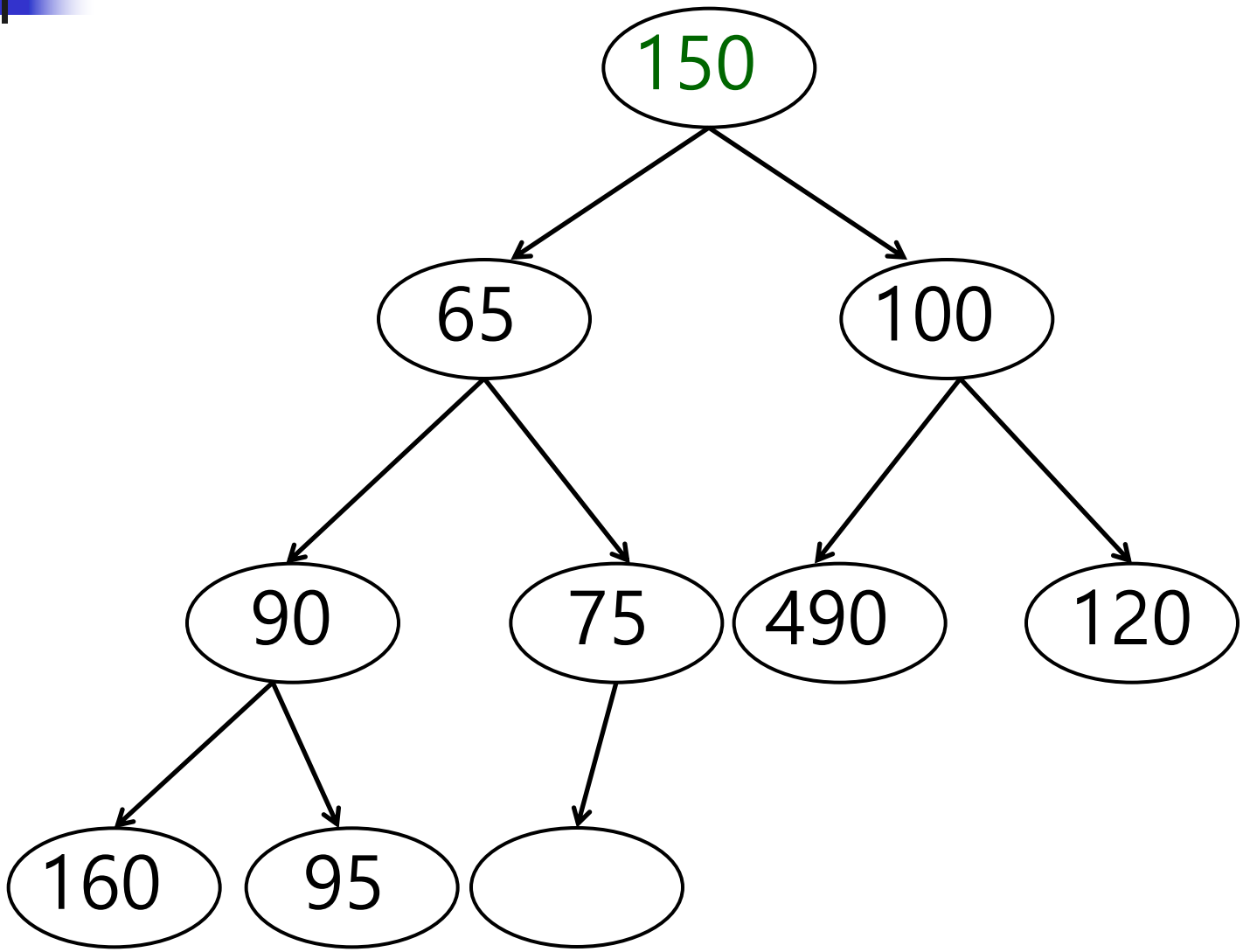
Deleting from a Min Heap: Example



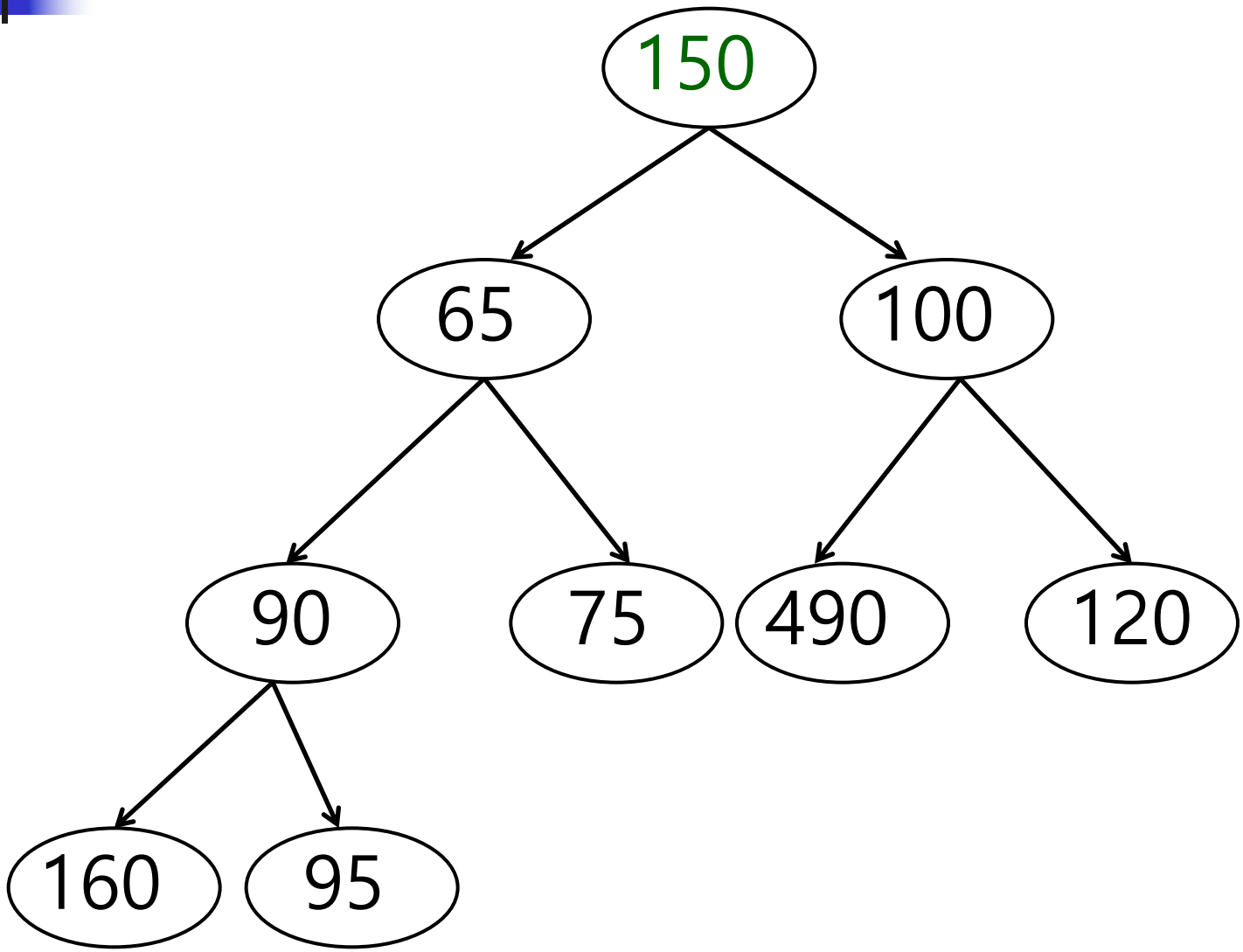
Deleting from a Min Heap: Example (cont'd)



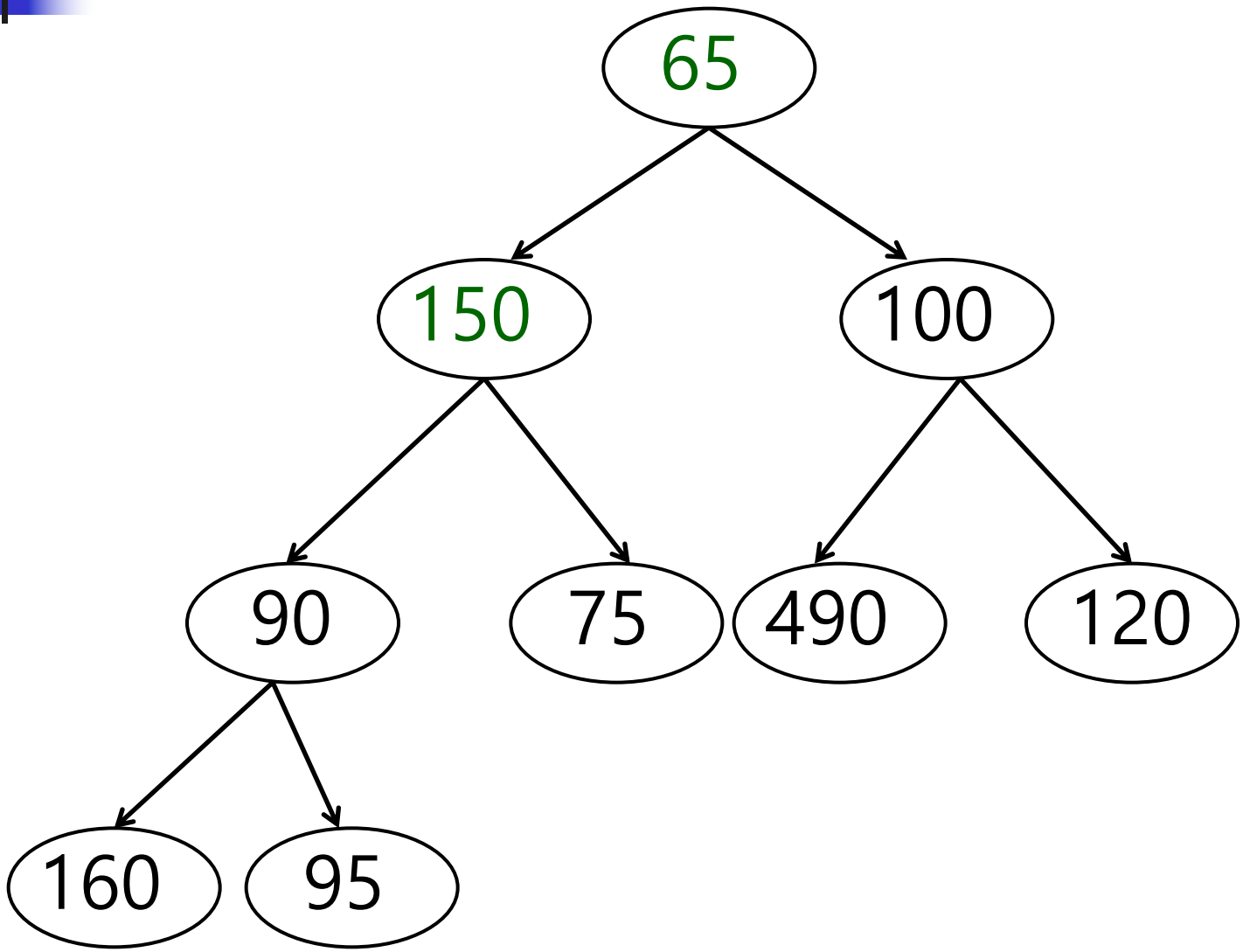
Deleting from a Min Heap: Example (cont'd)



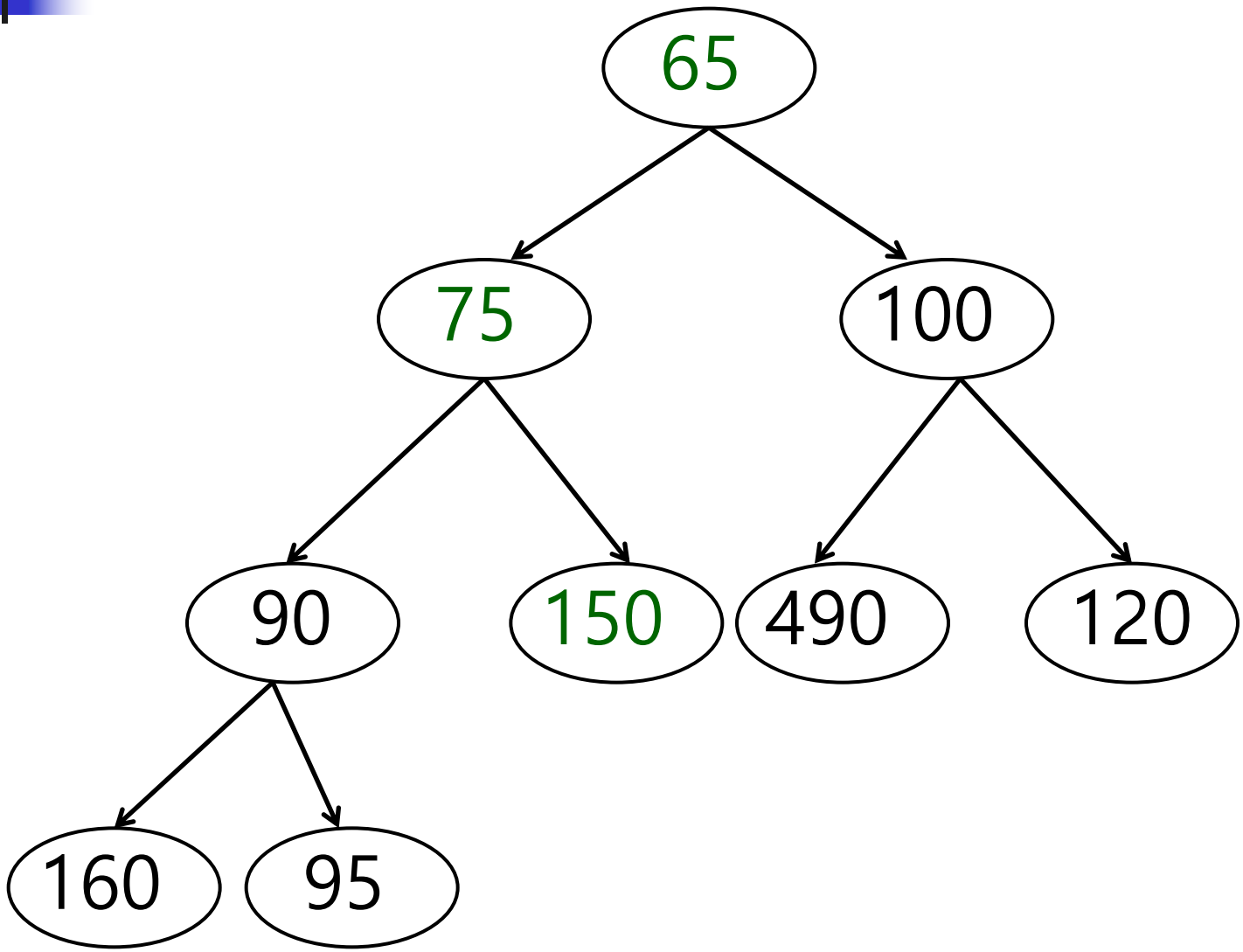
Deleting from a Min Heap: Example (cont'd)



Deleting from a Min Heap: Example (cont'd)

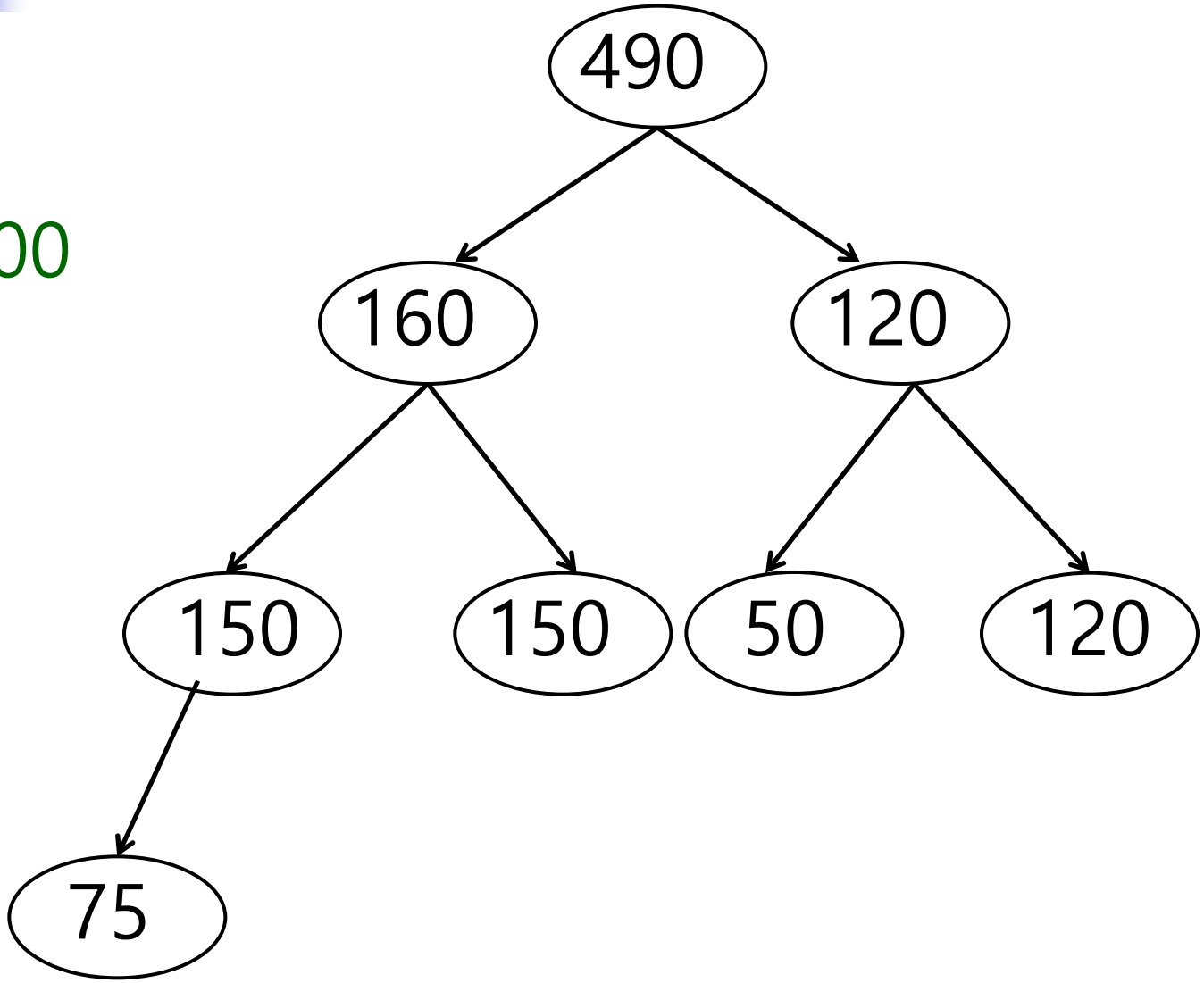


Deleting from a Min Heap: Example (cont'd)

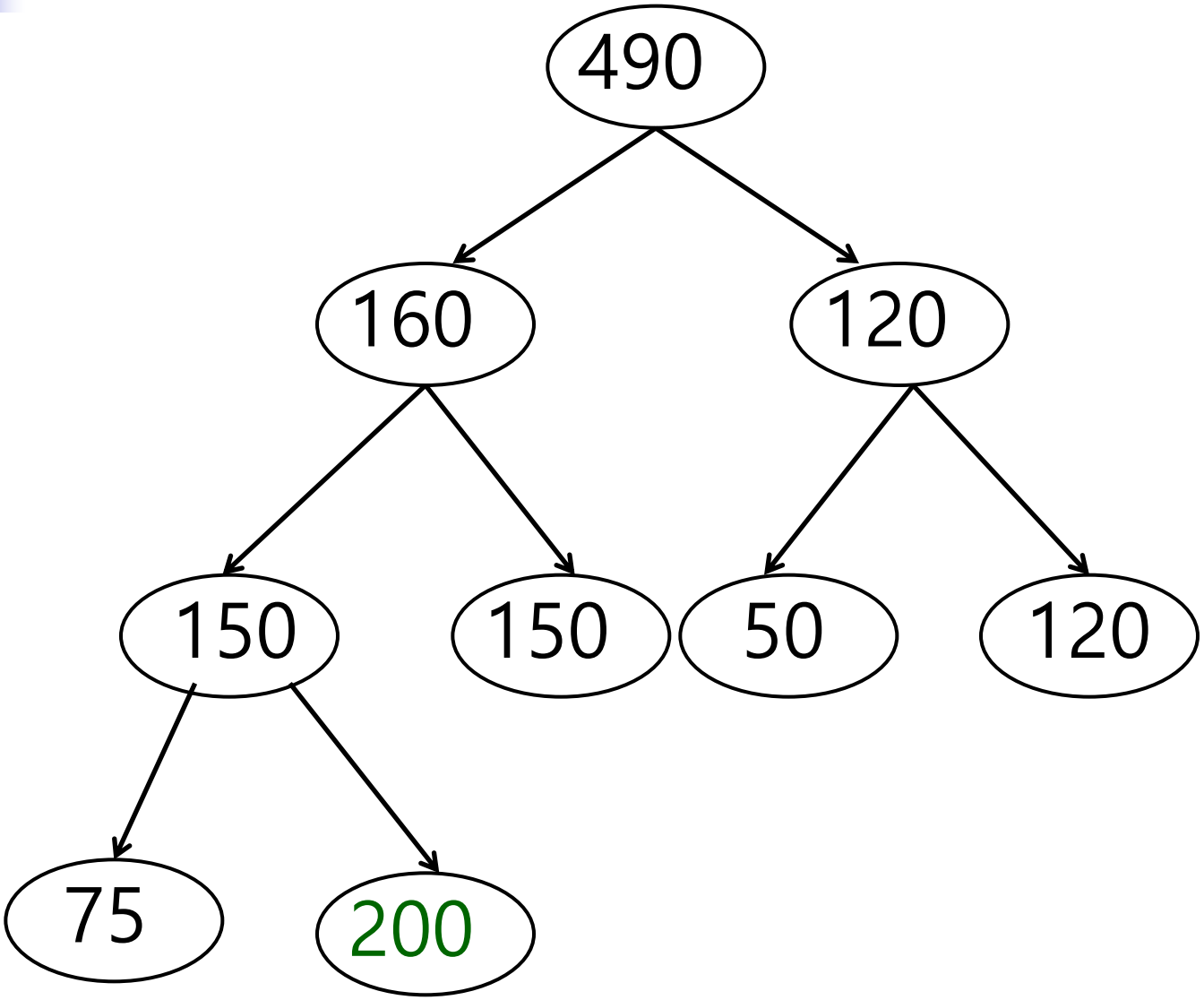


Inserting into a Max Heap: Example

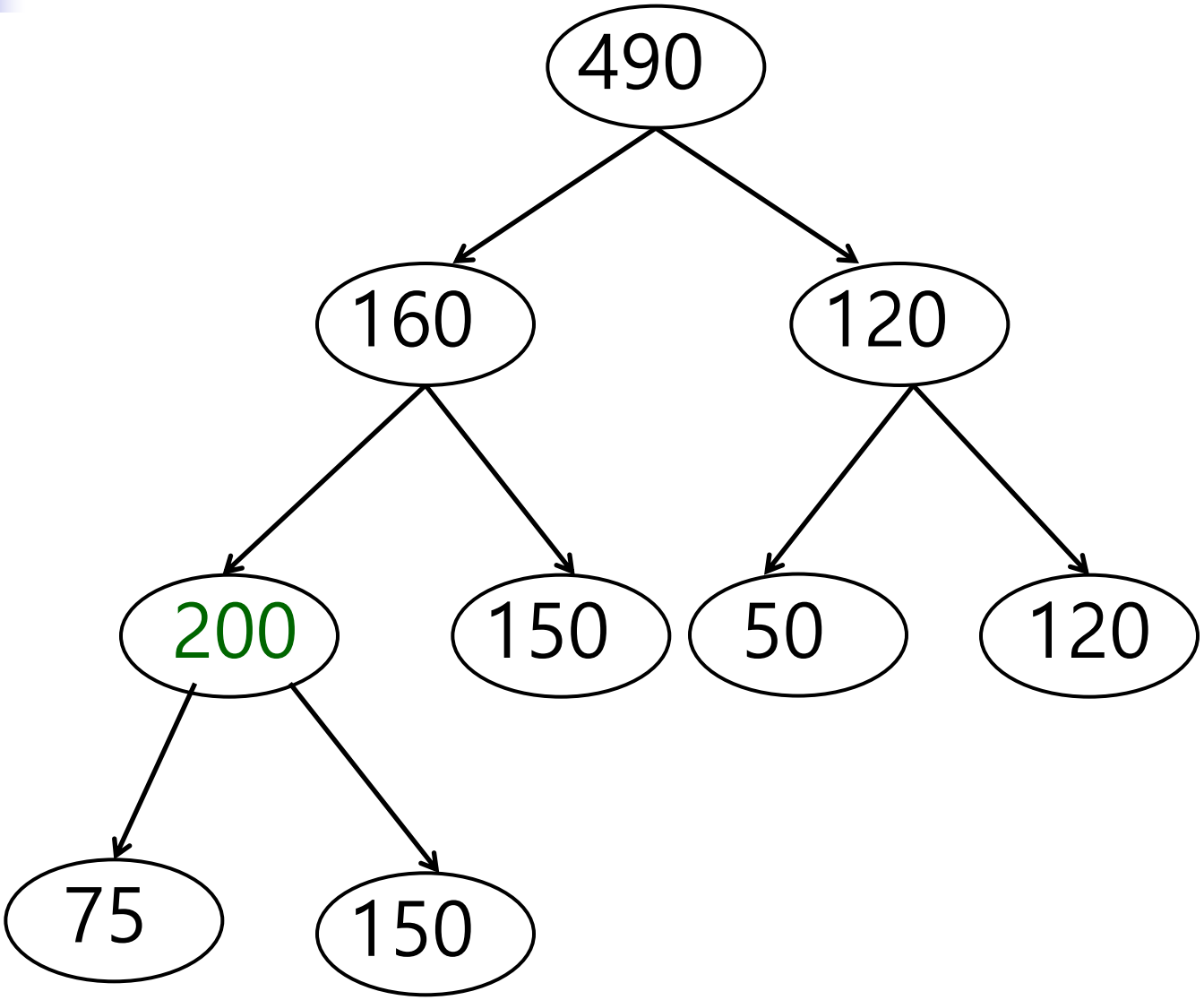
200



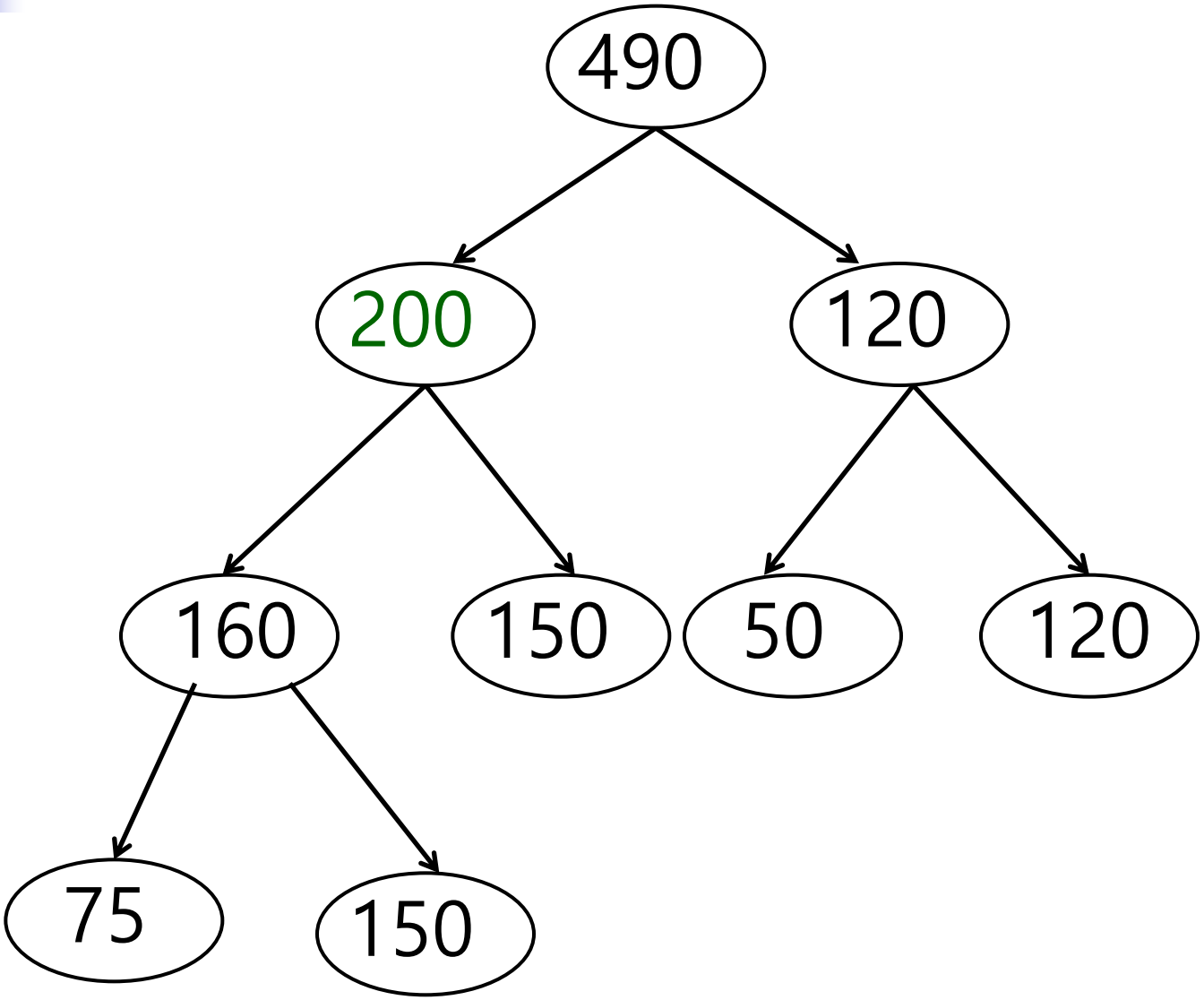
Inserting into a Max Heap: Example



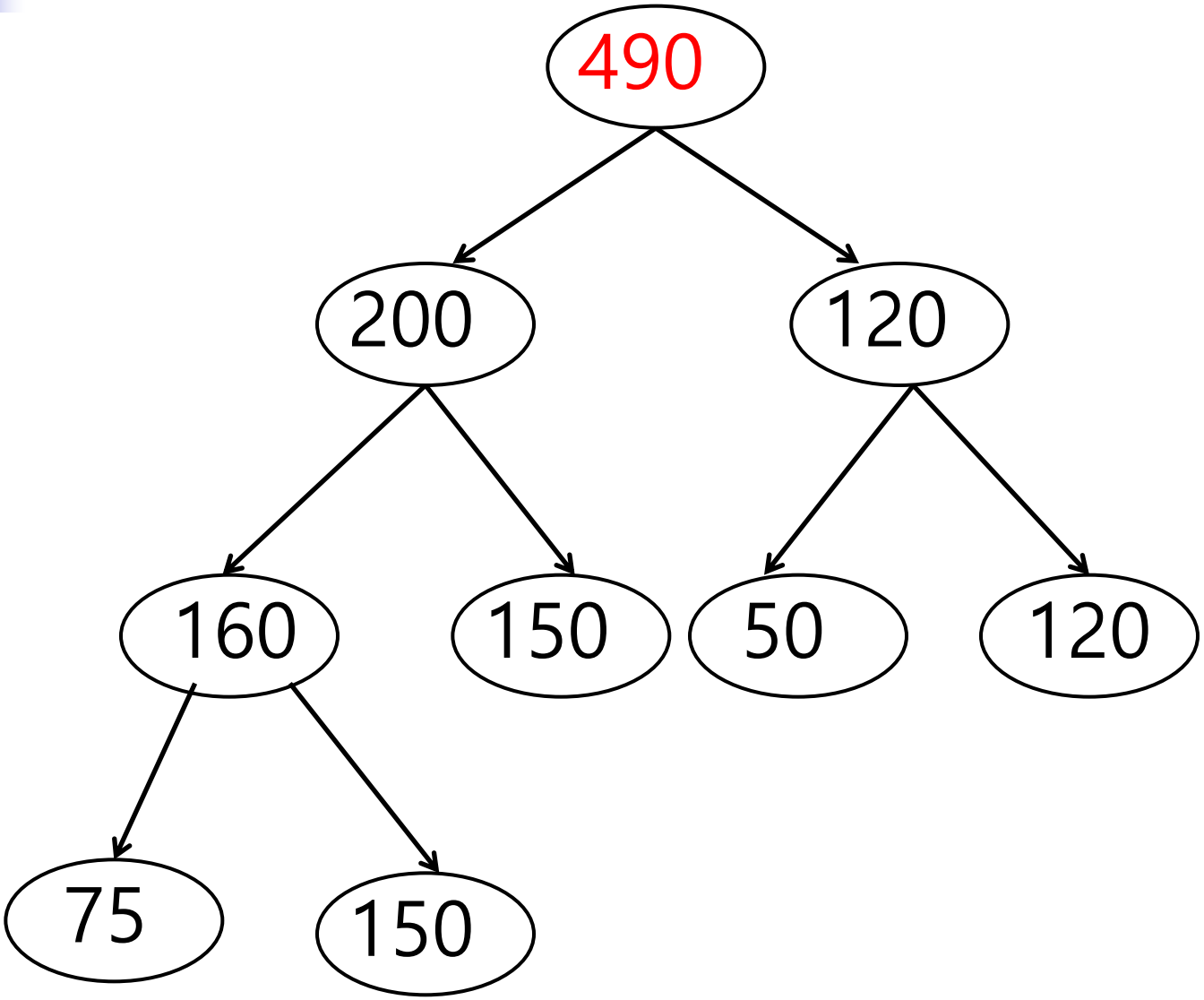
Inserting into a Max Heap: Example (contd.)



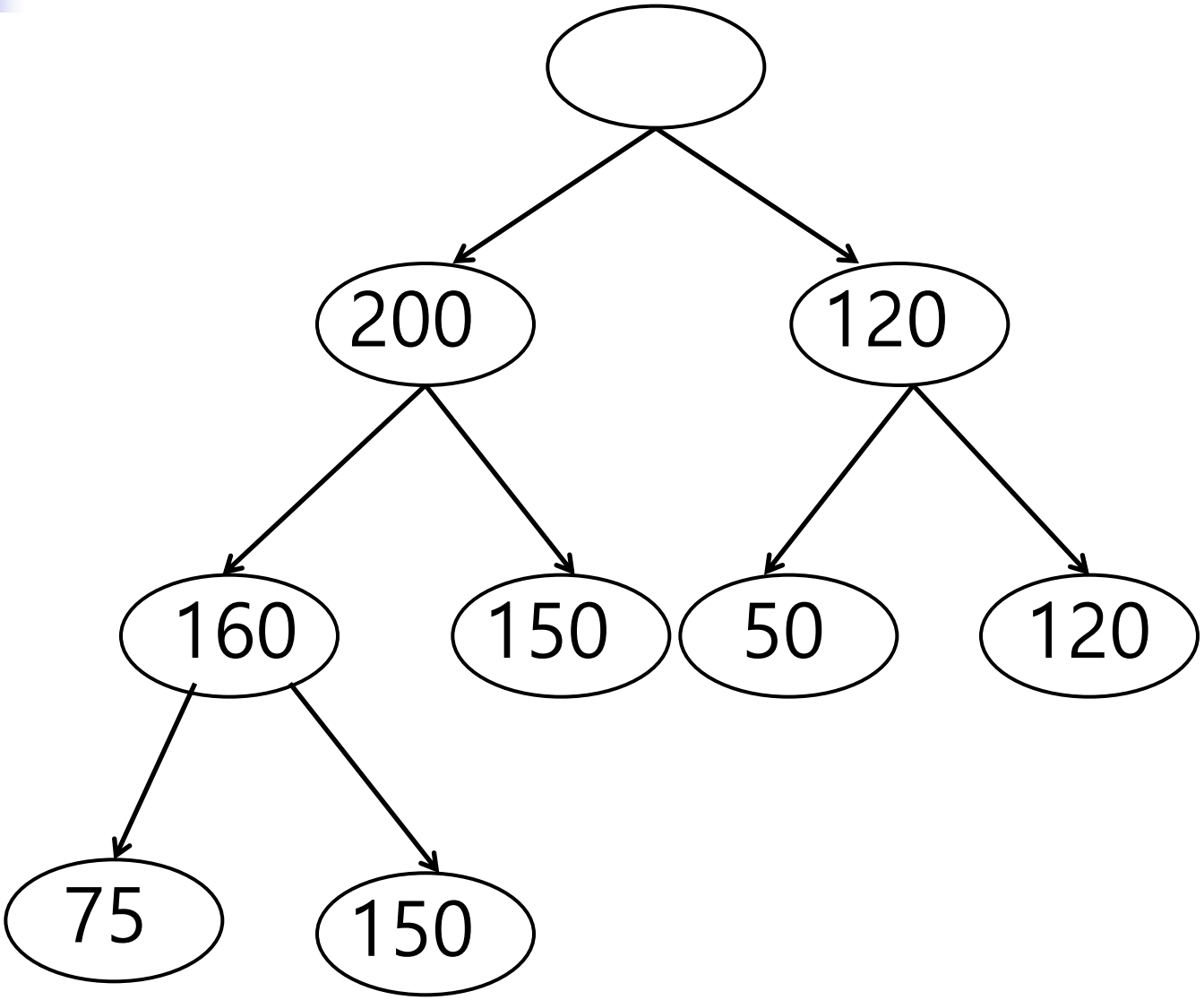
Inserting into a Max Heap: Example (contd.)



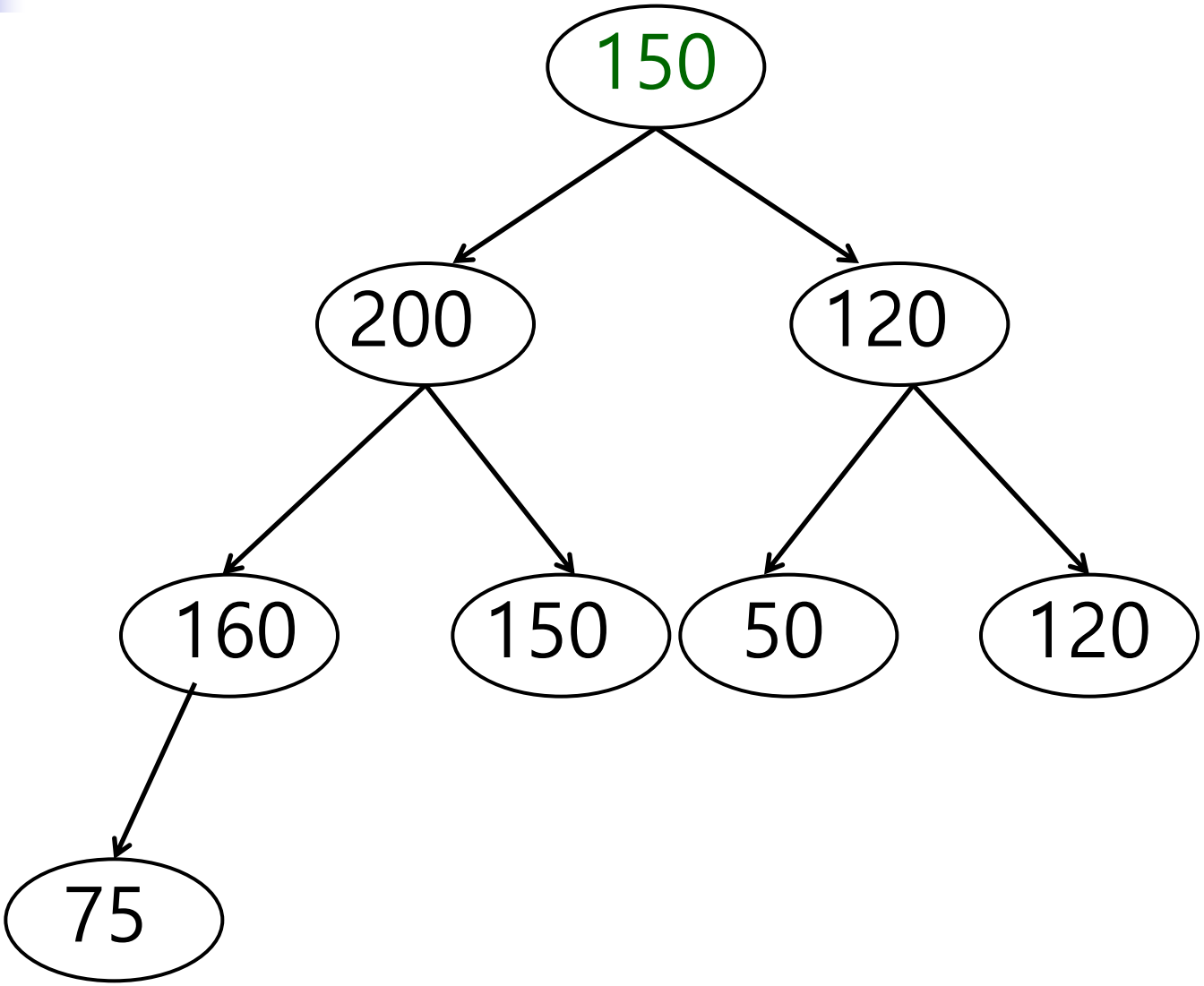
Deleting from a Max Heap: Example



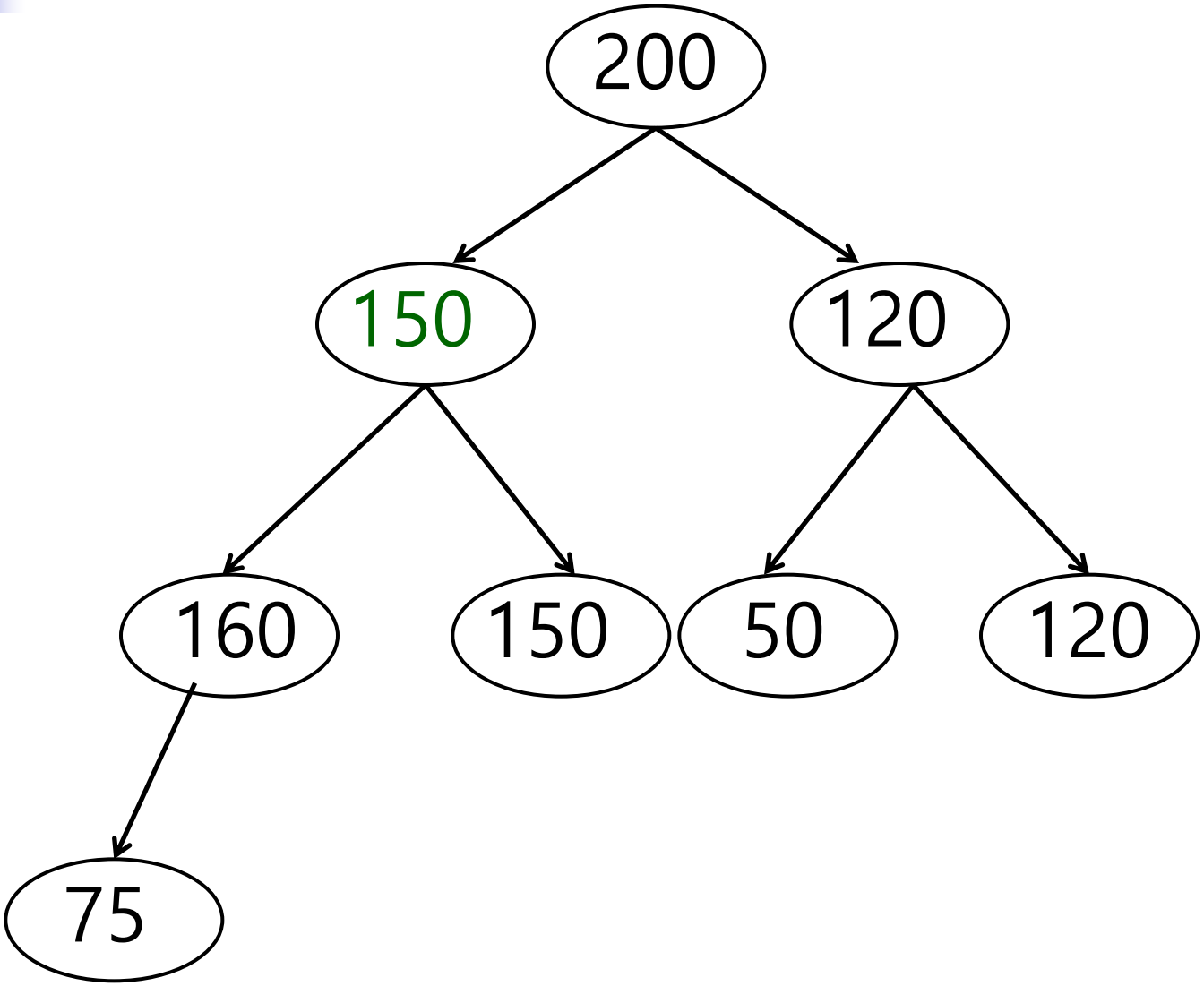
Deleting from a Max Heap: Example



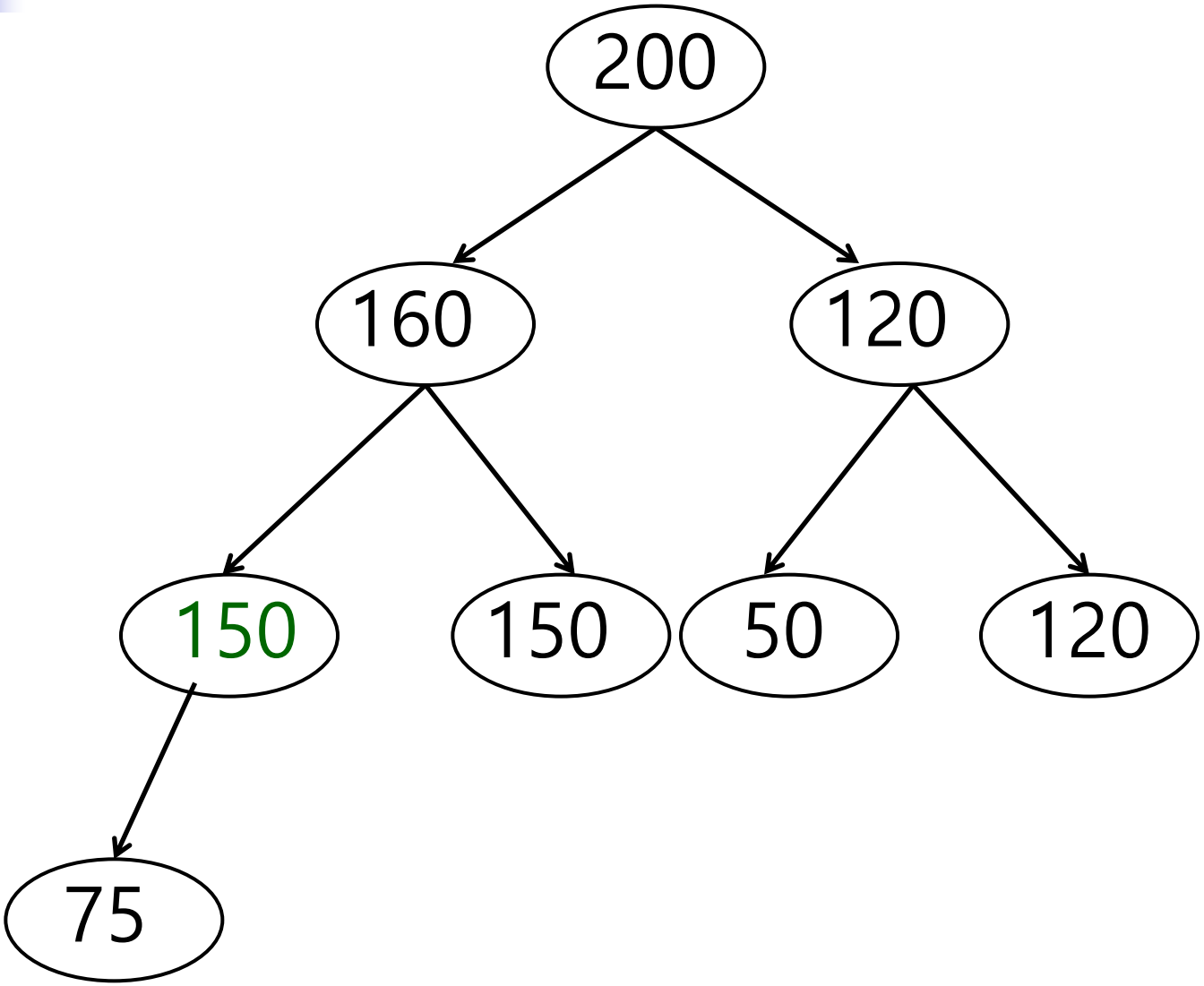
Deleting from a Max Heap: Example



Deleting from a Max Heap: Example



Deleting from a Max Heap: Example



Heap Sort (Ascending Order)

- Insert All Keys into a Min Heap.
- Keep deleting the root key and inserting the deleted key into a separate list.
  - Each deletion triggers the min heap to be restructured to maintain the min heap properties.

Heap Sort (Descending Order)

- Insert All Keys into a Max Heap.
- Keep deleting the root key and inserting the deleted key into a separate list.
  - Each deletion triggers the max heap to be restructured to maintain the max heap properties.

Heap Sort: Performance and Properties

- $O(n \log_2 n)$
  - n: number of keys in the list
  - $\log_2 n$ comparisons for each restructuring
- Small need for storage space
- Useful for embedded systems with real-time performance requirement

Exercise 1

- (1) Create a min heap with the following keys
  - 15 86 32 55 27 32 73 15 94 44
- (2) Do a heap sort (in ascending order)

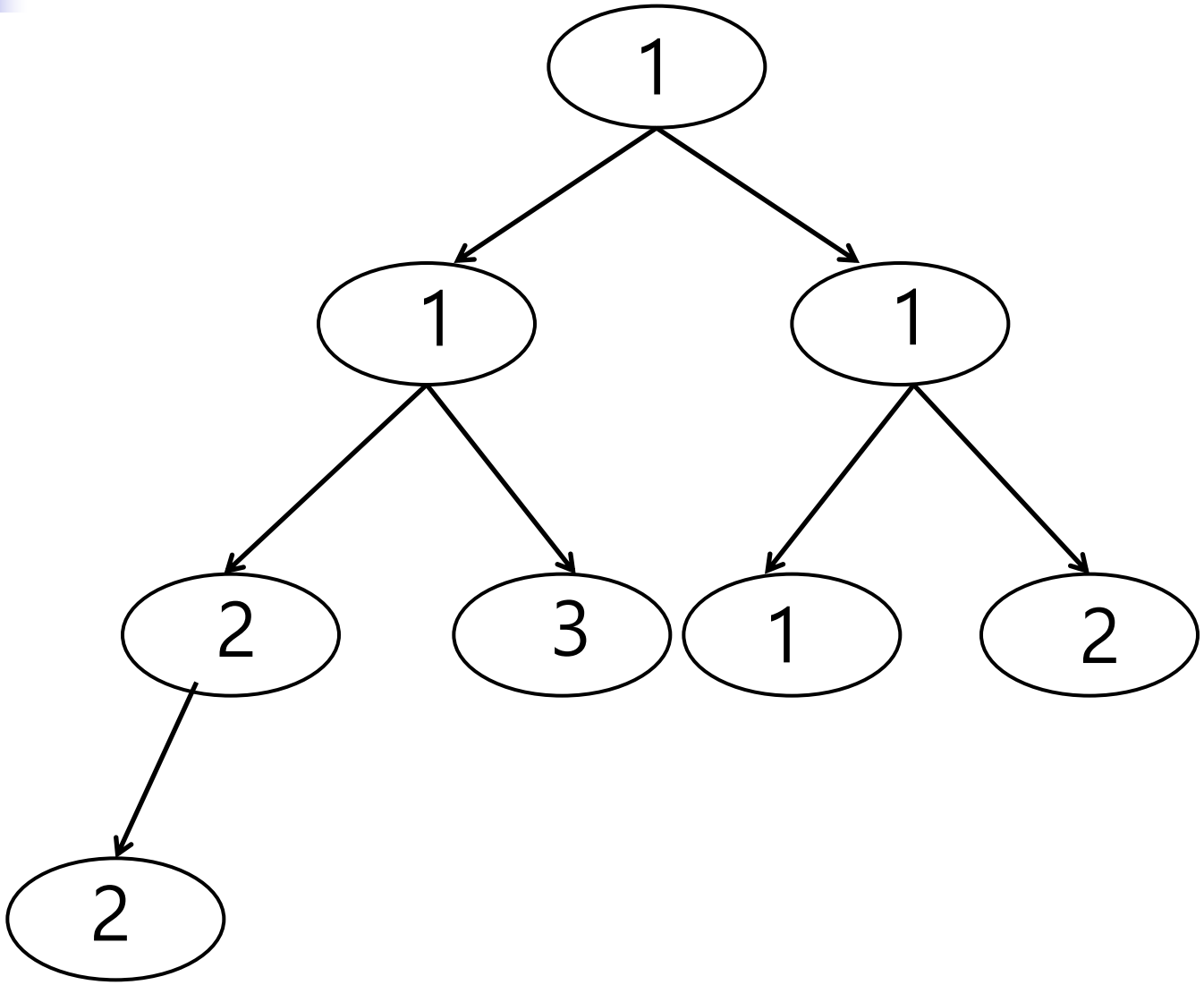
Exercise 2

- (1) Create a max heap with the following keys
  - 15 86 32 55 27 32 73 15 94 44
- (2) Do a heap sort (in descending order)

Summary Definition of a Heap

- A heap is a complete binary tree where data are automatically maintained in a priority order.

Min Heap as a Priority Queue
(Each key is one of the permissible priorities)



Radix Sort

Radix Sort

- “Radix” = “Character”
- Sort by numeric or alphabetic symbols in lexicographical order
- Least Significant Digit (LSD) Radix Sort
  - stable sort
- Most Significant Digit (MSD) Radix Sort
  - not stable sort
- Supplementary Reading
  - http://en.wikipedia.org/wiki/Radix_sort

LSD Radix Sort

- k Passes over n Keys
- 1st Pass
  - Create a list for each distinct LSD digit in lexicographic order, and throw the keys of the same LSD digit into the list in their original order.
  - Make a sequence of the lists created above.
- ith Pass (i = 2 through k)
  - Repeat the above for the ith LSD digit

LSD Radix Sort: Example

sort the list:
170, 90, 2, 802, 24, 45, 75, 66

1st pass:
(170, 90), (2, 802), (24), (45, 75), (66)

2nd pass:
(2), (802), (24), (45), (66), (170, 75), (90)

3rd pass:
(2), (24), (45), (66), (75), (90), (170), (802)

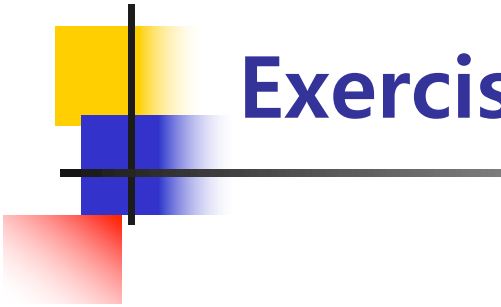
LSD Radix Sort: Example

sort the list:
170, 90, 2, 802, 24, 45, 75, 66
(170, 090, 002, 802, 024, 045, 075, 066)

1st pass:
(170, 090), (002, 802), (024), (045, 075), (066)

2nd pass:
(002), (802), (024), (045), (066), (170, 075), (090)

3rd pass:
(002), (024), (045), (066), (075), (090), (170), (802)



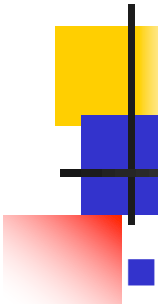
Exercise

sort the list:
170000, 10

Exercise

sort the list:

17000, 190, 2, 3802, 24, 45, 75, 66



MSD Radix Sort

- Recursively sort for each digit, starting from the most significant digit.
- Not a stable sort

MSD Radix Sort: Example

sort the list:
170, 90, 2, 802, 24, 25, 75, 66
(170, 090, 002, 802, 024, 025, 075, 066)

1st pass:
(090, 002, 024, 025, 075, 066), (170), (802)

2nd pass:
(002), (024,025), (066), (075), (090), (170), (802)

3rd pass:
(002), (024), (025), (066), (075), (090), (170), (802)

MSD Radix Sort: Exercise

sort the list:
170, 4, 160, 90, 4, 45, 24, 890, 892, 802, 2, 45

Radix Sort: Performance and Qualities

- O(kn)
  - k: number of digits per key
  - n: number of keys to sort
  - Performance depends on the size of the keys and choice of the radix (digit, alphabet,…)
- Problems
  - Space for the radix-sorted lists

Bubble Sort

Bubble Sort

- (Not Recommended)
  - (Included just for comparison with Selection Sort)
- Similar to Selection Sort
- On the i^{th} Pass
  - Compare the j^{th} key with the $j+1^{th}$ key, and exchange them if the j^{th} key > $j+1^{th}$ key ($1 <= j <= n-i+1$)
  - Percolate ("bubble") up the largest key among the first n-i keys to the $n-i+1^{th}$ position.
- http://en.wikipedia.org/wiki/Bubble_sort

Bubble Sort: Example

sort the list:
13, 4, 9, 21, 37, 17, 22, 3, 8

pass 1: 4, 13, 9, 21, 37, 17, 22, 3, 8

4, 9, 13, 21, 37, 17, 22, 3, 8

4, 9, 13, 21, 17, 37, 22, 3, 8

4, 9, 13, 21, 17, 22, 37, 3, 8

4, 9, 13, 21, 17, 22, 3, 37, 8

4, 9, 13, 21, 17, 22, 3, 8, 37

Air Bubble in Water

Bubble Sort: Example (cont'd)

sort the list:

13, 4, 9, 21, 37, 17, 22, 3, 8

pass 2: 4, 9, 13, 21, 17, 22, 3, 8, 37

4, 9, 13, 17, 21, 3, 8, 22, 37

pass 3: 4, 9, 13, 17, 21, 3, 8, 22, 37

4, 9, 13, 17, 3, 8, 21, 22, 37

pass 7: 4, 3, 8, 9, 13, 17, 21, 22, 37

pass 8: 3, 4, 8, 9, 13, 17, 21, 22, 37

Some Sorting-Related Concepts

- Stable sort
- Indirect sort (indexing)

Stable Sort

- Sorting on a primary key maintains sort order based on a secondary key.

name	age	employer
Lee	30	IBM
Kim	25	LG
Lee	23	LG
Kim	40	Samsung
Chung	28	Samsung

in employer order

Stable Sort (cont'd)

in name order

name	age	employer
Chung	28	Samsung
Kim	25	LG
Kim	40	Samsung
Lee	30	IBM
Lee	23	LG

stable

name	age	employer
Chung	28	Samsung
Kim	40	Samsung
Kim	25	LG
Lee	23	LG
Lee	30	IBM

unstable

Indirect Sort

- Direct sort: move records
- Indirect sort: use an index

	name	age	employer
1	Lee	30	IBM
2	Kim	25	LG
3	Lee	23	LG
4	Kim	40	Samsung
5	Chung	28	Samsung

1	5
2	2
3	4
4	1
5	3

index

Summary Observations

- Every sorting algorithm has its uses and problems.
  - There is no such thing as “the best sorting algorithm for every situation”.
- For a large internal list, a combination of a few algorithms may be best.
  - (e.g.) insertion sort, quick sort, merge sort
    - Quick sort can use insertion sort for small sublists.
    - Merge sort can use quick sort for medium-size sublists.

Sorting in Widely Used Systems

- Efficient quicksort implementations generally outperform merge sort for sorting RAM-based arrays.
- Merge sort is a stable sort and is more efficient at handling slow-to-access sequential media. Merge sort is often the best choice for sorting a linked list.
- The slow random-access performance of a linked list makes some other algorithms (such as quicksort) perform poorly, and others (such as heapsort) impossible.
- The Linux kernel uses merge sort for linked lists. Python uses Timsort, a tuned hybrid of merge sort and insertion sort, that has become the standard sort algorithm in Java SE 7 on Android platform and in GNU Octave (a free alternative to Matlab).
- As of Perl 5.8, merge sort is its default sorting algorithm (it was quicksort in previous versions of Perl).
- In Java, the Arrays.sort() methods use merge sort or a tuned quicksort depending on the data types; switch to insertion sort when fewer than seven array elements are being sorted.

WHW6-1: (5 points) Descending Order Heap Sort (Max Heap)

31, 11, 3, 20, 19, 24, 17, 3, 31

WHW6-2: (5 points) Ascending Order Heap Sort (Min Heap)

31, 11, 3, 20, 19, 24, 17, 3, 31

WHW6-3: (5 points) LSD Radix Sort

310, 11, 3, 20, 11119, 24, 17, 3, 31

WHW6-4: (5 points) MSD Radix Sort

310, 11, 3, 20, 11119, 24, 17, 3, 31

End of Lecture