

BuildHeap and Heapsort

Goals

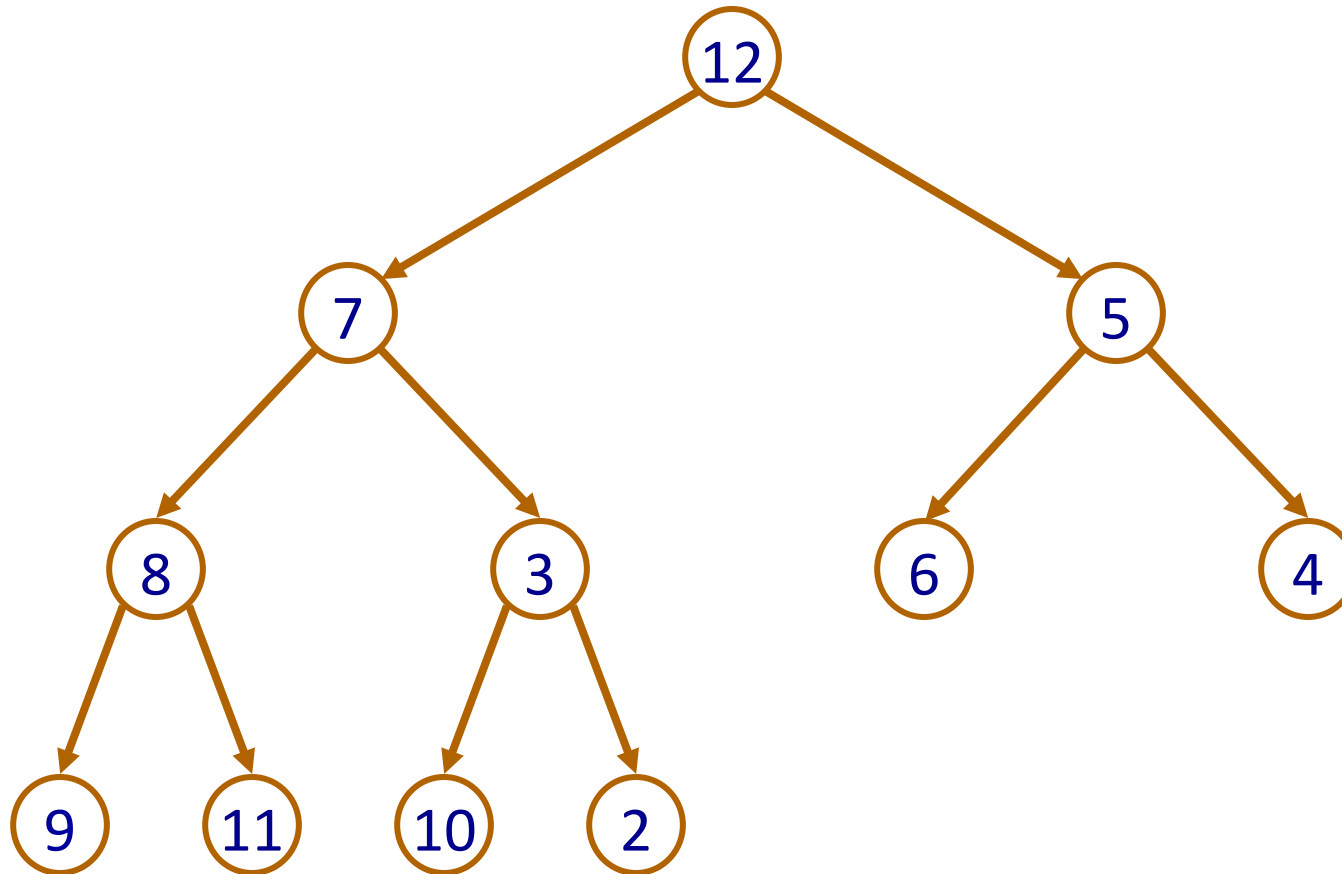
- Build a heap from an array of arbitrary values
- HeapSort algorithm
- Analysis of HeapSort

BuildHeap

- How do we build a heap from an arbitrary array of values???

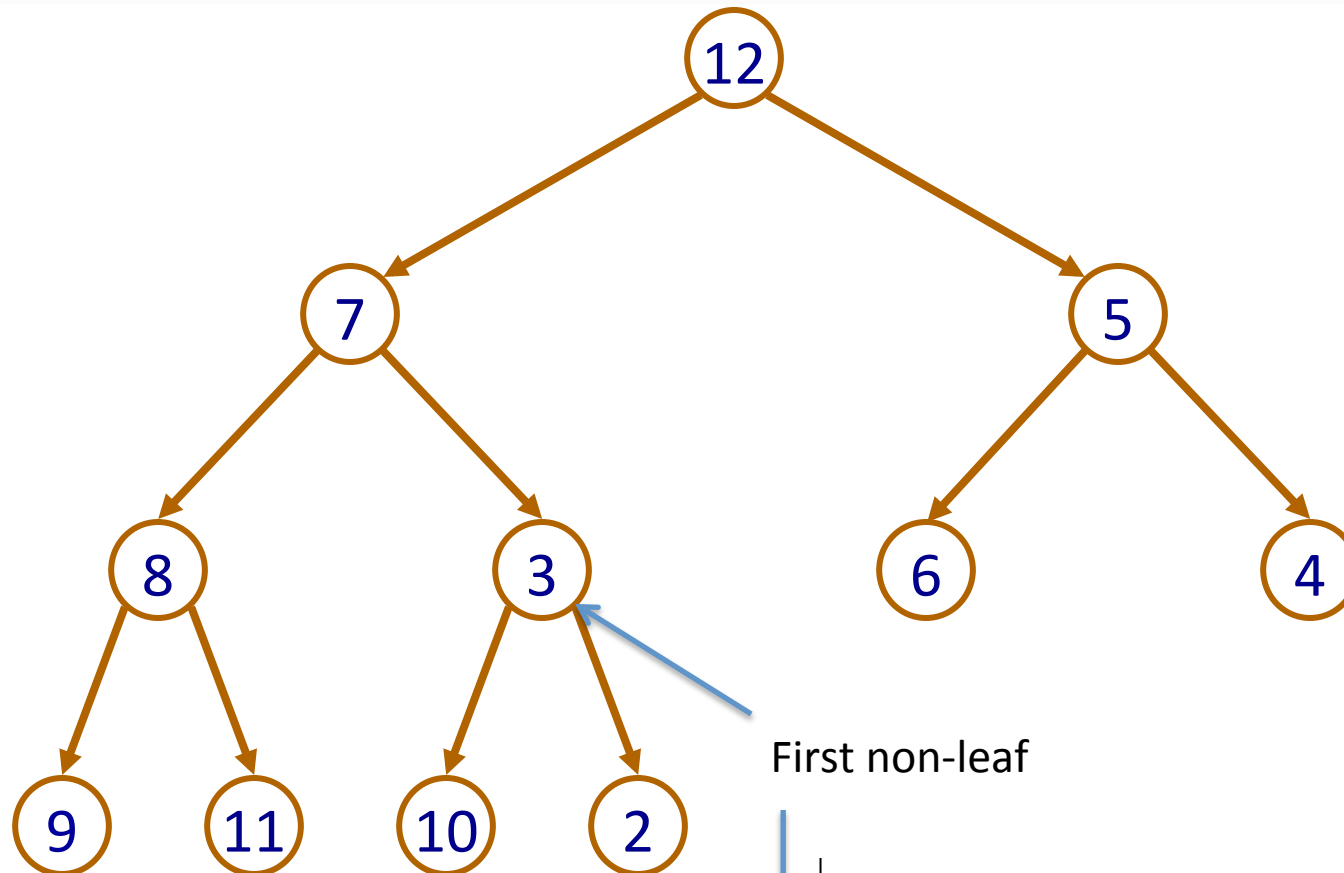
BuildHeap: Is this a proper heap?

0	1	2	3	4	5	6	7	8	9	10
12	7	5	8	3	6	4	9	11	10	2



Are any of the subtrees ***guaranteed*** to be proper heaps?

BuildHeap: Leaves are proper heaps



First non-leaf

Size = 11

Size/2 - 1 = 4

0	1	2	3	4	5	6	7	8	9	10	11
12	7	5	8	3	6	4	9	11	10	2	

BuildHeap

- How can we use this information to build a heap from a random array?
- `_adjustHeap`: takes a binary tree, rooted at a node, and percolates down from that node to ensure that the subtree is a proper heap

`void _adjustHeap(struct DynArr *heap, int max, int pos)`

Adjust up to
(not inclusive)

Adjust from

BuildHeap

- Find the last non-leaf node, i , (going from left to right)
- adjust heap from it to max
- Decrement i and repeat until you process the root

HeapSort

- BuildHeap and _adjustHeap are the keys to an efficient , in-place, sorting algorithm
- in-place means that we don't require any extra storage for the algorithm
- Any ideas???

HeapSort

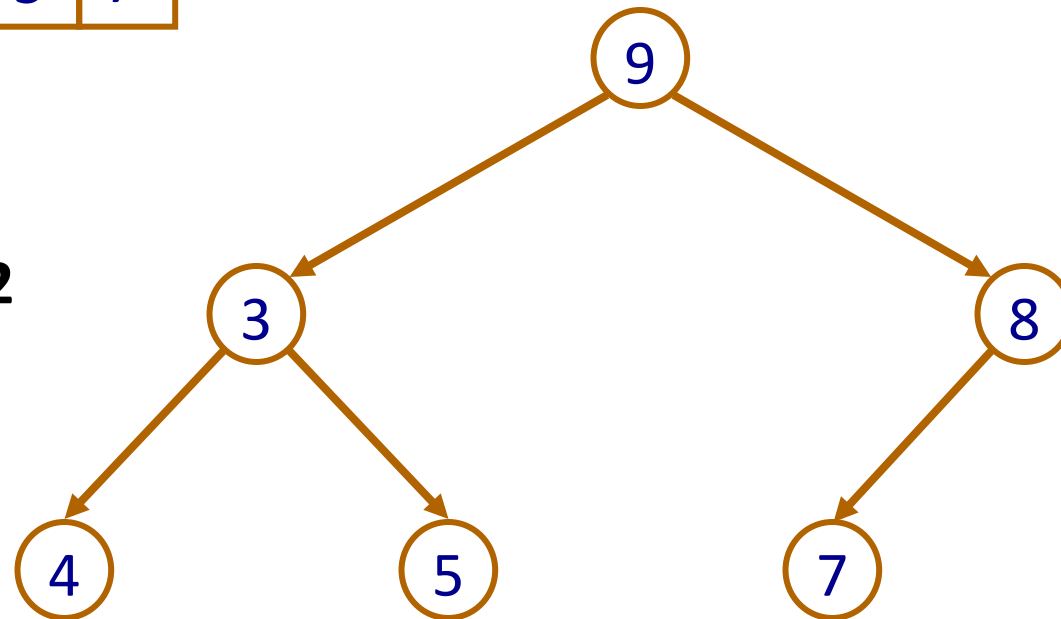
1. BuildHeap – turn arbitrary array into a heap
2. Swap first and ***last*** elements
3. Adjust Heap (from 0 to the ***last***...not inclusive!)
4. Repeat 2-3 but decrement ***last*** each time through

HeapSort Simulation: BuildHeap

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

Max = 6

$i = \text{Max}/2 - 1 = 2$



$i=2$

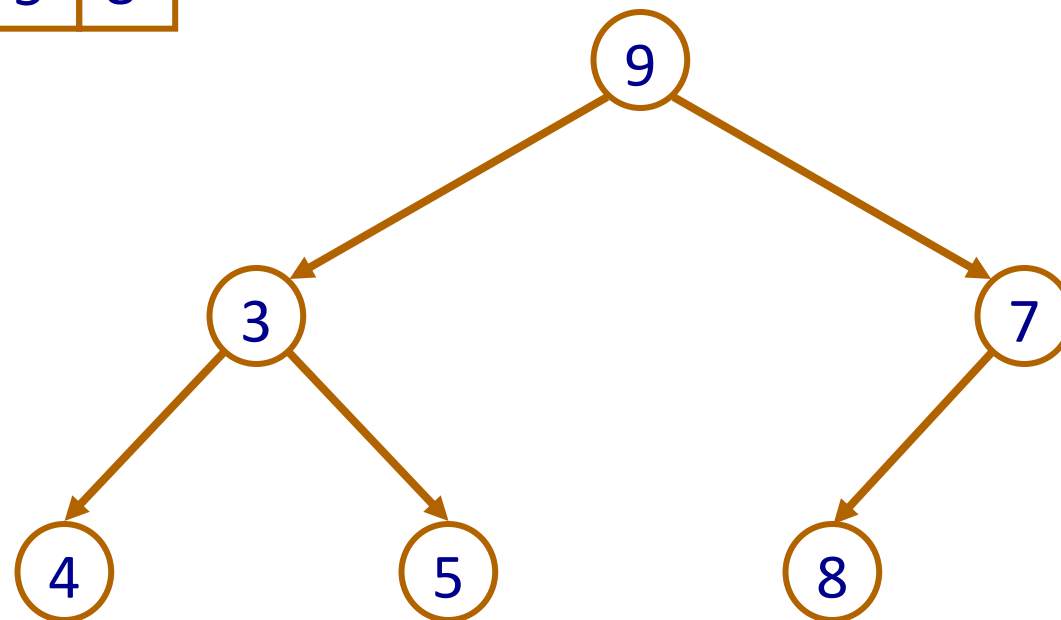
`_adjustHeap(heap,6,2)`

HeapSort Simulation: BuildHeap

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

Max = 6

i=1



i=1

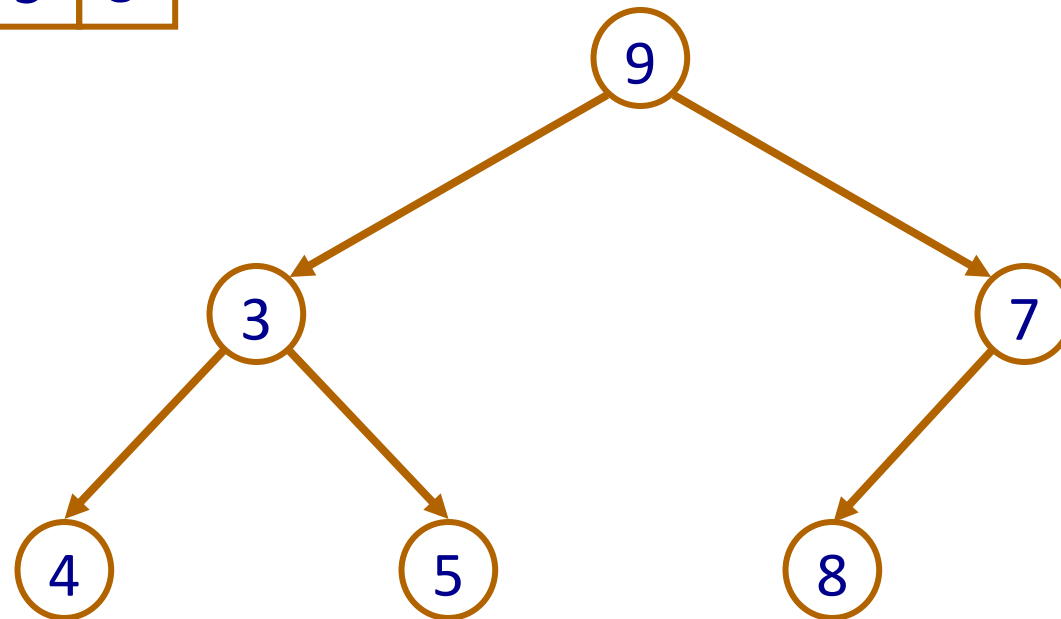
`_adjustHeap(heap,6,1)`

HeapSort Simulation: BuildHeap

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

Max = 6

i = 0

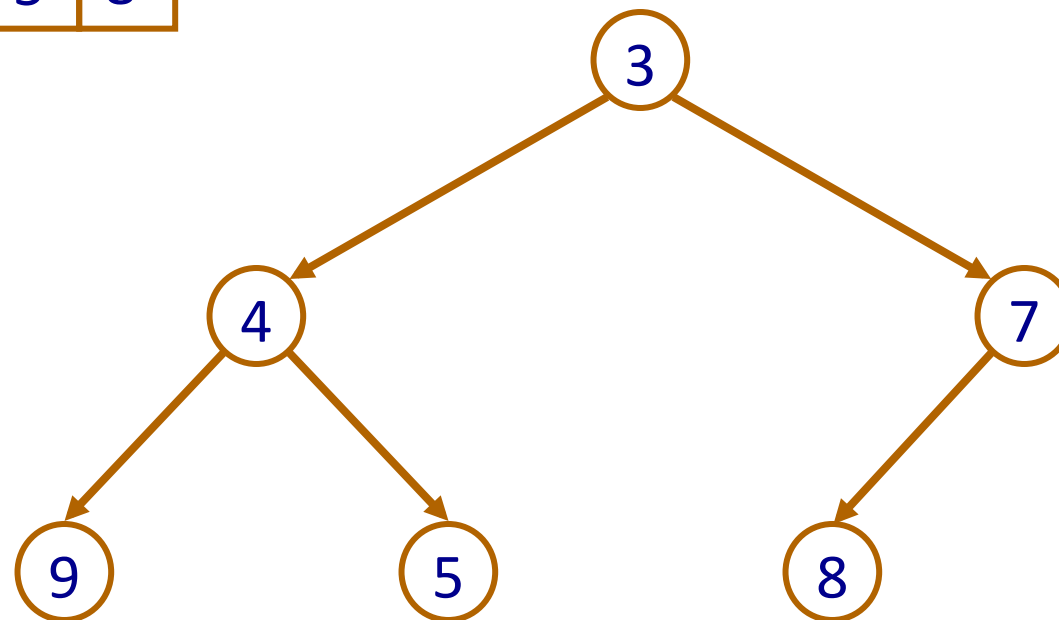


i=0

_adjustHeap(heap,6,0)

HeapSort Simulation: BuildHeap

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



i=-1

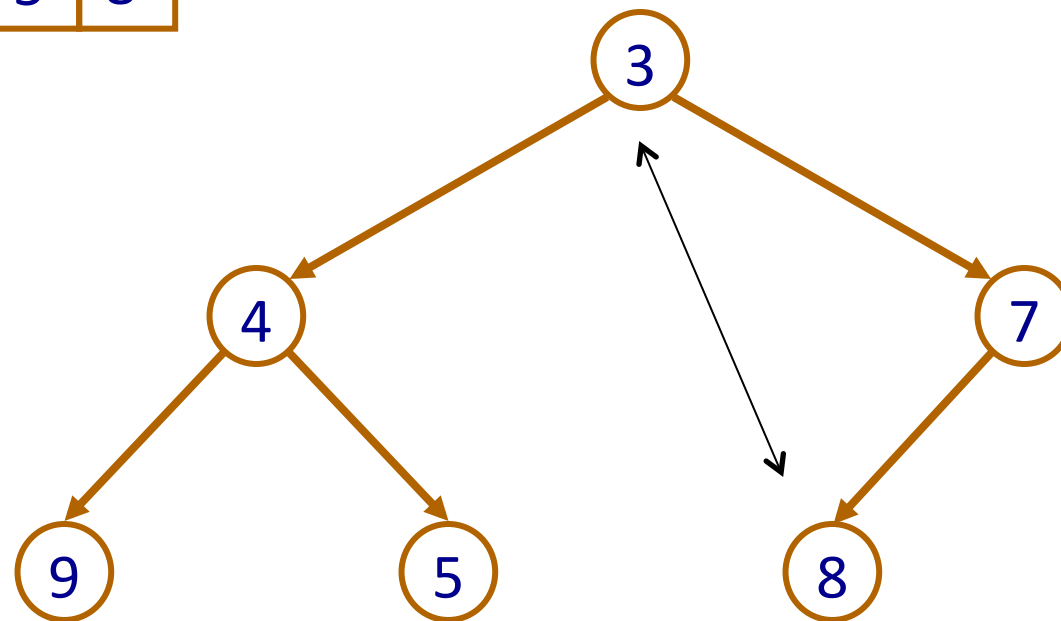
Done...with BuildHeapnow let's sort it!

HeapSort

1. BuildHeap
2. Swap first and last
3. Adjust Heap (from 0 to the last)
4. Repeat 2-3 but decrement last

HeapSort Simulation: Sort in Place

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



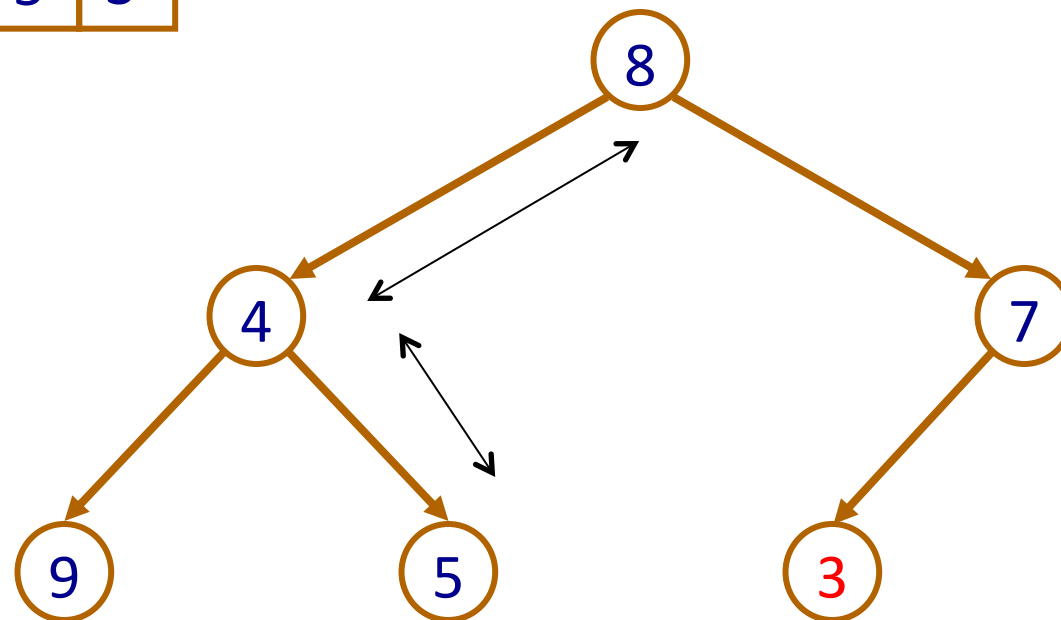
i=5

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



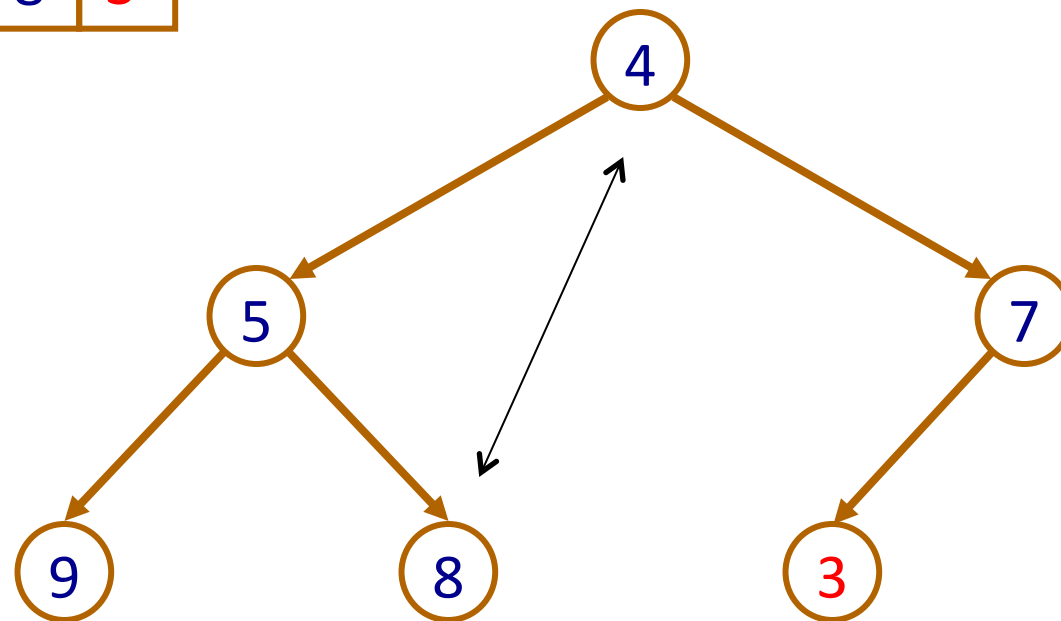
$i=5$

Swap(v, 0, i)

`_adjustHeap(v, i, 0);`

HeapSort Simulation: Sort in Place

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



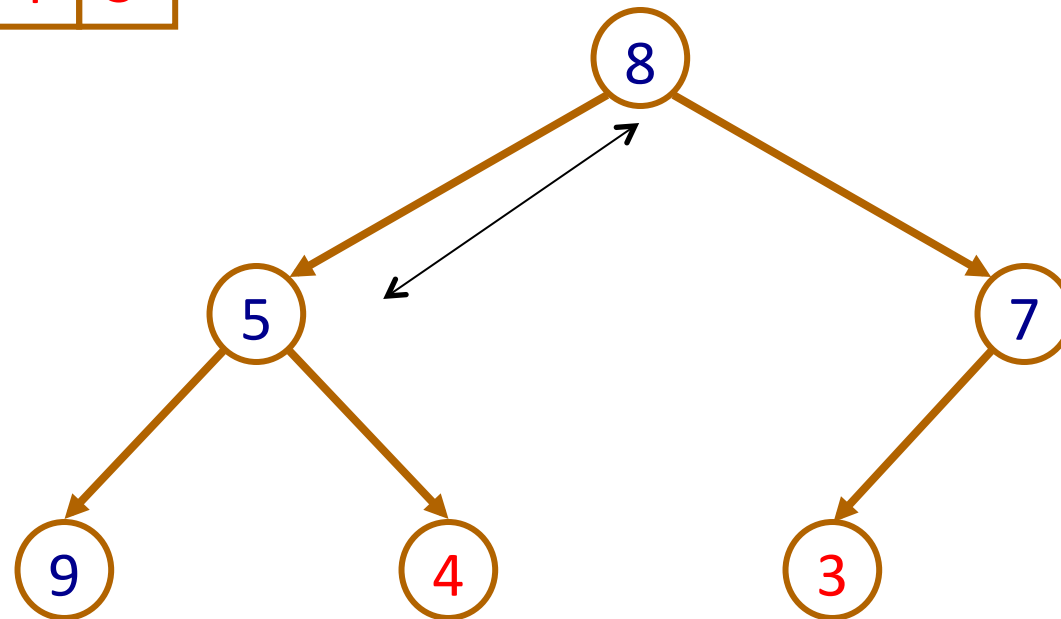
i=4

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



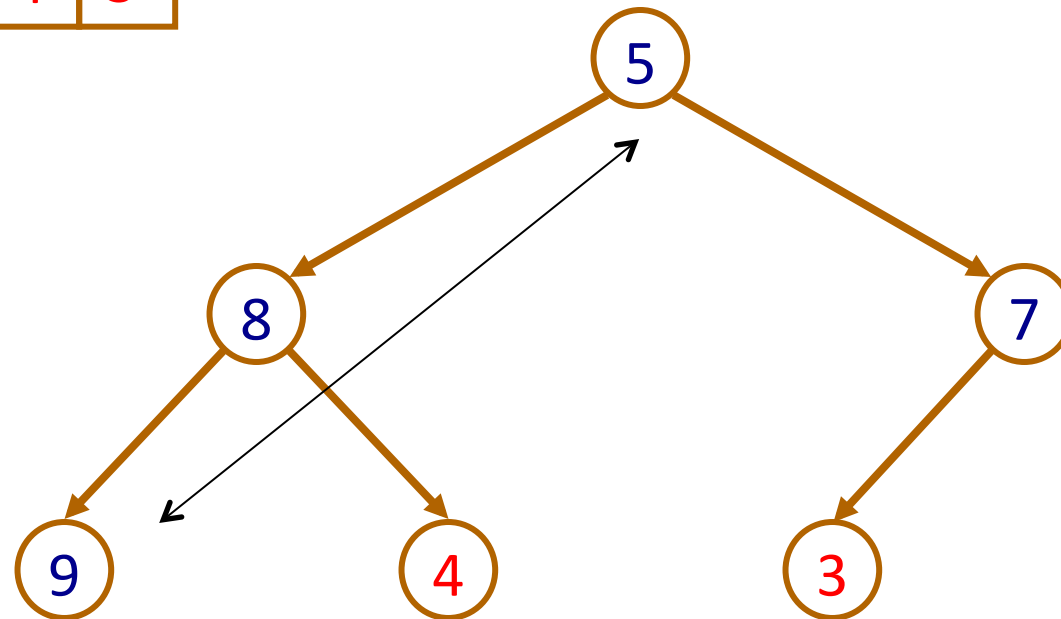
$i=4$

`Swap(v, 0, i)`

`_adjustHeap(v, i, 0);`

HeapSort Simulation: Sort in Place

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



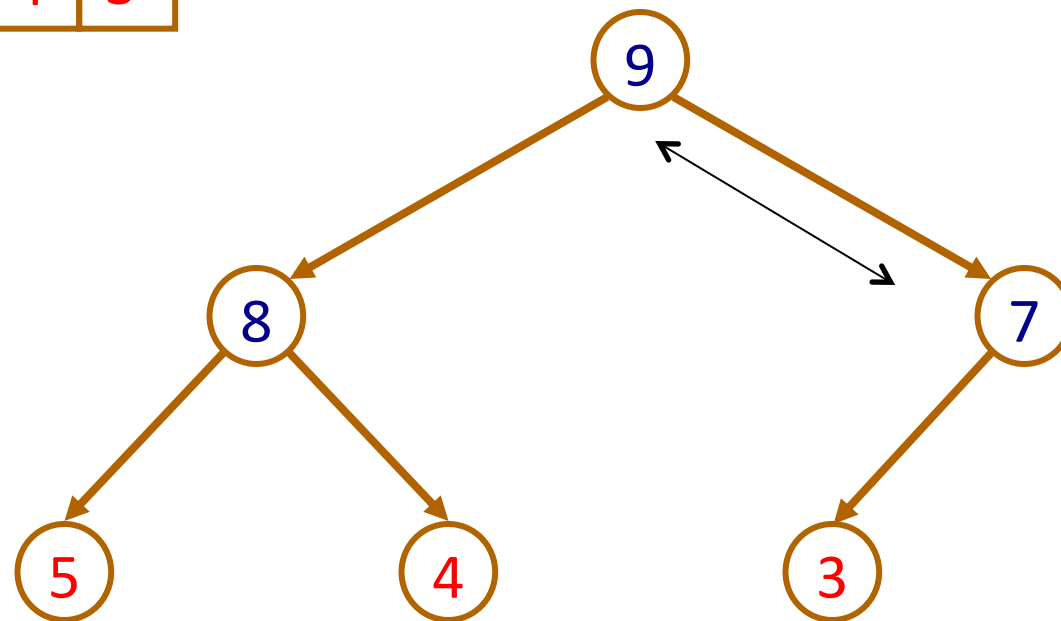
i=3

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



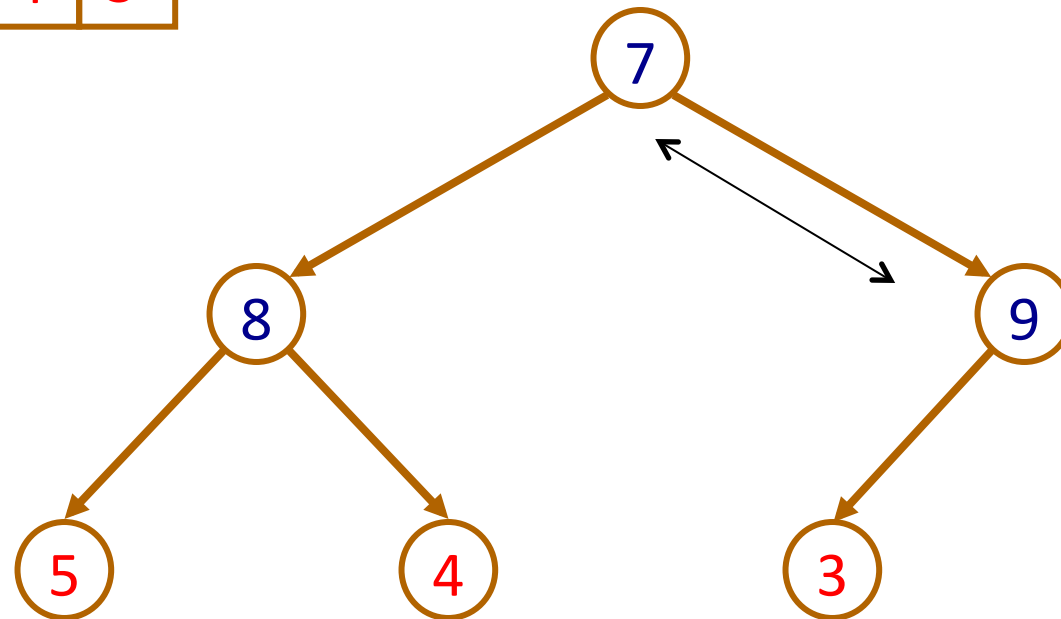
i=3

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



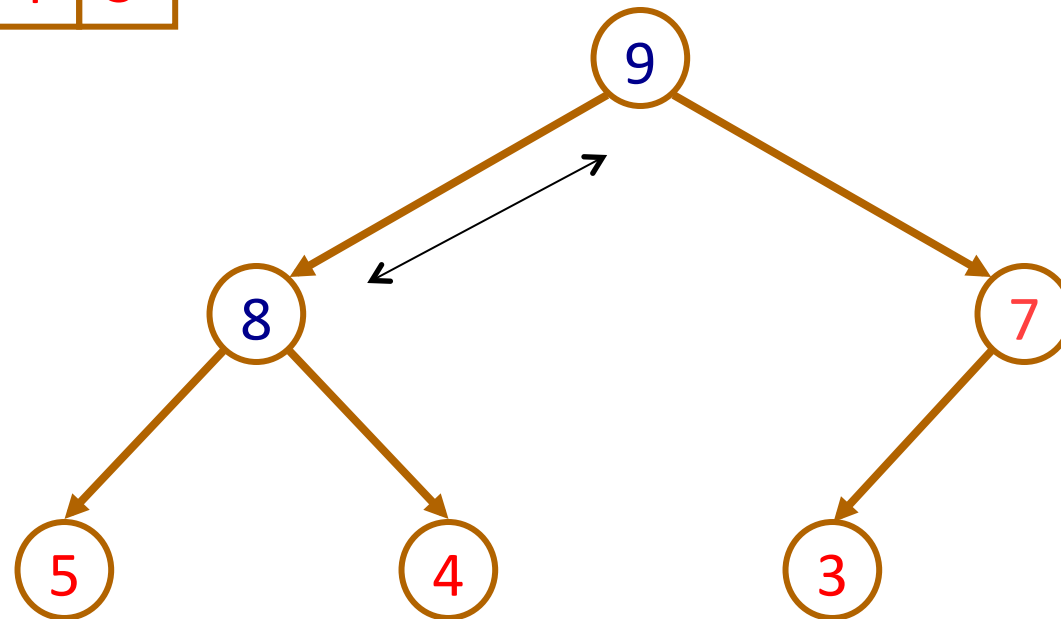
i=2

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



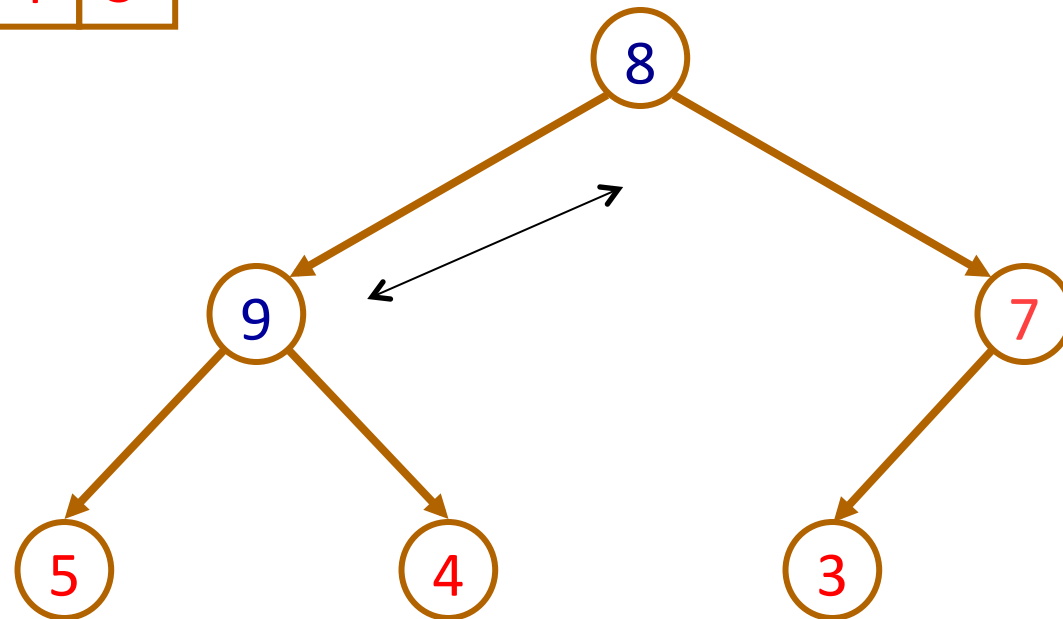
$i=2$

Swap(v, 0, i)

`_adjustHeap(v, i, 0);`

HeapSort Simulation: Sort in Place

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



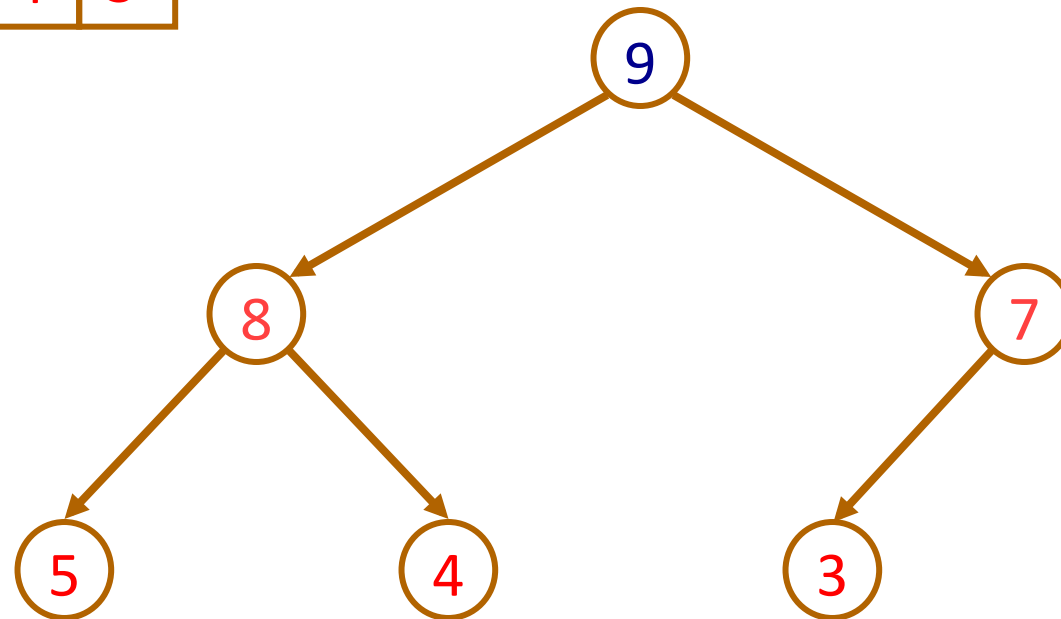
i=1

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



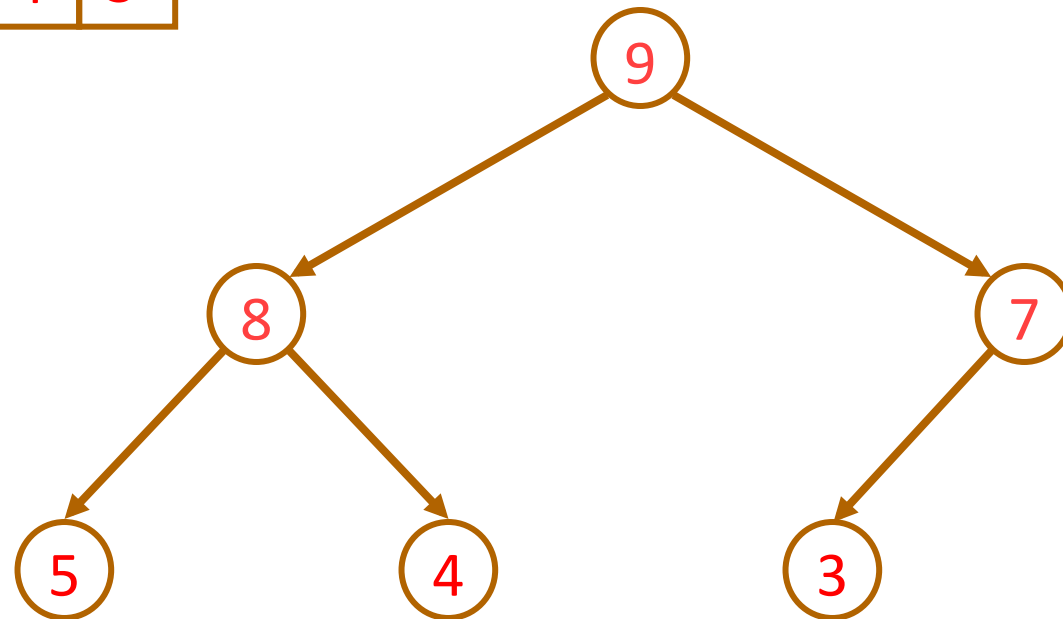
i=1

Swap(v, 0, i)

_adjustHeap(v, i, 0);

HeapSort Simulation: Sort in Place

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



i=0
DONE

HeapSort Performance

- Build Heap:

$n/2$ calls to `_adjustHeap` = $O(n \log n)$

- HeapSort:

n calls to swap and adjust = $O(n \log n)$

- Total:

$O(n \log n)$

Your Turn

- Worksheet 34 – BuildHeap and Heapsort