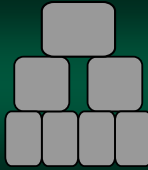




Heaps & Priority Queues

Part 9

1



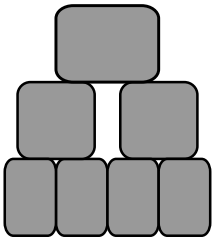
Heaps

Piles of data!

2


What is a heap?

- A *heap* is a binary tree, but a notable format to the nodes
- The value of a node is smaller (or larger) than both of its children
- Every subtree is a heap



3

Terminology Warning



- The *heap data structure* is not the same as the operating system's heap
- They are often confused...
- The heap data structure is a tree that stores "heavier" objects at the bottom

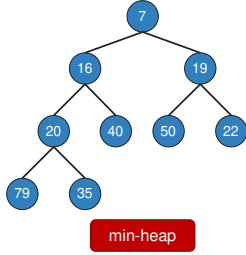
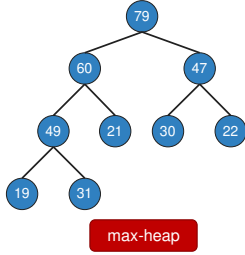
4

Min and max-heaps

- Min-heap**
 - each of a node's descendants have a "heavier" value
 - stores smaller items (minimal items) at the top of the tree
- Max-heap**
 - each node's parent has a "heavier" value
 - stores larger items (maximum items) at the top of the tree

5

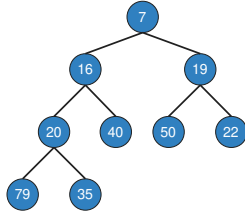
Min and Max-heaps

6

Heaps

- Heaps are *complete* binary trees
- Nodes are added in *breadth-first* order
- The resulting tree is always optimal and *balanced*



Fall 2024

Sacramento State - Gosh - CS130

7

7

Height of a Heap

- Let i be the depth of a node
- Then, there are 2^i nodes of depth i
- Heap *always* has a height of $\log_2(n)$



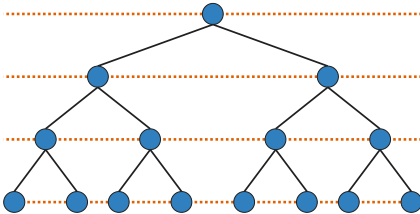
Fall 2024

Sacramento State - Gosh - CS130

8

8

Height of a Heap



Fall 2024

Sacramento State - Gosh - CS130

9

9

Adding a Node

1. Begin at next available position for a leaf
2. Now the item needs to be *up-heaped*
 - move the entry up depending on its value until a correct position is found
 - as this is done, nodes are swapped - parent to child change position
 - since a heap *always* has height $\log_2(n)$, upheap runs in $O(\log n)$ time

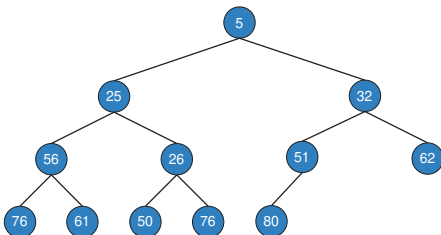
Fall 2024

Sacramento State - Gosh - CS130

10

10

Min-heap: Adding a Node 13



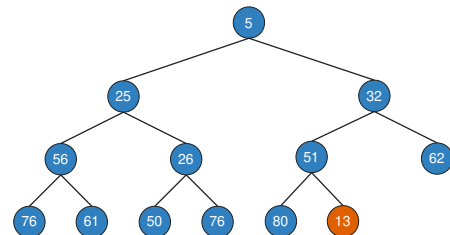
Fall 2024

Sacramento State - Gosh - CS130

11

11

Min-heap: Adding a Node 13

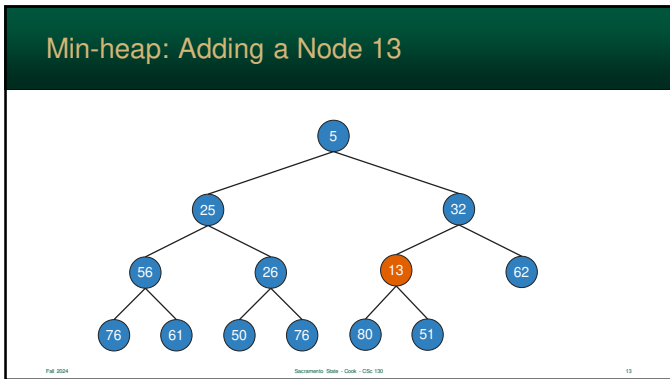


Fall 2024

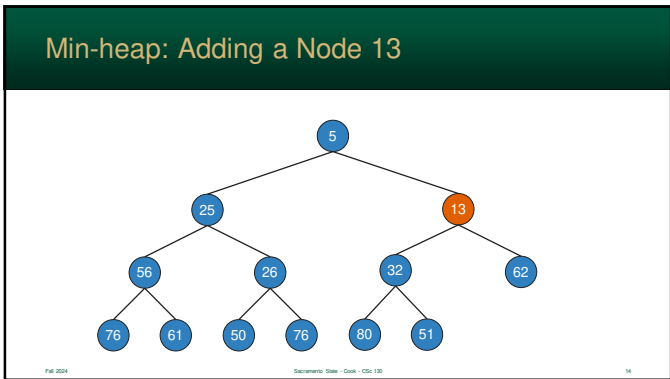
Sacramento State - Gosh - CS130

12

12




13



14

Total Up-heap-val



- Just to make matters confusing, up-heap is also known by various other terms – *which are all valid*
- These are some:
 - bubble-up
 - percolate-up
 - sift-up
 - heapify-up
 - cascade-up

Fall 2024 Sacramento State - Oak - CS130 15

15

Deleting a Node

- Deleting a node is quite different from adding
- Heaps must maintain *completeness*
 - so, the right-most leaf is needed to replace the deleted node
 - why? We replace the deleted node with the last one added.

Fall 2024 Sacramento State - Oak - CS130 16

16

Deleting a Node

- The steps to delete:
 1. remove the node
 2. replace it with the right-most leaf
 3. now, it needs to down-heaped (moved down) to the correct location
- This runs in **$O(\log n)$** time

Fall 2024 Sacramento State - Oak - CS130 17

17

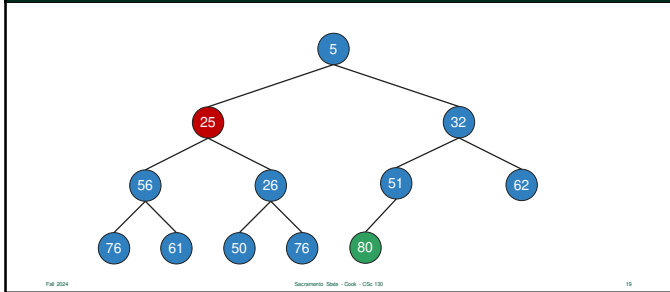
Downheap Algorithm

- With a heap, every node has two children
 - as you downheap, you swap nodes
 - so, which one do you select?
- Preserve the heap structure ← **vital**
 - on a min-heap, swap with the smallest child
 - on a max-heap, swap with the largest child

Fall 2024 Sacramento State - Oak - CS130 18

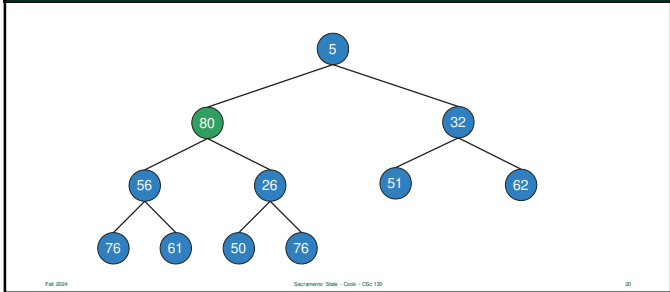
18

Min-heap: Deleting 25



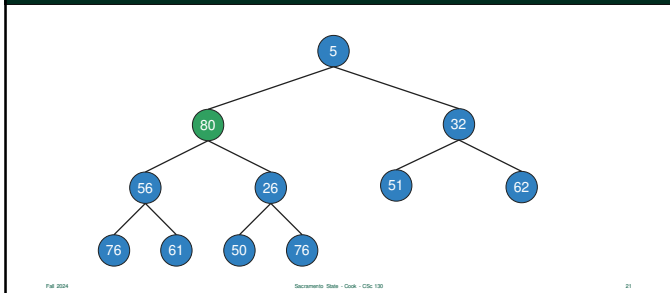
19

Replace. Now must down-heap.



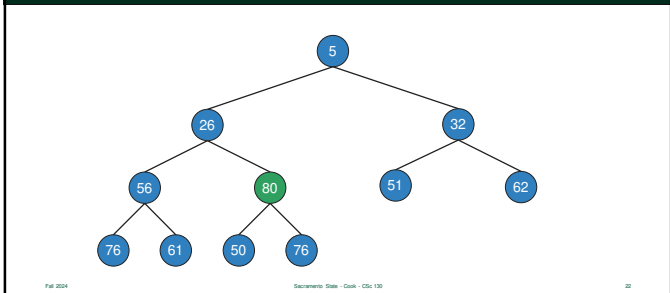
20

Replace. Now must down-heap.



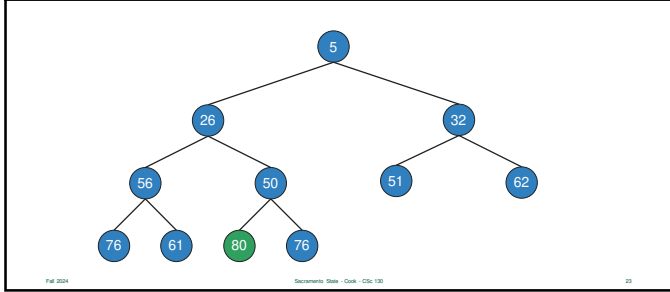
21

Replace. Now must down-heap.




22

Replace. Now must down-heap.



23

As You Expected...



- Just like up-heap, down-heap has several other, completely valid, names
- These are some:
 - bubble-down
 - percolate-down
 - sift-down
 - heapify-down
 - cascade-down

24




Priority Queues

Queues can play favorites

25

Priority Queues


- A stack is first-in-last-out
- A queue is first-in-first-out
- A priority queue is modification of the queue ADT that follows the logic of *first-in-least-out*



26

First-in-Least-Out

- The *"least"* element is the first one that is removed
- If two items have the same "rank", items can be queued as normal
- The object's key can be used to determine if it is "least", but any field will do



27

What is the "Least" Item?

- Meaning of "least" is defined by the ADT
- It is abstract - does not mean "less than"
 - so, "least" can be any way of ranking items
 - ...if the items are mathematically transitive
 - "least" can be the largest value
- Examples of least:
 - the smallest / largest value
 - some ranking classification

28


Priority Queue Interface

public class PriorityQueue	
PriorityQueue ()	Create an empty PQ
void add (Item item)	Same as enqueue()
Item removeLeast ()	Same as dequeue()
Item getLeast ()	Same as peek(), first()
bool isEmpty ()	
int size ()	

29

Implementation

- Before we select a data structure to implement a priority queue, we should look how data will be used
- The goal is to get the best time efficiency with as little overhead as possible



30

Implementation

- The type data to be stored will influence how the priority queue is implemented
- We have quite a few options:
 - array
 - linked-list
 - tree / heap



Fall 2024

Sacramento State - CS&E - CS110

31

31

Implementation with an Array

- Unsorted array
 - enqueue requires $O(n)$ – resize array
 - dequeue requires $O(n)$ – search and moving
- Sorted array
 - enqueue requires $O(n)$ – find a position to insert and then move the rest
 - dequeue requires $O(n)$

Fall 2024

Sacramento State - CS&E - CS110

32

32

Implementation with a Linked List

- Unsorted linked list
 - enqueue takes $O(1)$
 - dequeue requires $O(n)$ – find & remove node
- Sorted linked list
 - enqueue requires $O(n)$ – must find a position and insert
 - dequeue requires $O(1)$ – just remove the head/tail

Fall 2024

Sacramento State - CS&E - CS110

33

33

We Need Another Data Structure!

- Arrays have a time complexity of $O(n)$ for Enqueue and Dequeue
- Linked Lists did have a single $O(1)$ operation, but the other was $O(n)$
- Given priority queues are updated often (just like normal queues), **arrays and linked lists are poor solutions**

Fall 2024

Sacramento State - CS&E - CS110

34

34

Hybrid Implementations

- In some cases, the key value can have a minor range of values – possibly just a few
- Examples:
 - hospital triage – immediate, delayed, minor
 - computer processes – OS, application, GUI
- We can make clever hybrid structures that maximize efficiency

Fall 2024

Sacramento State - CS&E - CS110

35

35

Hybrid Implementations

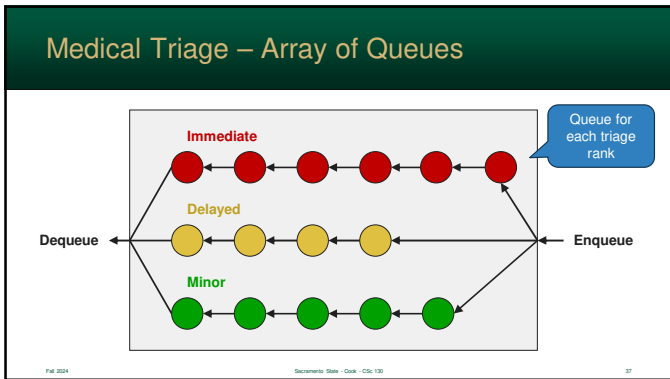
- If the key contains a small number of values, you can use multiple queues – one for each key value
- Basically, the priority queue, internally, will have an array of queues
- Adding/removing items will always be $O(1)$
 - $O(1)$ for the queue head
 - $O(1)$ for enqueue/dequeue (using linked list)

Fall 2024

Sacramento State - CS&E - CS110

36

36



37

- ### ... But Heaps are Universal
- However, in most cases, the key values have large ranges
 - For example, if the key is a 32-bit integer, do you want to create 4 million queues?
 - Didn't think so...
 - So, this only works in limited situations

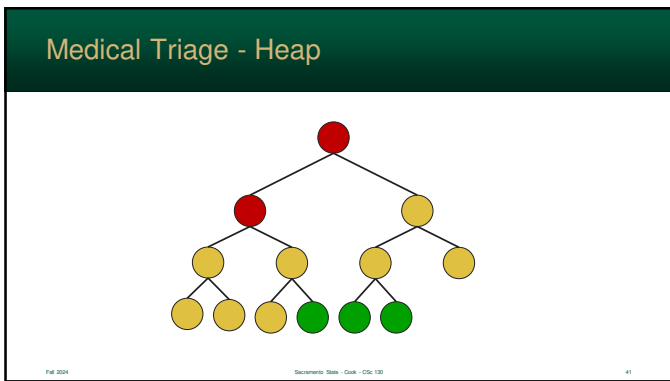
38

- ### Implementation with a Heap
- However, a priority queue can be implemented as a heap
 - Remember...
 - in a heap, all the items below a node have a greater value
 - so, *the root is the least item!*
 - heaps *naturally* implement a priority queue

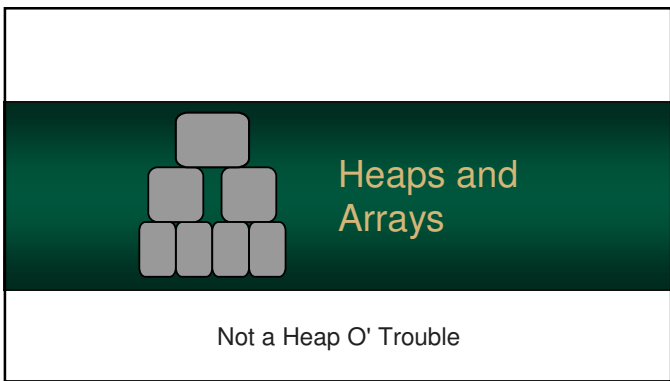
39

- ### Implementation with a Heap
- To enqueue an item...
 - just add to it the heap
 - it will up-heap to the correct position
 - requires $O(\log n)$
 - To dequeue an item...
 - just remove the root
 - requires $O(\log n)$ rebalance
-

40



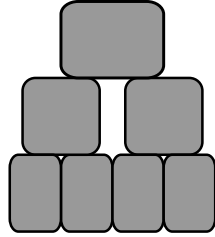
41



42

Heaps and Arrays

- Heaps are *complete*, balanced, binary trees
- This rigid, predictable, structure...
 - lends itself to being stored in an array
 - each node has a pre-ordained location



Fall 2024

Sacramento State - Oak - CS110

43

43

Heaps and Arrays

- Using an array, links between items are not explicitly stored
- Finding the location of an array item can be found using simple mathematics
- Heaps are no different - due to their predictable structure



Fall 2024

Sacramento State - Oak - CS110

44

44

Heaps and Arrays

- Any node's parent and children can be computed *mathematically*
- Heap ADTs only need to...
 - track the index of the end of the heap
 - all new items are added here – before upheap
 - and this is where the last item will be swapped for a deleted item (before it is downheaped)

Fall 2024

Sacramento State - Oak - CS110

45

45

Heap Array Math

Find	0 Indexed Array	1 Indexed Array
Parent of node i	$(i - 1) / 2$	$i / 2$
Left child of node i	$(2 * i) + 1$	$2 * i$
Right child of node i	$(2 * i) + 2$	$(2 * i) + 1$

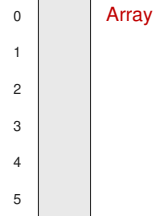
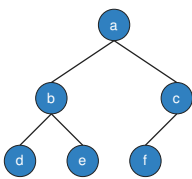
Fall 2024

Sacramento State - Oak - CS110

46

46

Heap in an Array



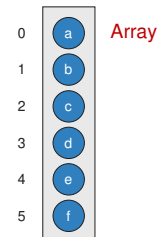
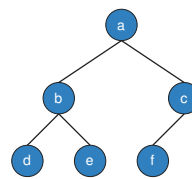
Fall 2024

Sacramento State - Oak - CS110

47

47

Heap in an Array



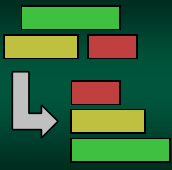
Fall 2024

Sacramento State - Oak - CS110

48

48

Heap Sort

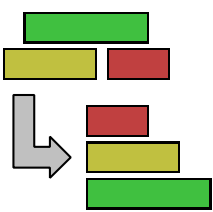


Heap-a-rific algorithm!

49

Heap Sort

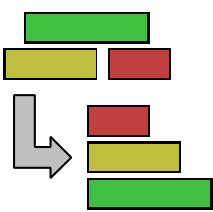
- *Heap Sort* is an **ingenious** algorithm that uses a max-heap to sort an array
- *John W. J. Williams* invented both heaps and the Heap Sort in 1964 (*O ABW*)
- The same year, the sort was improved by *Robert Floyd*



50

Heap Sort

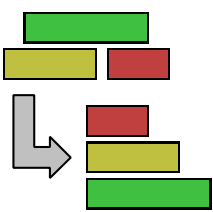
- Heap Sort takes advantage of the fact that a heap is a natural priority queue
- ... and that a heap will **always** add / remove from the right-most leaf



51

Heap Sort Works in Two Phases

- Phase 1: Heapify
 - converts the existing array into a **max-heap**
- Phase 2: Empty the heap
 - removes all the nodes (treating it as priority queue)
 - sorted data is added to the **end** of the array



52

Implementation

- Both the "heap" and the remaining array can be used in memory at the same time
- The sorted array is stored at the empty space **after** the end of the heap
- This concept works for both Phase 1 and Phase 2

53

Phase 1: Array → Heap

- In Phase 1, we convert the array into a max-heap. This step is called *heapify*.
- Remember....
 - a heap can be stored in an array
 - so, we can just **look at the array as a heap**
 - ...but, its not quite a heap yet
 - data needs to be rearranged to turn the array into a heap

54

How do we convert it?

- First approach: *top-down*
 - start building the heap at the top of the array
 - iterate i starting at 0 and build a heap above i
 - item are **upheaped**
- Second approach: *bottom-up*
 - fastest approach is to downheap all the leaves
 - run the **downheap**, at the root, all the leaves

Fall 2024

Srinivasan Sivas - Gosh - CS510

55

55

Phase 1: Heapify (top down)

```
procedure heapify(array, count)
  last = count - 1
  n = 0 //First item

  while (n <= last)
    upHeap(array, 0, n - 1)
    n++
  end while
end procedure
```

Fall 2024

Srinivasan Sivas - Gosh - CS510

56

56

Phase 1: Heapify (top down)

```
procedure heapify(array, count)
  last = count - 1
  n = last //last item

  while (n >= 0)
    downHeap(array, n + 1, last)
    n--
  end while
end procedure
```

Fall 2024

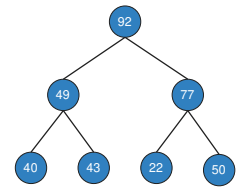
Srinivasan Sivas - Gosh - CS510

57

57

Phase 2: Root Deletion

- Now that the array is a **max-heap**, the root contains the maximum item
- If we remove the root, we have the **last item of the sorted array!**



Fall 2024

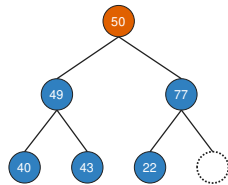
Srinivasan Sivas - Gosh - CS510

58

58

Phase 2: Root Deletion

- When we remove the root... right-most leaf is moved to the root
- ...and then downheaped into the correct position



Fall 2024

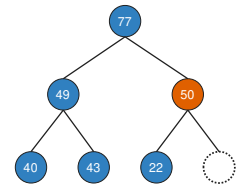
Srinivasan Sivas - Gosh - CS510

59

59

Phase 2: Root Deletion

- Now the root contains the second-largest item in the array
- This leaf position is now empty**



Fall 2024


Srinivasan Sivas - Gosh - CS510

60

60

Phase 2: Root Deletion

- We can put the root, that was just removed, in this new empty space
- *What a sec!* We just put the largest item in the last position in the array!




Fall 2024 Sacramento State - CS&E - CS110 61

61

Phase 2: Root Deletion

- So, to sort the array...
 - so, we just keep removing the root and placing it position where the leaf was located
 - the "heap" section of the array shrinks as the sorted array grows from the bottom
- *OMG! Sooooo, awesome!*



Fall 2024 Sacramento State - CS&E - CS110 62

62

Heap Sort Algorithm

```

last = count - 1
heapify(array, 0, last)

while (last > 0)
    swap array[0] and array[last]
    downHeap(0, last - 1)
    last--
end while
    
```

Fall 2024 Sacramento State - CS&E - CS110 63

63

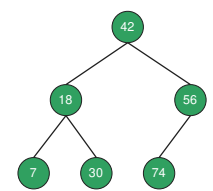
Phase 1 – Top-down Upheap

■ Heap (blue)

■ Array (green)

0	42
1	18
2	56
3	7
4	30
5	74

Conceptual View



Fall 2024 Sacramento State - CS&E - CS110 64

64

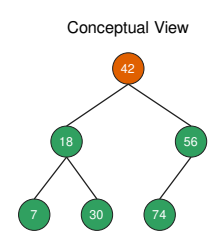
Phase 1 – Top-down Upheap

■ Heap (blue)

■ Array (green)

0	42
1	18
2	56
3	7
4	30
5	74

Conceptual View



Fall 2024 Sacramento State - CS&E - CS110 65

65

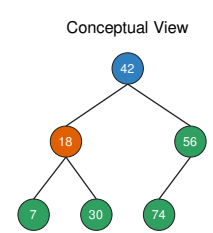
Phase 1 – Top-down Upheap

■ Heap (blue)

■ Array (green)

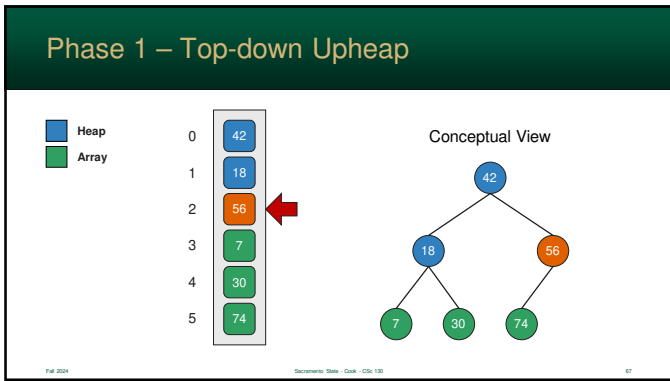
0	42
1	18
2	56
3	7
4	30
5	74

Conceptual View

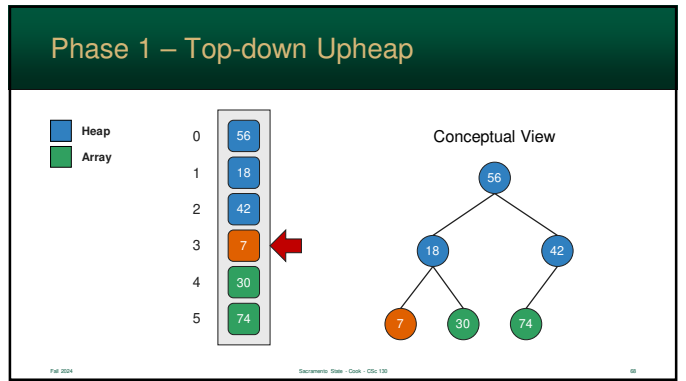


Fall 2024 Sacramento State - CS&E - CS110 66

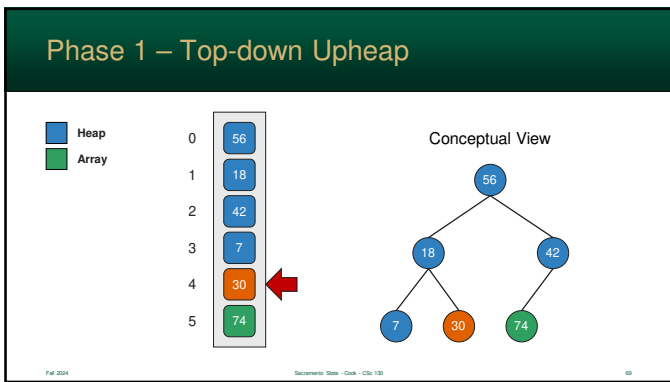
66



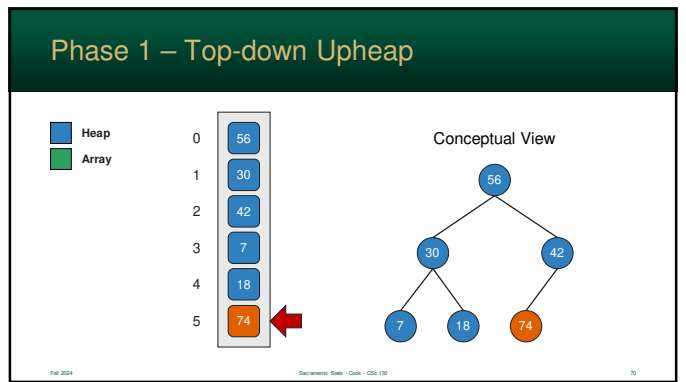
67



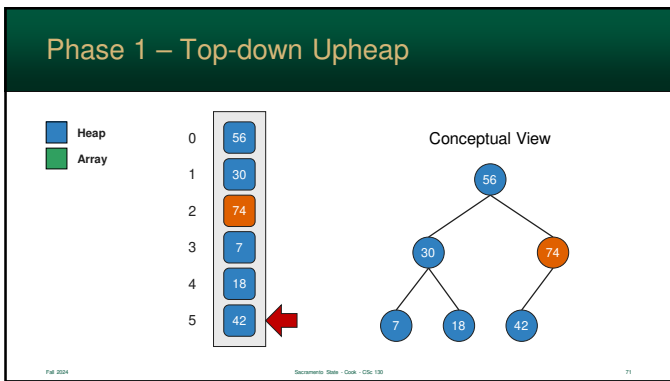
68



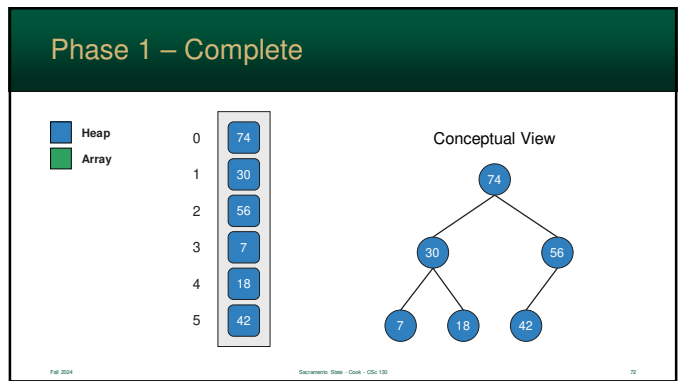
69



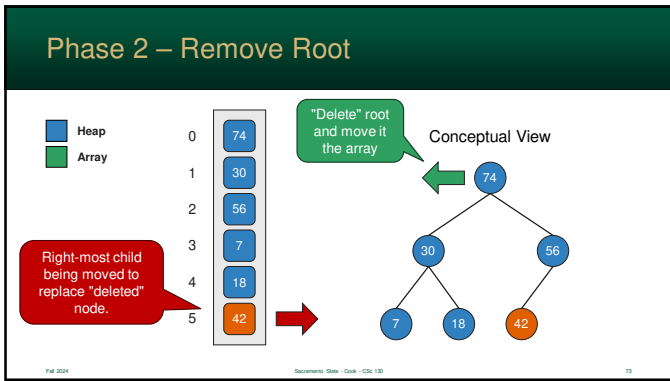
70



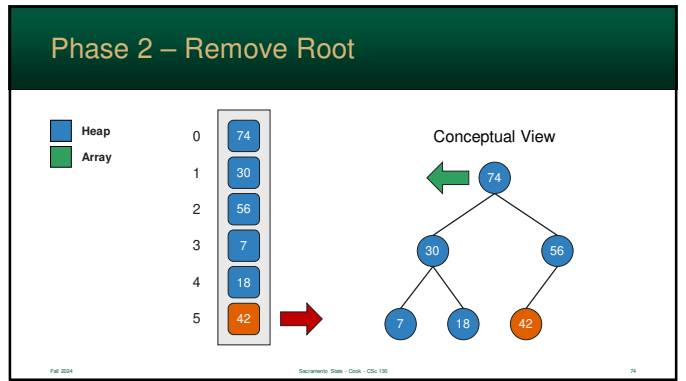
71



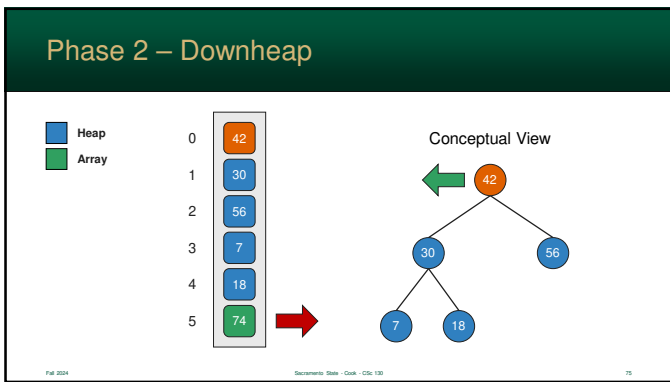
72



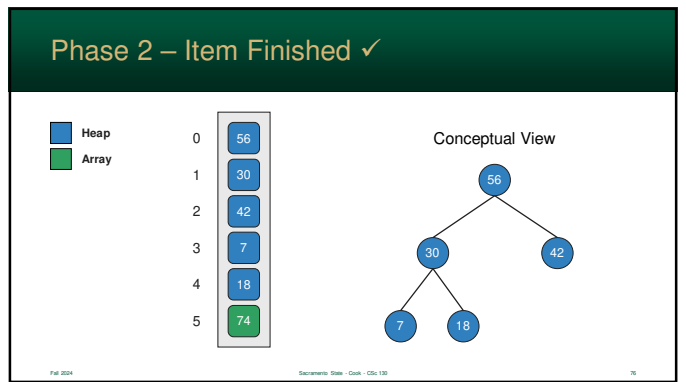
73



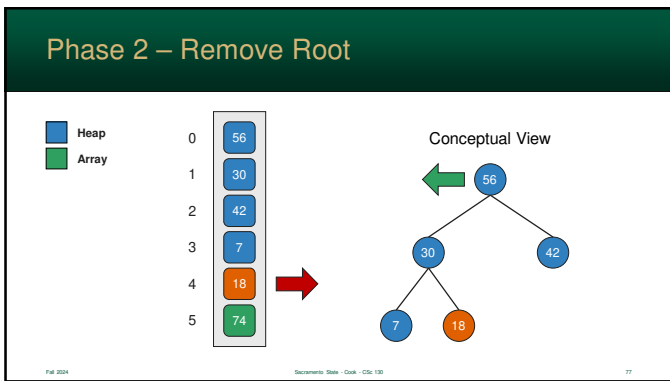
74



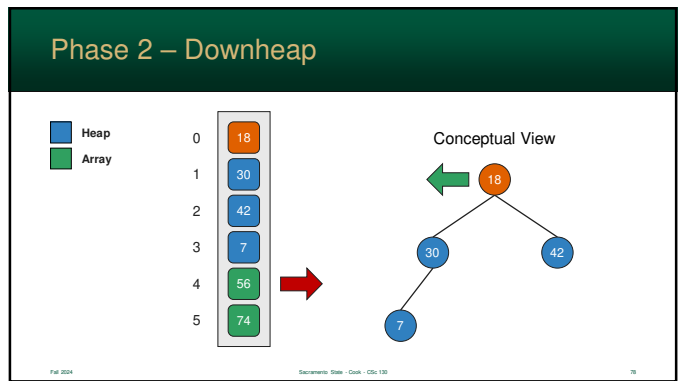
75



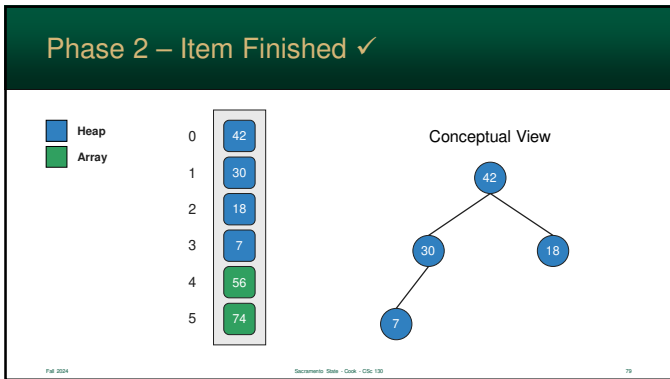
76



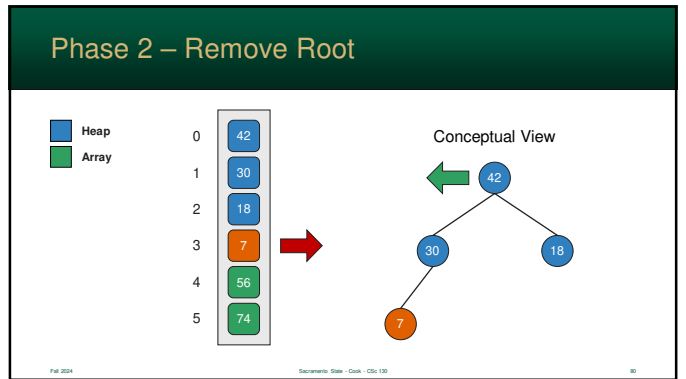
77



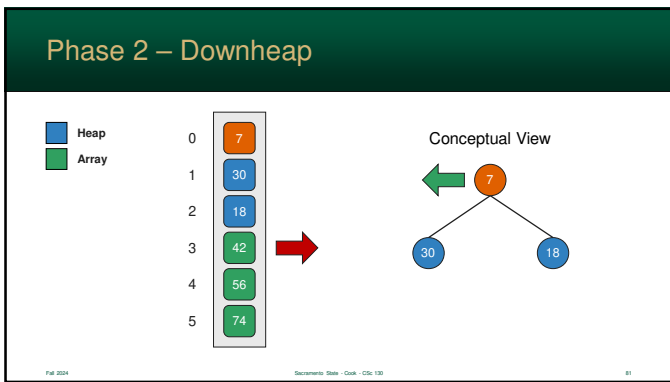
78



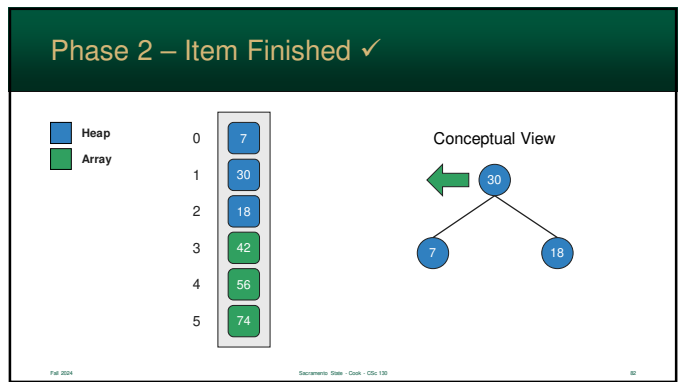
79



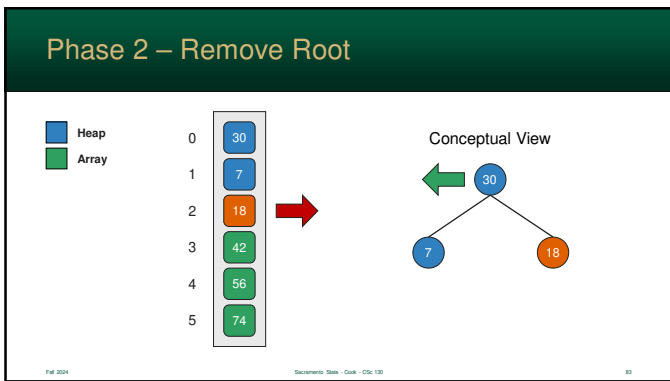
80



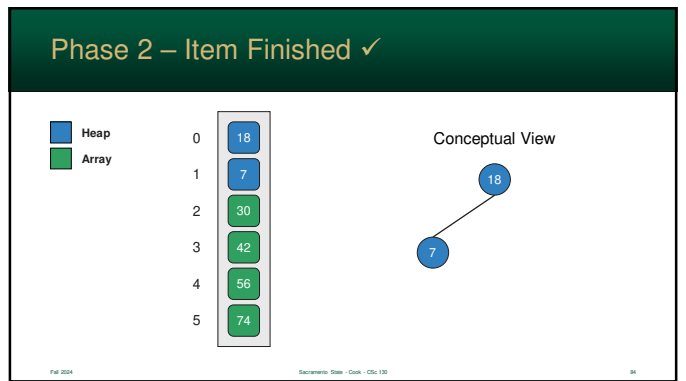
81



82



83



84

Phase 2 – Remove Root

Legend:
 ■ Heap (Blue)
 ■ Array (Green)

Array:
 0: 18
 1: 7
 2: 30
 3: 42
 4: 56
 5: 74

Conceptual View:
 18 (Blue circle)
 7 (Orange circle)

Fall 2024 Sacramento State - CS&E - CS110 85

85

Phase 2 – Remove Root

Legend:
 ■ Heap (Blue)
 ■ Array (Green)

Array:
 0: 7
 1: 18
 2: 30
 3: 42
 4: 56
 5: 74

Conceptual View:
 7 (Orange circle)
 18 (Blue circle)

Fall 2024 Sacramento State - CS&E - CS110 86

86

Complete

Legend:
 ■ Heap (Blue)
 ■ Array (Green)

Array:
 0: 7
 1: 18
 2: 30
 3: 42
 4: 56
 5: 74

Conceptual View:

Fall 2024 Sacramento State - CS&E - CS110 87

87

Merge Sort vs. Heap Sort

- Heap-Sort allows us to sort any array in $O(n \log n)$ just like Merge-Sort & Quicksort
- However, there is no overhead
 - Heap-Sort can be sorted in-place, meaning auxiliary storage is $O(1)$
 - Merge-Sort, however, requires $O(n)$
 - Quick-Sort can become $O(n^2)$

Fall 2024 Sacramento State - CS&E - CS110 88

88

Merge Sort vs. Heap Sort

- However, in some cases, the recursive nature of Merge Sort is better
 - easy to distribute to multiple computers
 - Heap-Sort uses the entire array – *not online*
- But...in the Real World, it gets complex
 - you can cut an array into sub-lists, send them to different machines which Heap-Sort them
 - ... and then you Merge

Fall 2024 Sacramento State - CS&E - CS110 89

89

Heap Sort Summary

Heap Sort	
Time Average	$O(n \log n)$
Time Best	$O(n \log n)$
Time Worst	$O(n \log n)$
Auxiliary space	$O(1)$
Stable	No – Equal element order not preserved
Online?	No

Fall 2024 Sacramento State - CS&E - CS110 90

90

Summary of Sorting Algorithms

Sort Algorithm	Best	Average	Worst	Aux. Storage
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Shell	$O(n \log n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(1)$
Heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
Radix	$O(k \times n)$	$O(k \times n)$	$O(k \times n)$	$O(b \times n)$

91