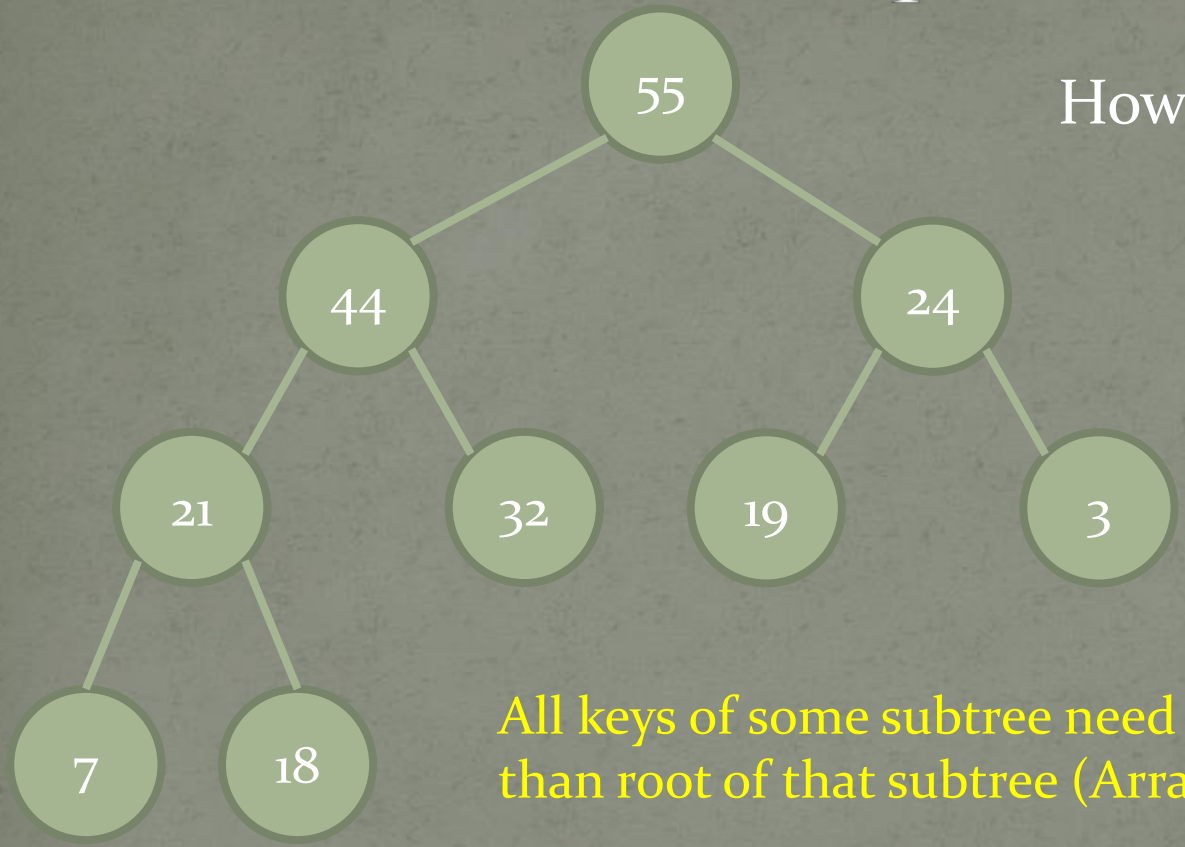
Algorithms and Data Structures

Andrzej Pisarski

Plan of the lecture

- Heap (~~sterta~~, kopiec; español: montículo)
  - Introduction
  - Example of implementations of heap
  - Heap based on tree
  - Heapsort

Introduction – Heap max



How to obtain index of:

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

All keys of some subtree need to have value not bigger than root of that subtree (Array[i]<=Array[Father[i]])

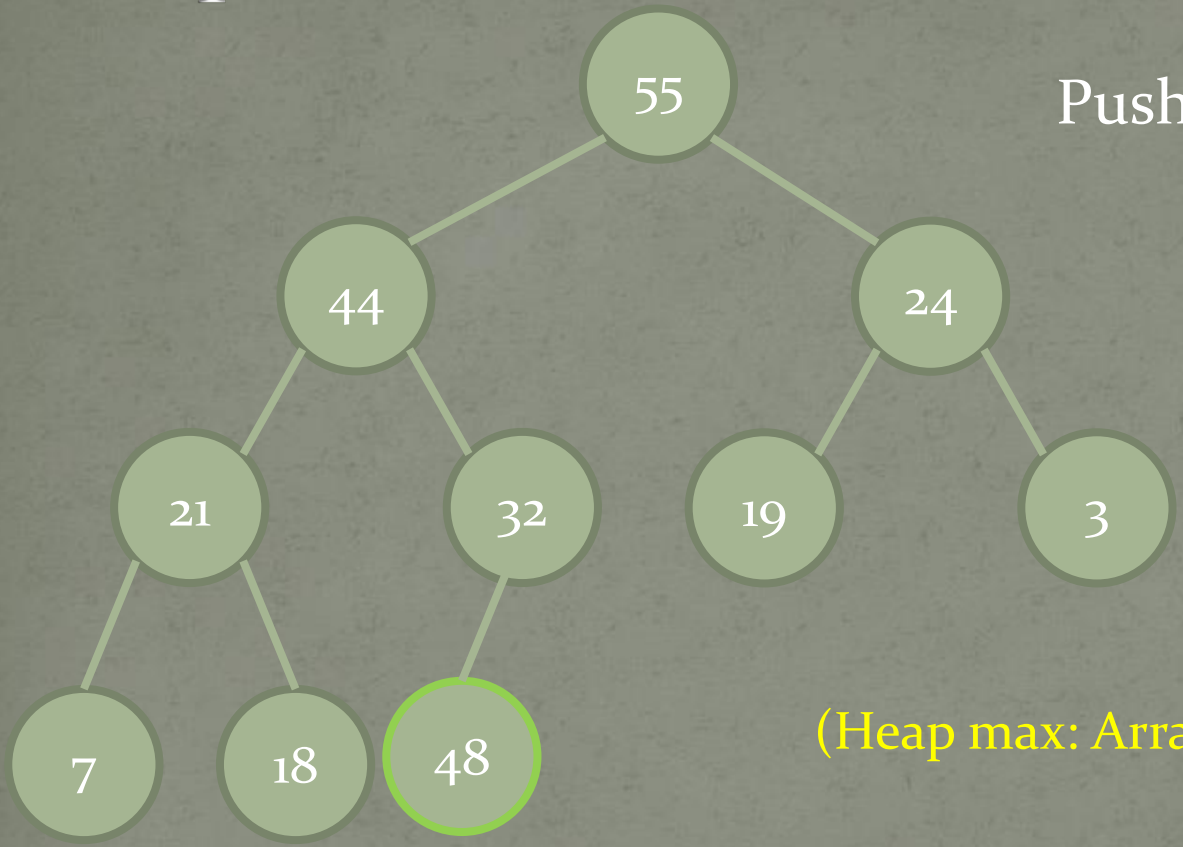
0	1	2	3	4	5	6	7	8	9	10	11
	55	44	24	21	32	19	3	7	18		

Example of implementations of heap

- Priority queues
  - types (max, min)
  - insertion (pushing) a new item
  - finding an item with max or min key
  - deleting (popping)

Heap based on tree

Heap based on tree



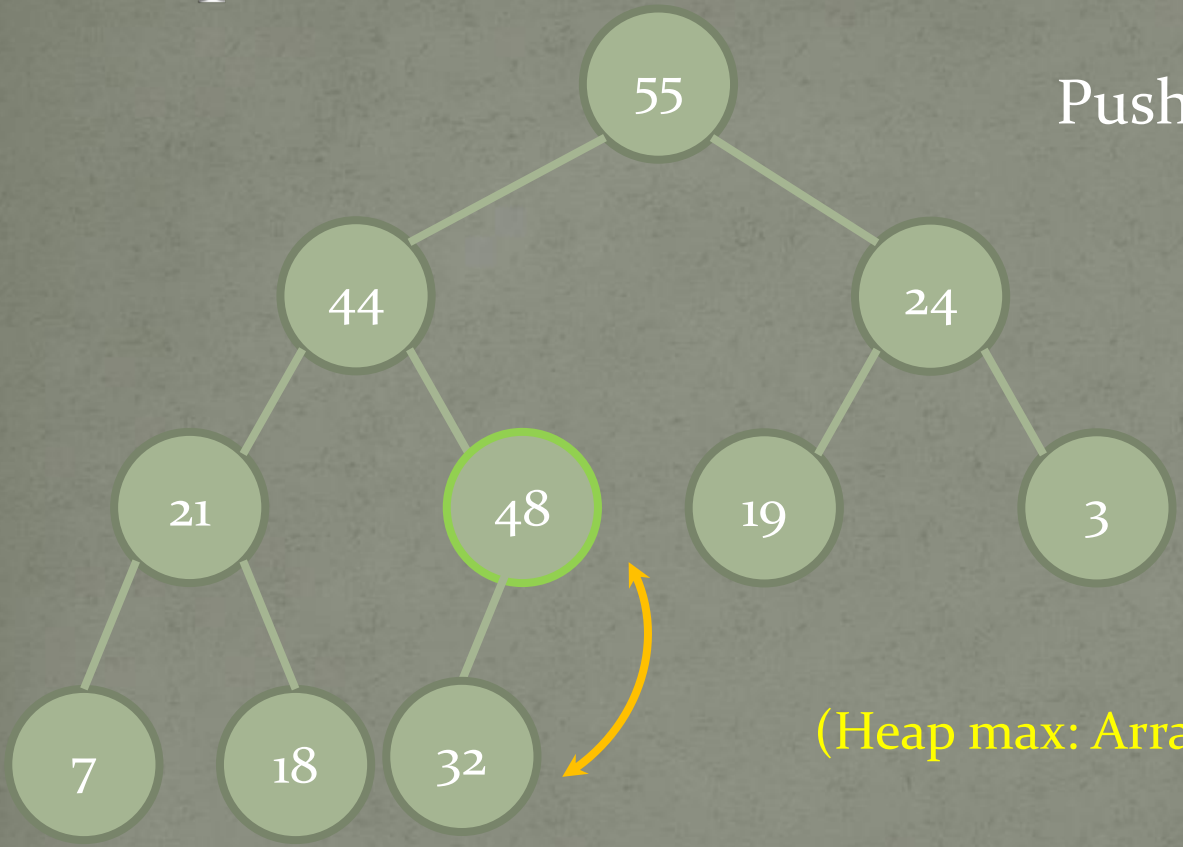
Pushing a new item (48)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	55	44	24	21	32	19	3	7	18	48	

Heap based on tree



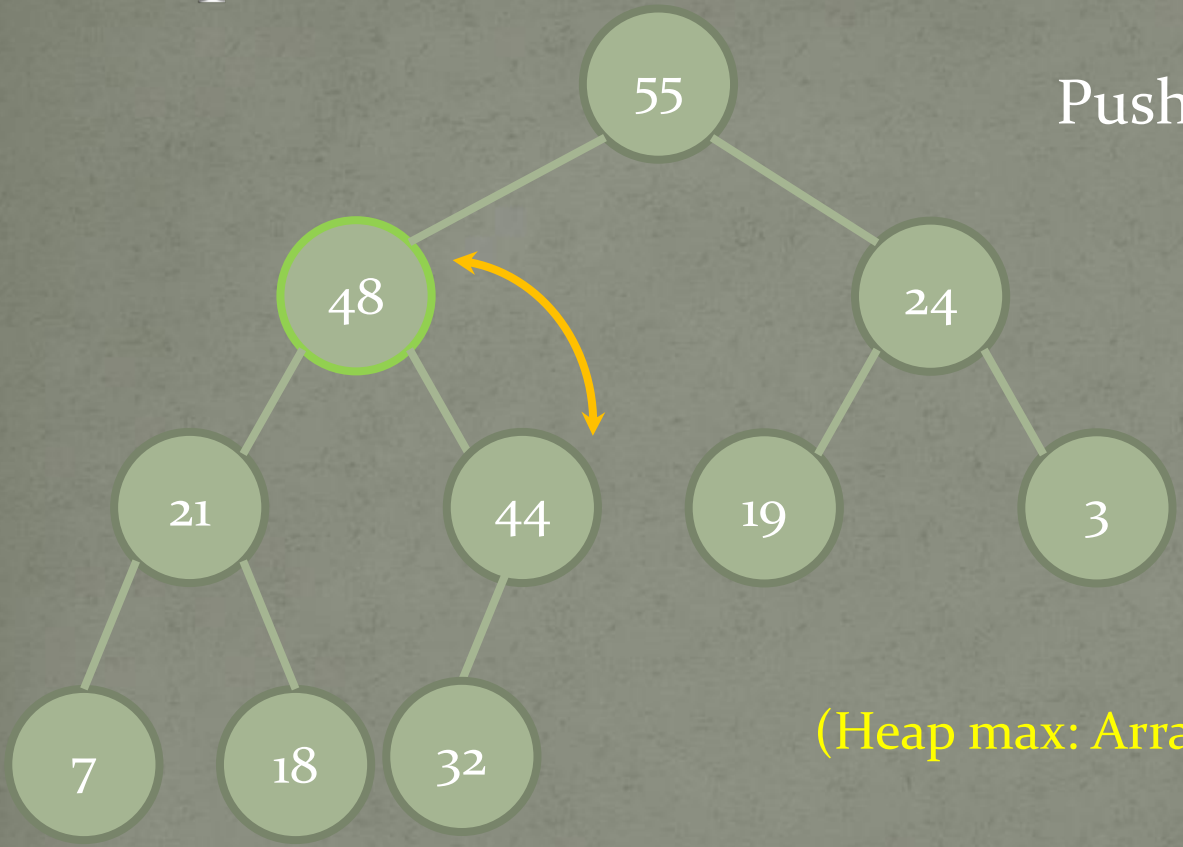
Pushing a new item (48)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	55	44	24	21	48	19	3	7	18	32	

Heap based on tree



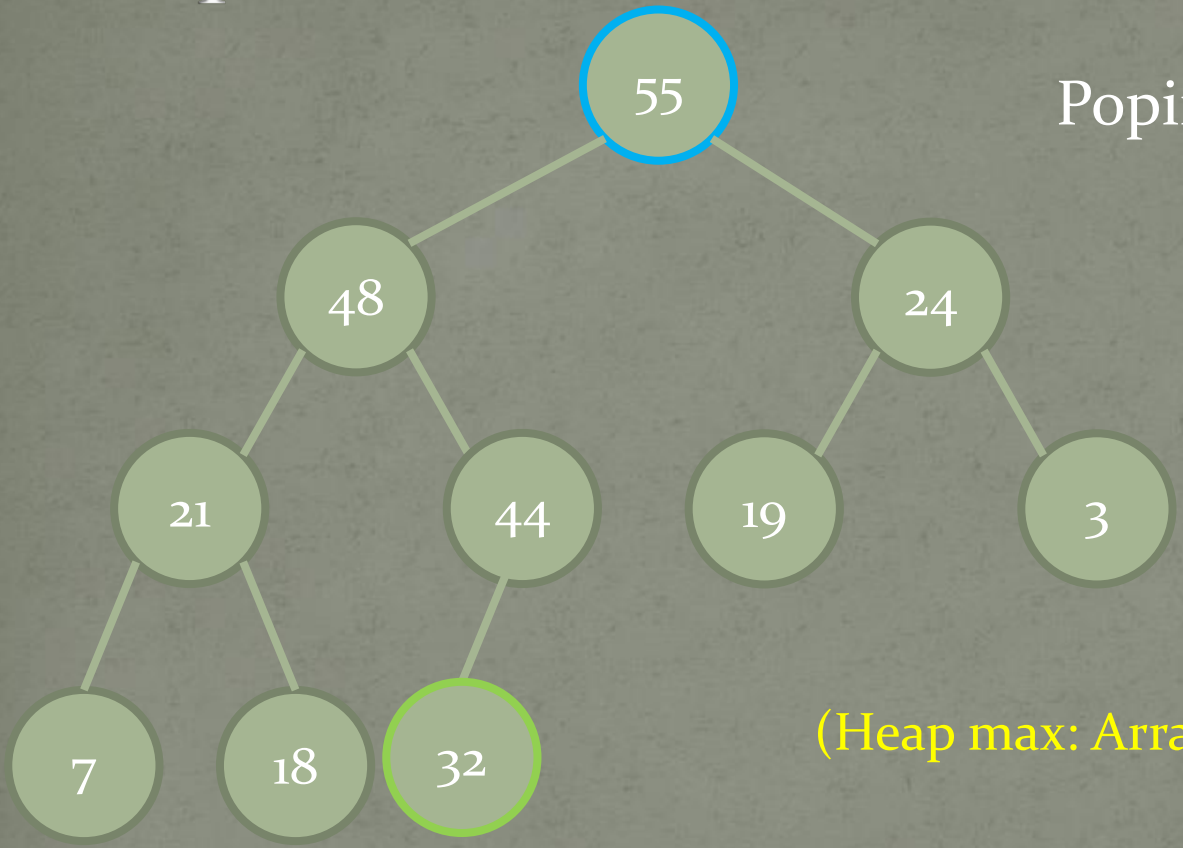
Pushing a new item (48)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	55	48	24	21	44	19	3	7	18	32	

Heap based on tree



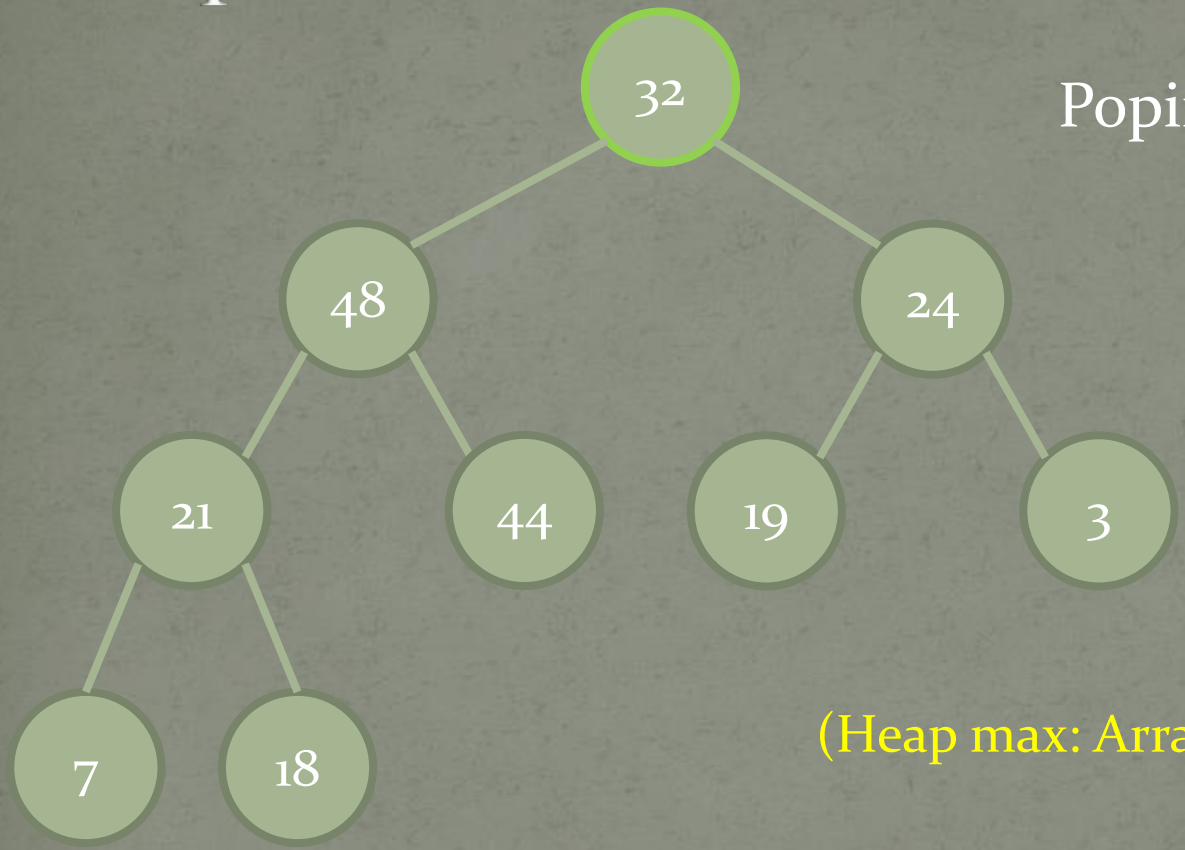
Poping an item (55)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	55	48	24	21	44	19	3	7	18	32	

Heap based on tree



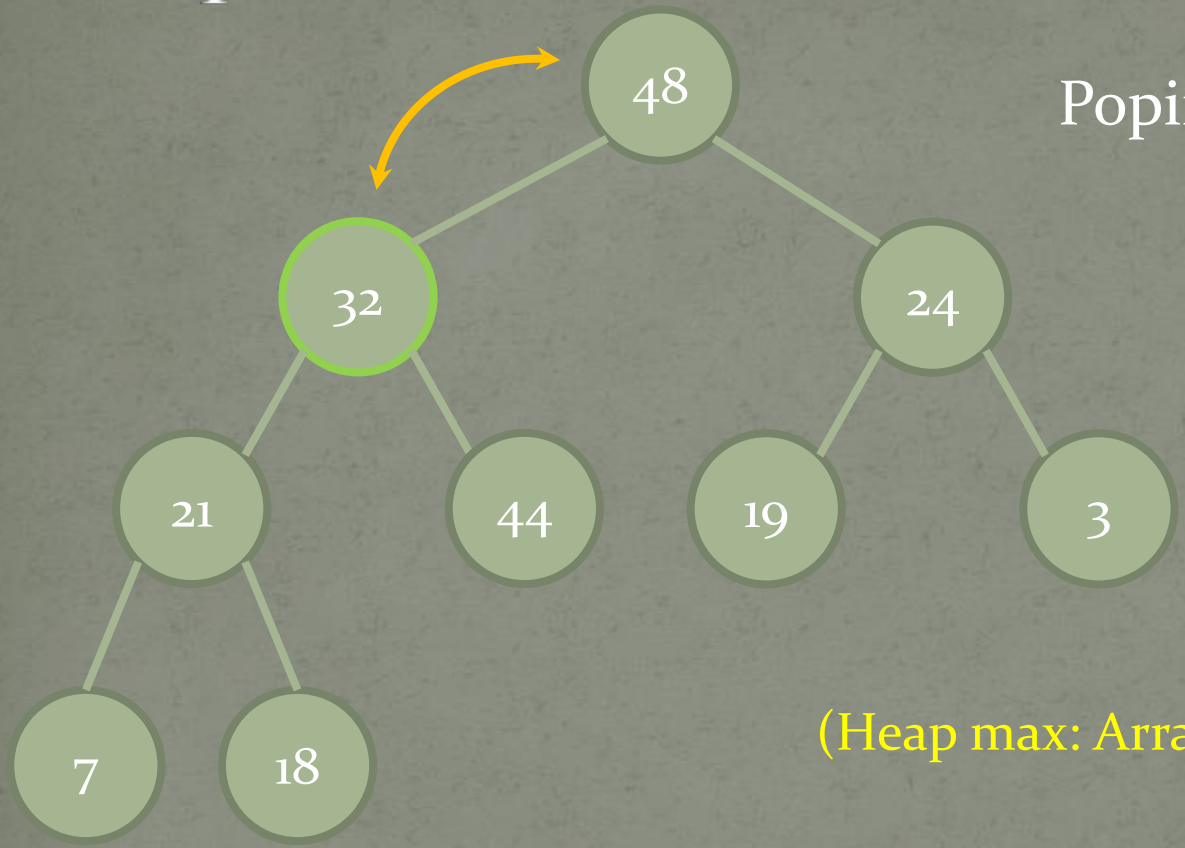
Poping an item (55)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	32	48	24	21	44	19	3	7	18		

Heap based on tree



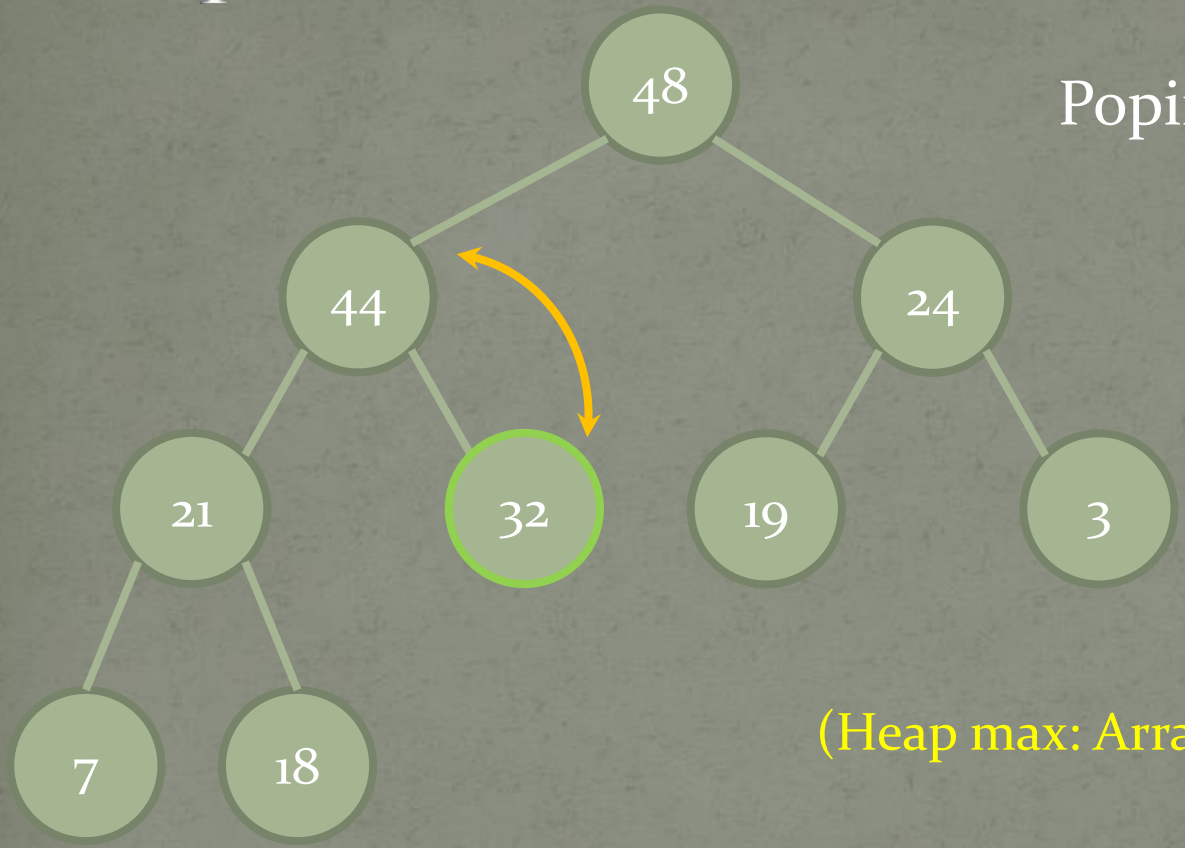
Poping an item (55)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	48	32	24	21	44	19	3	7	18		

Heap based on tree



Poping an item (55)

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	48	44	24	21	32	19	3	7	18		

Heapsort

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;

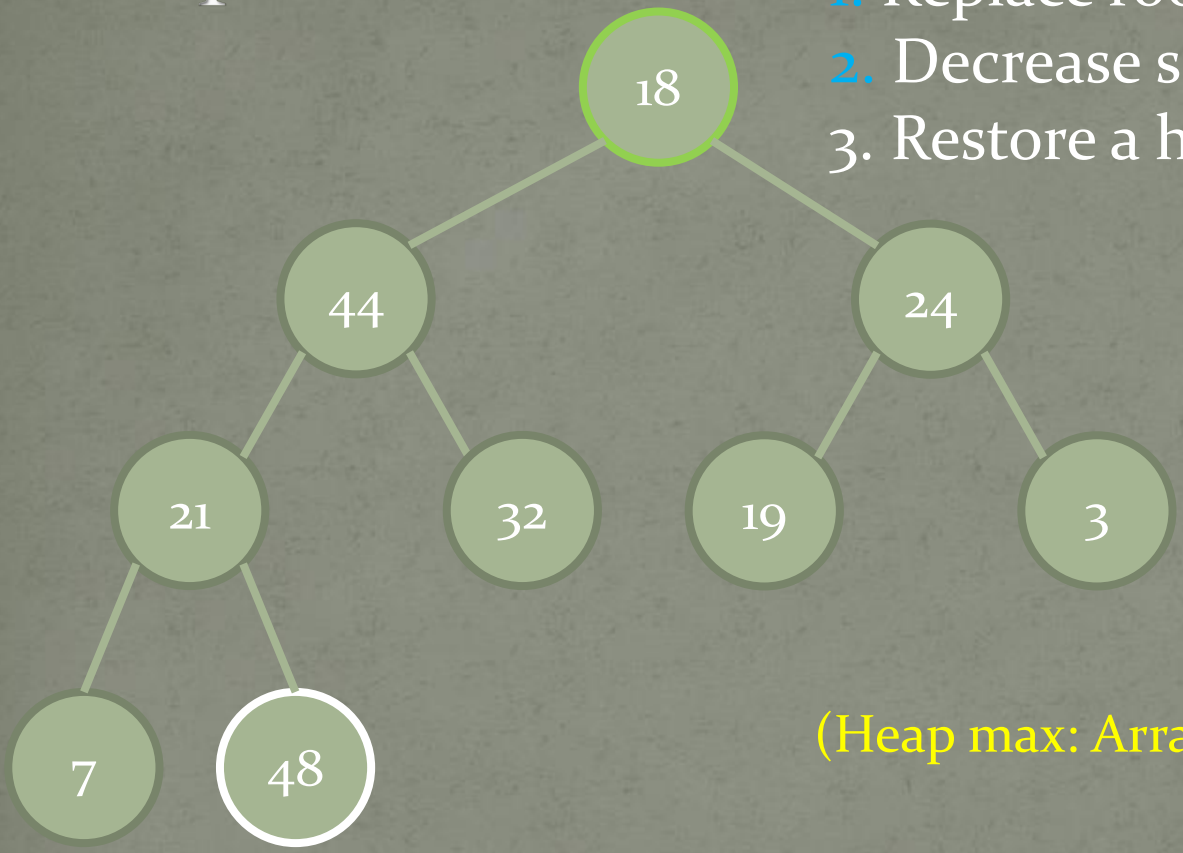
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	48	44	24	21	32	19	3	7	18		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;



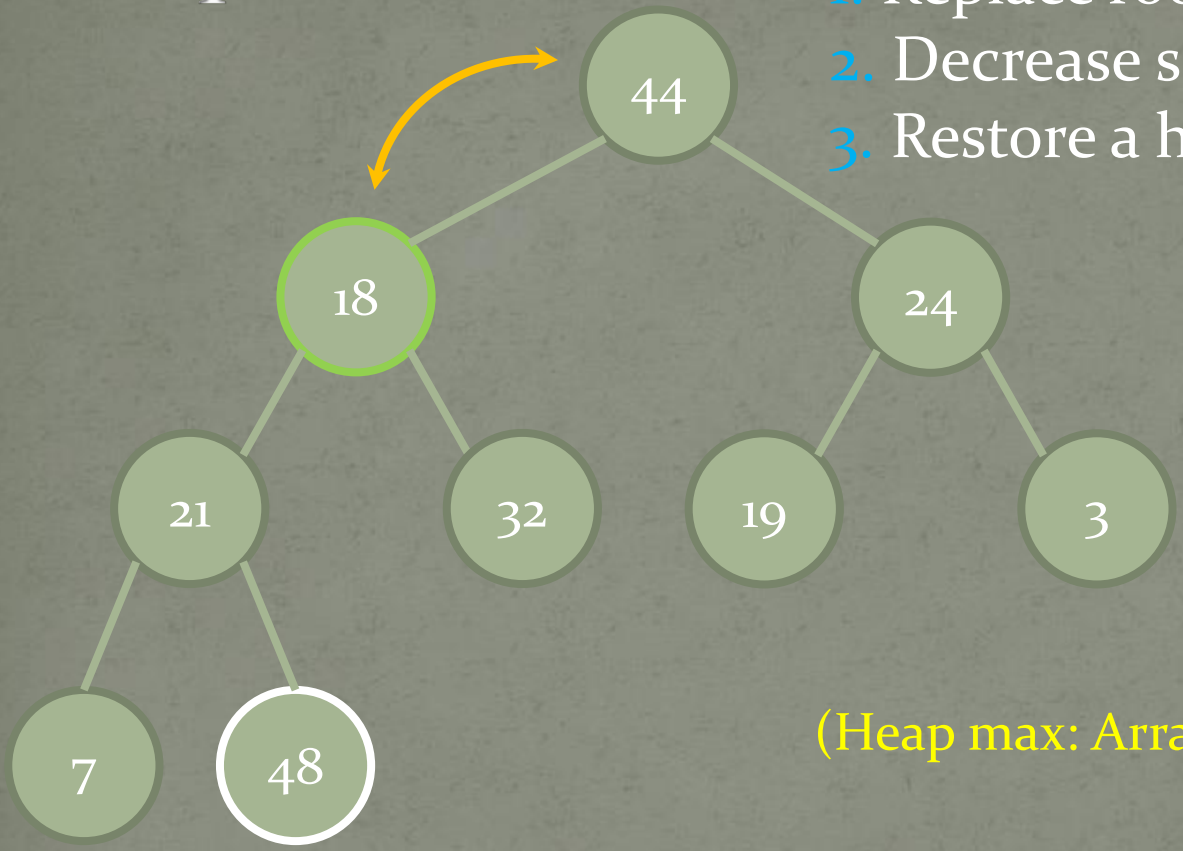
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	18	44	24	21	32	19	3	7	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;



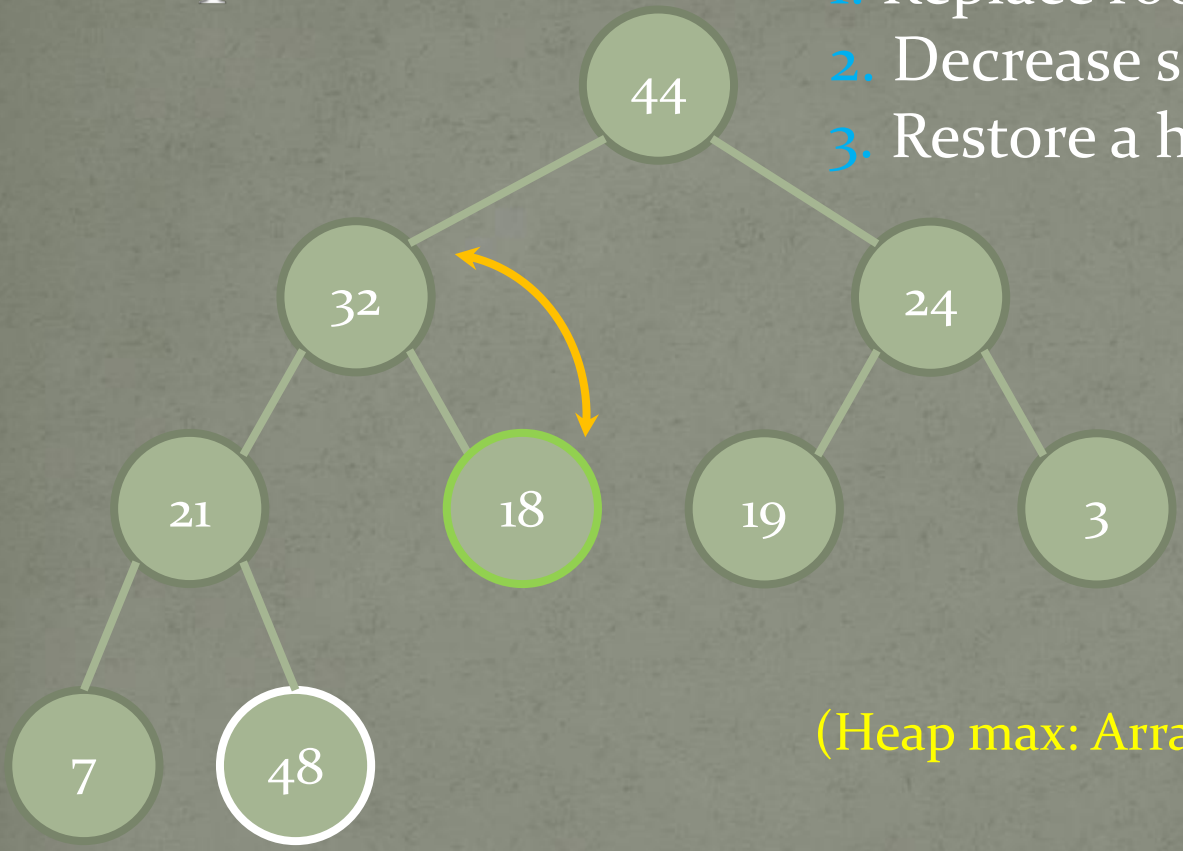
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	44	18	24	21	32	19	3	7	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;



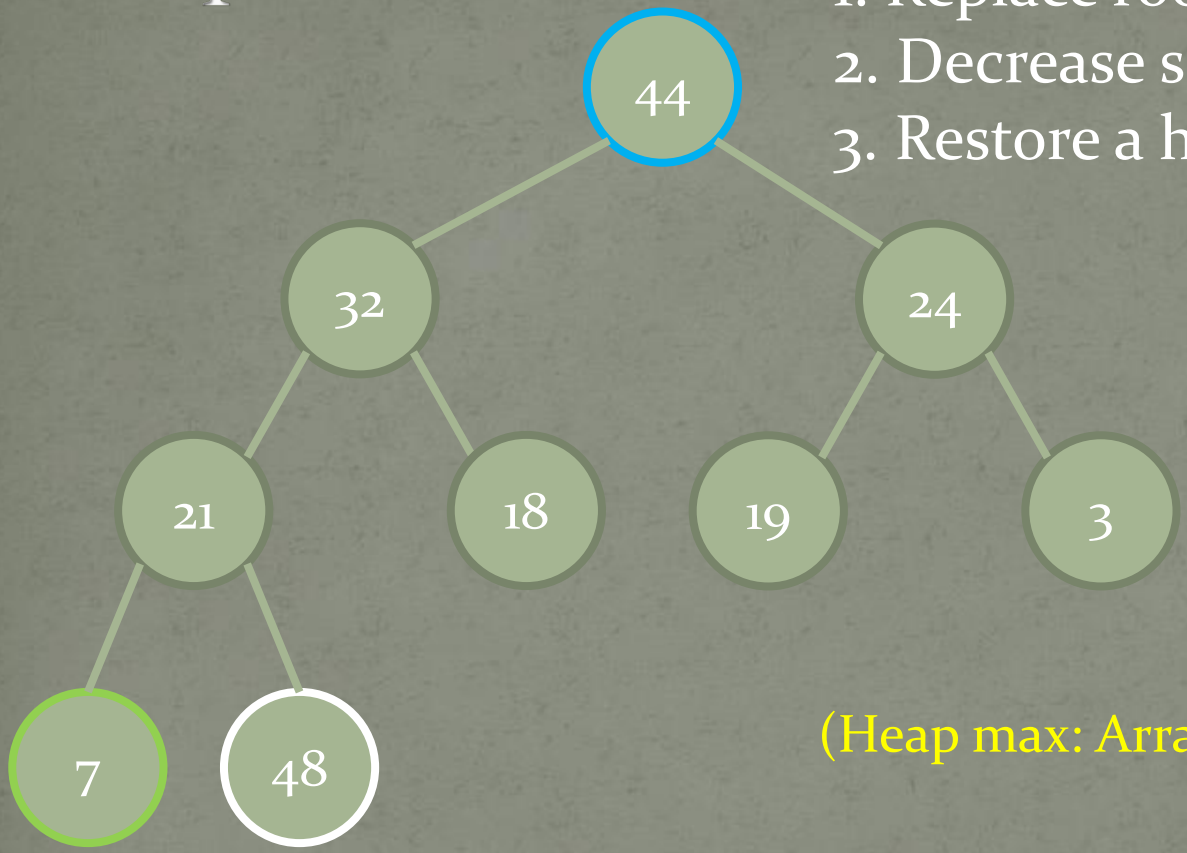
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	44	32	24	21	18	19	3	7	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;

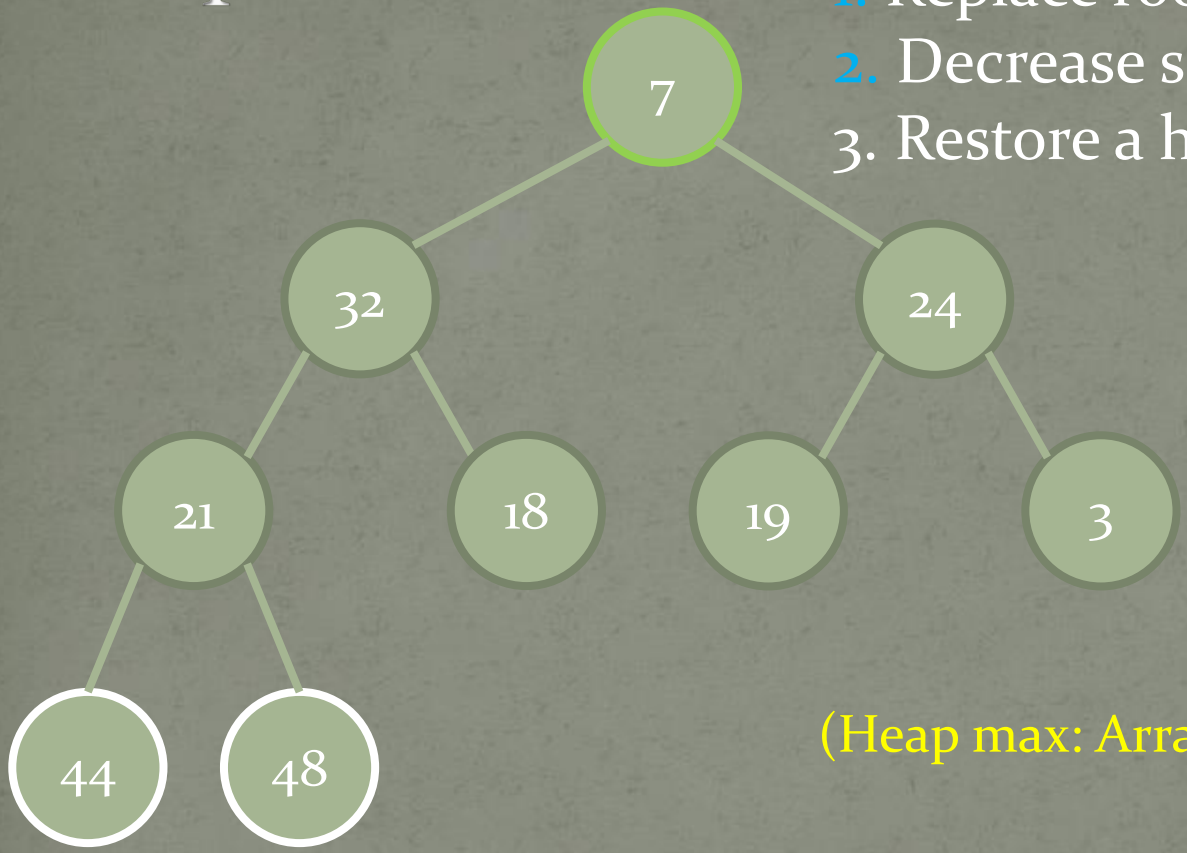


Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	44	32	24	21	18	19	3	7	48		

Heapsort



1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;

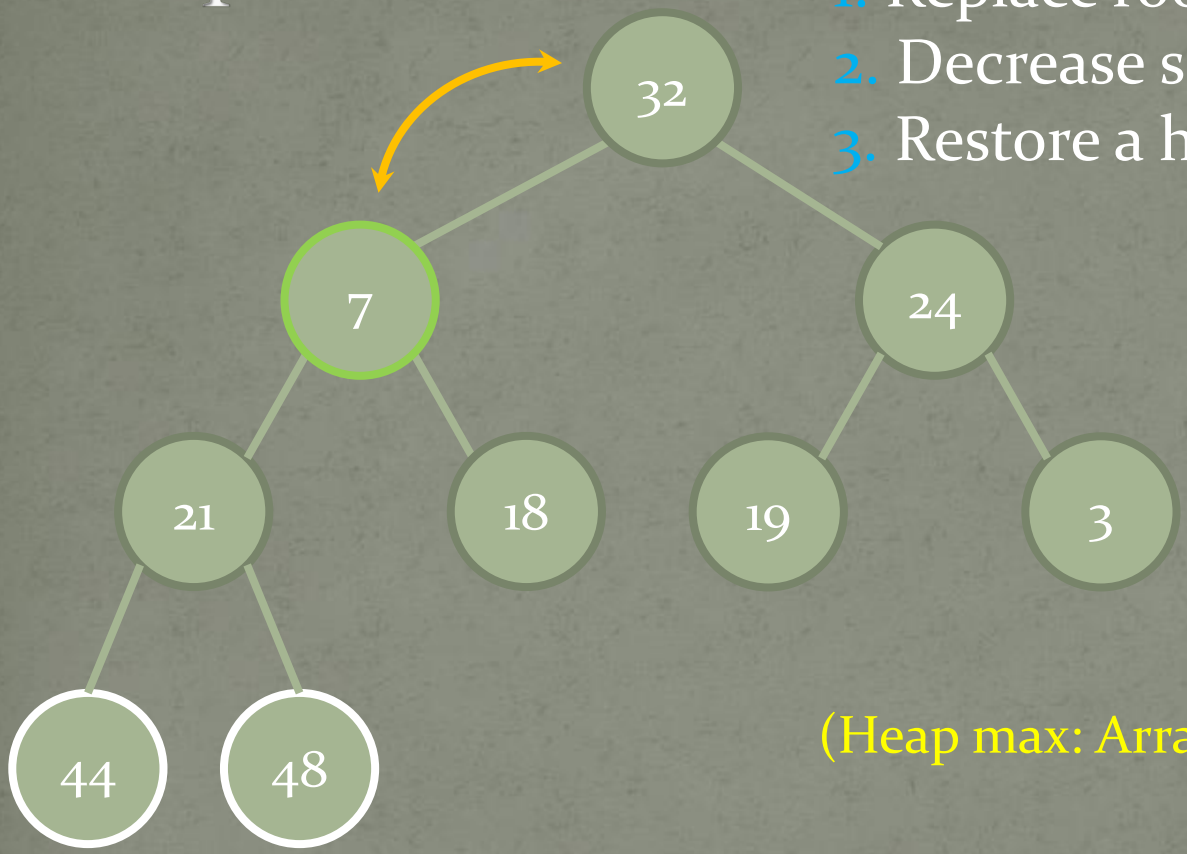
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	7	32	24	21	18	19	3	44	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;



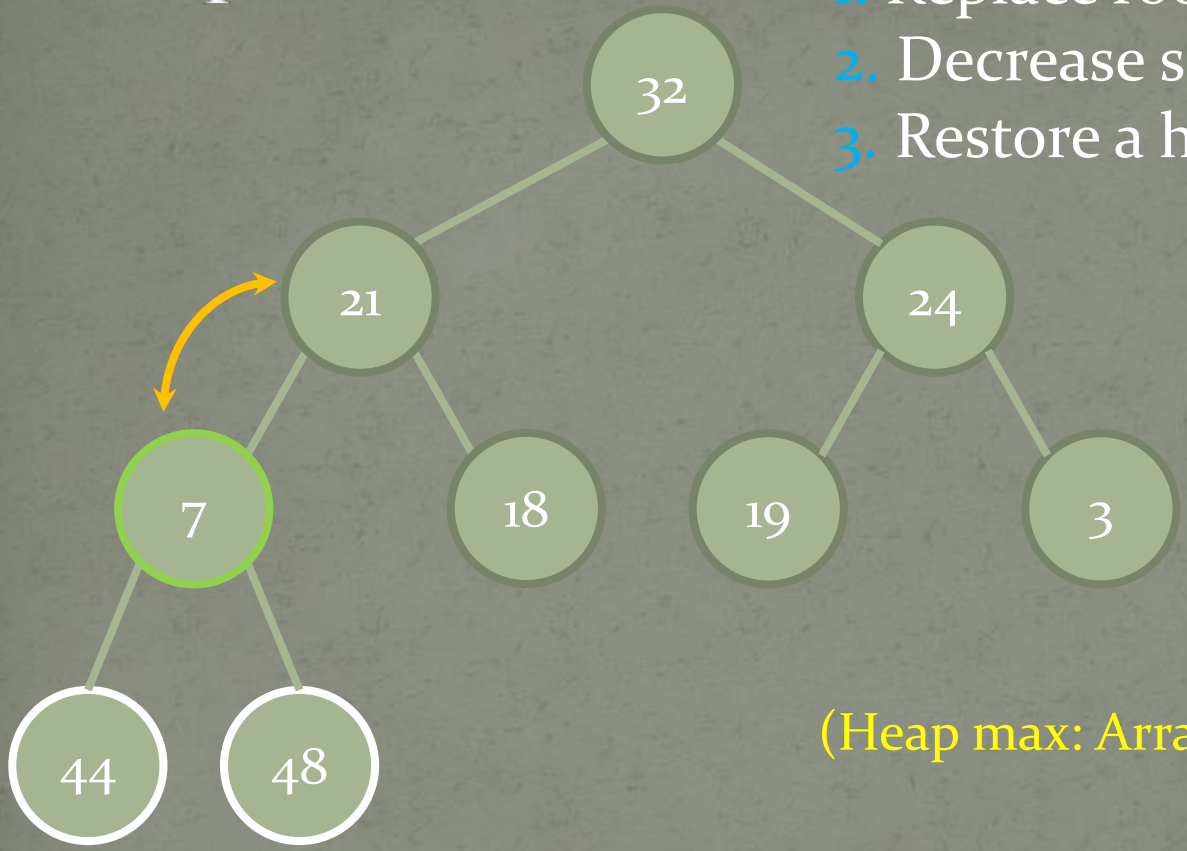
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	32	7	24	21	18	19	3	44	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;

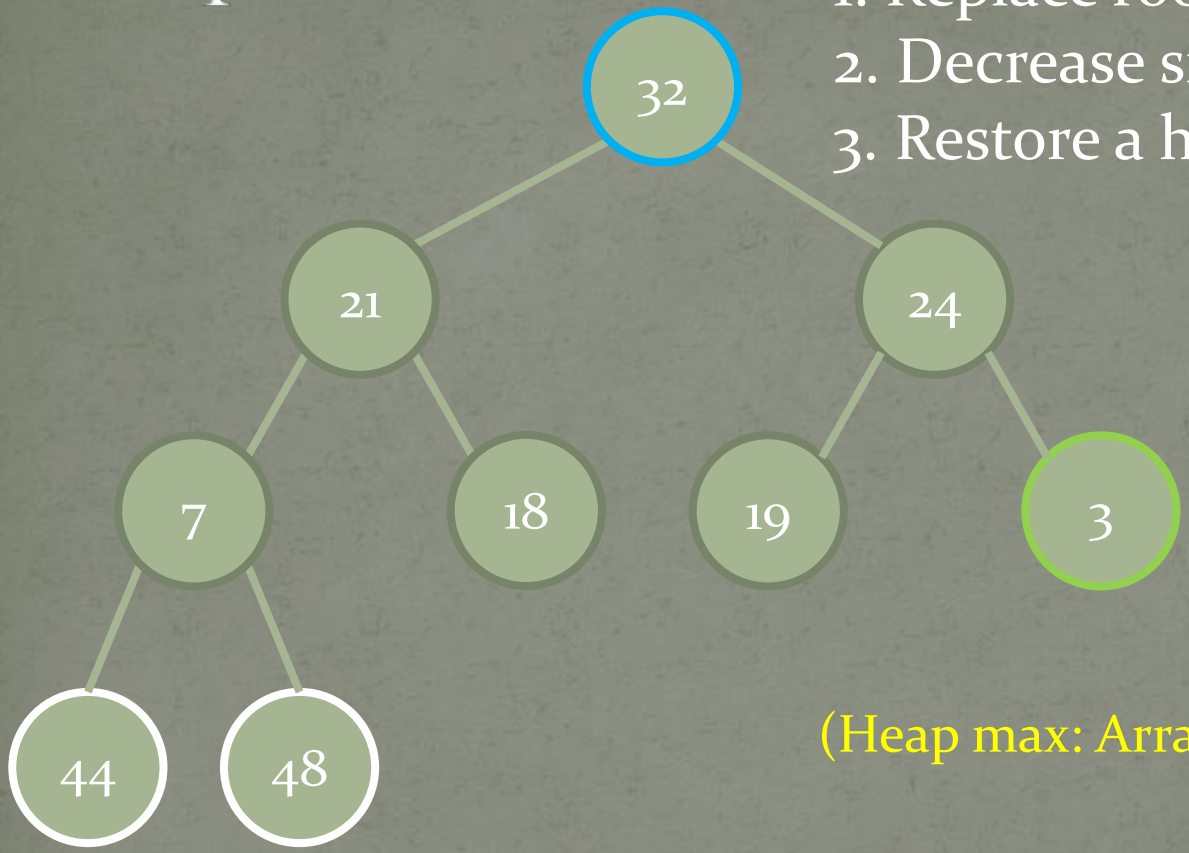


Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	32	21	24	7	18	19	3	44	48		

Heapsort



1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;

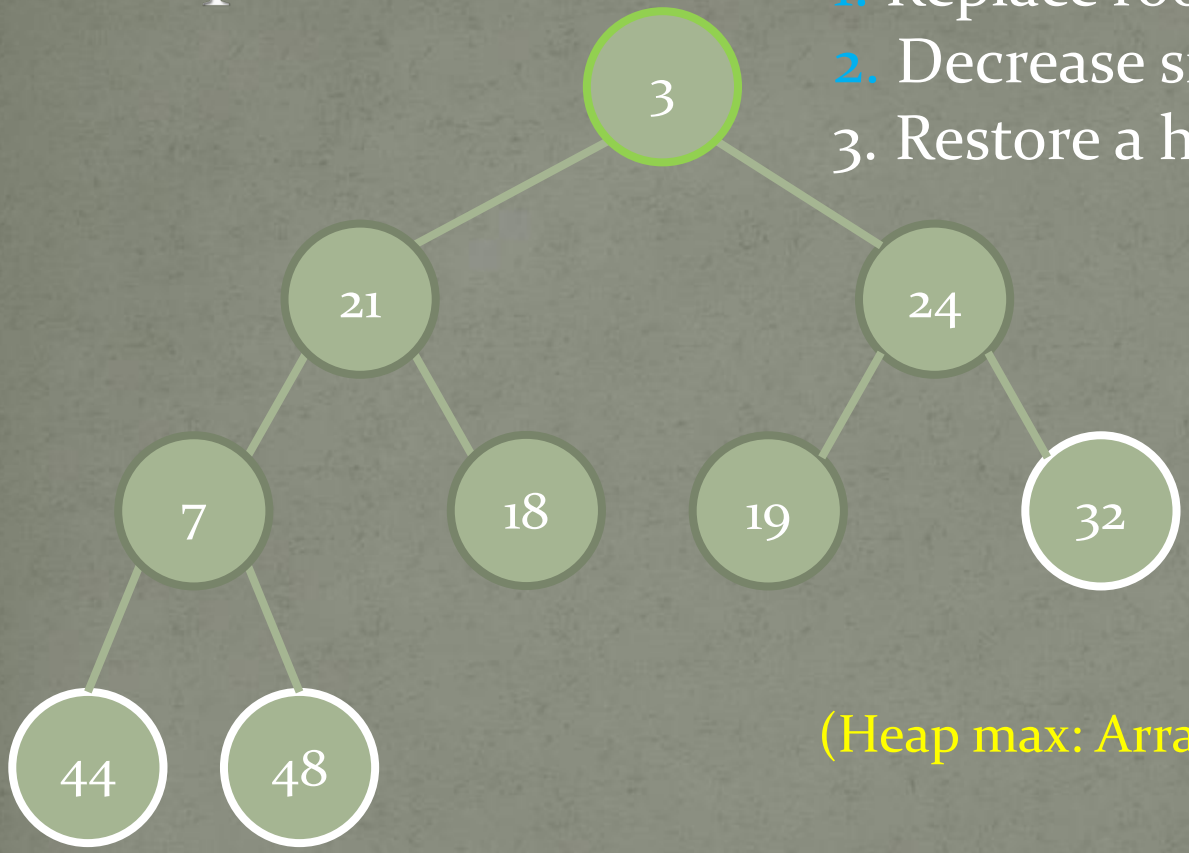
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	32	21	24	7	18	19	3	44	48		

Heapsort

1. Replace root with the last item,
2. Decrease size of heap,
3. Restore a heap structure;



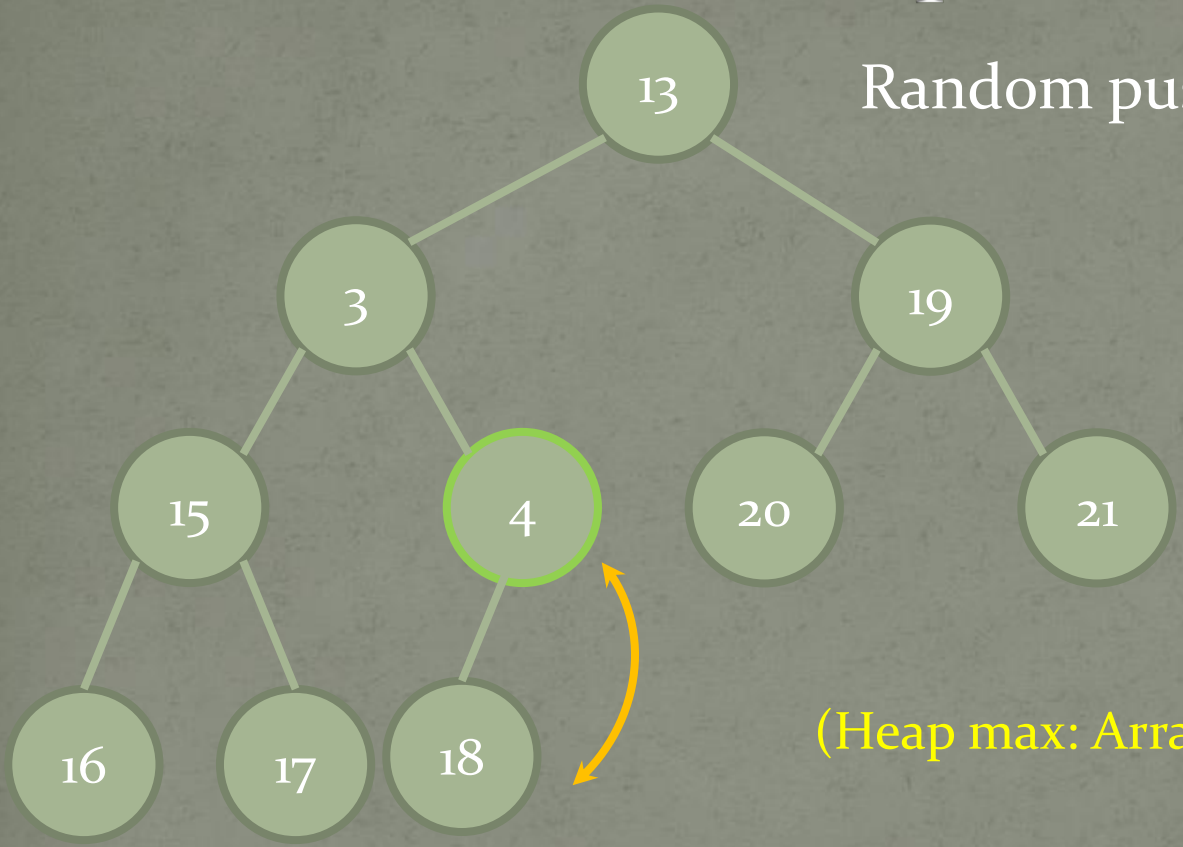
Continue as long as size of the heap is not less than 2

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	3	21	24	7	18	19	32	44	48		

Function: makeHeap()

Function: makeHeap()



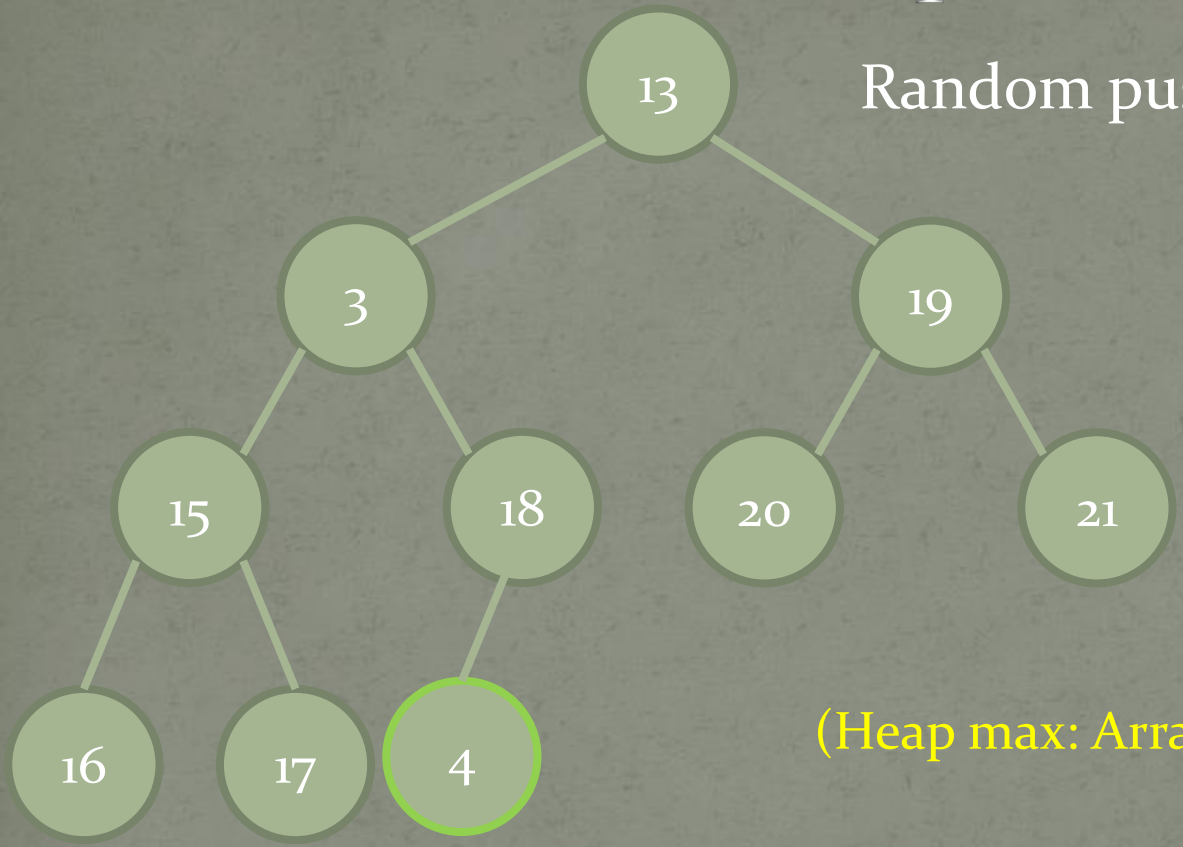
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	19	15	4	20	21	16	17	18	

Function: makeHeap()



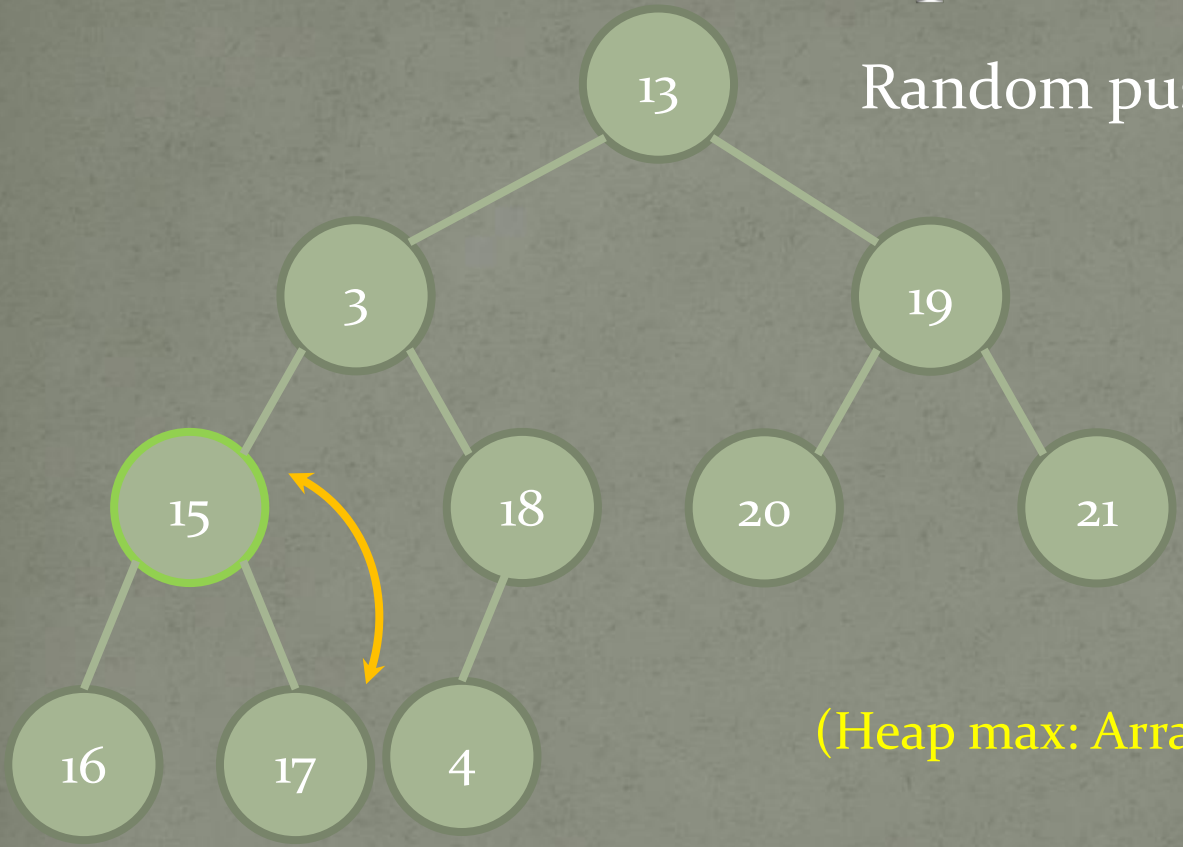
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	19	15	18	20	21	16	17	4	

Function: makeHeap()



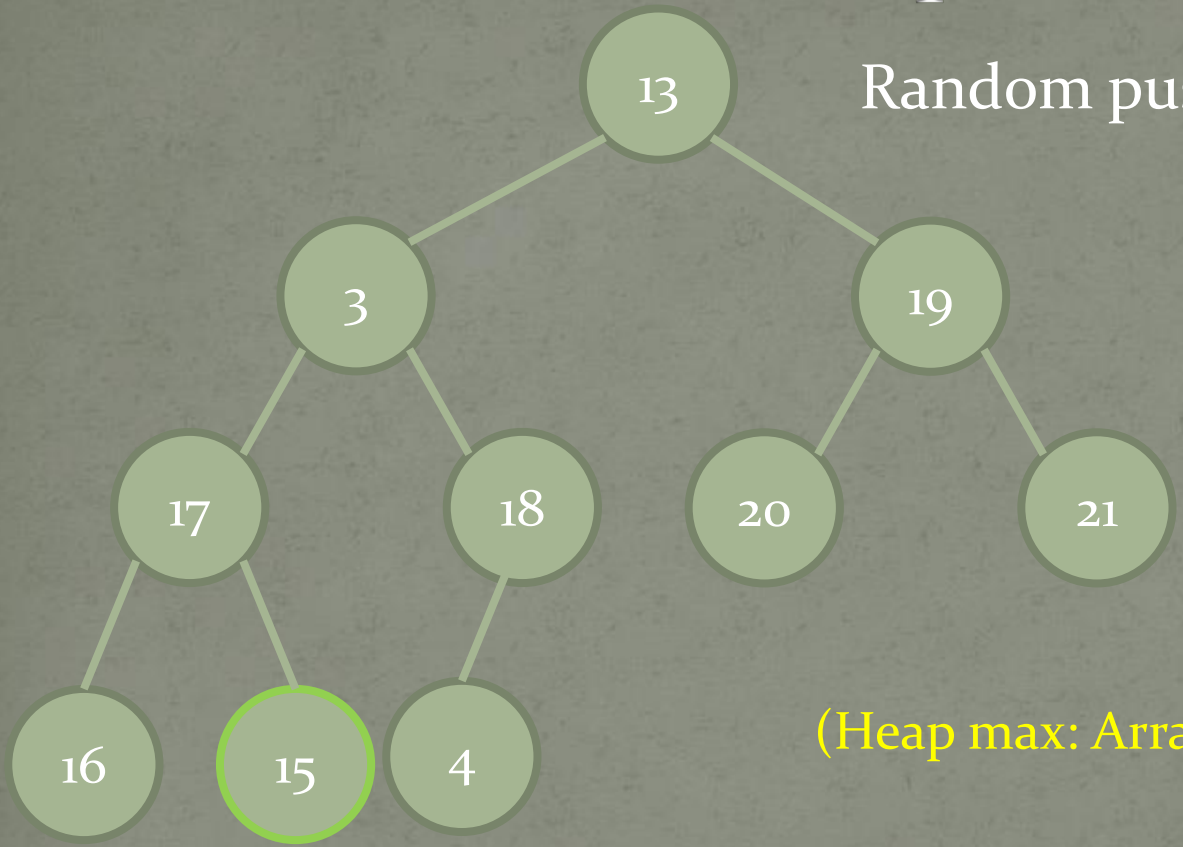
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	14	19	15	18	20	21	16	17	4	

Function: makeHeap()



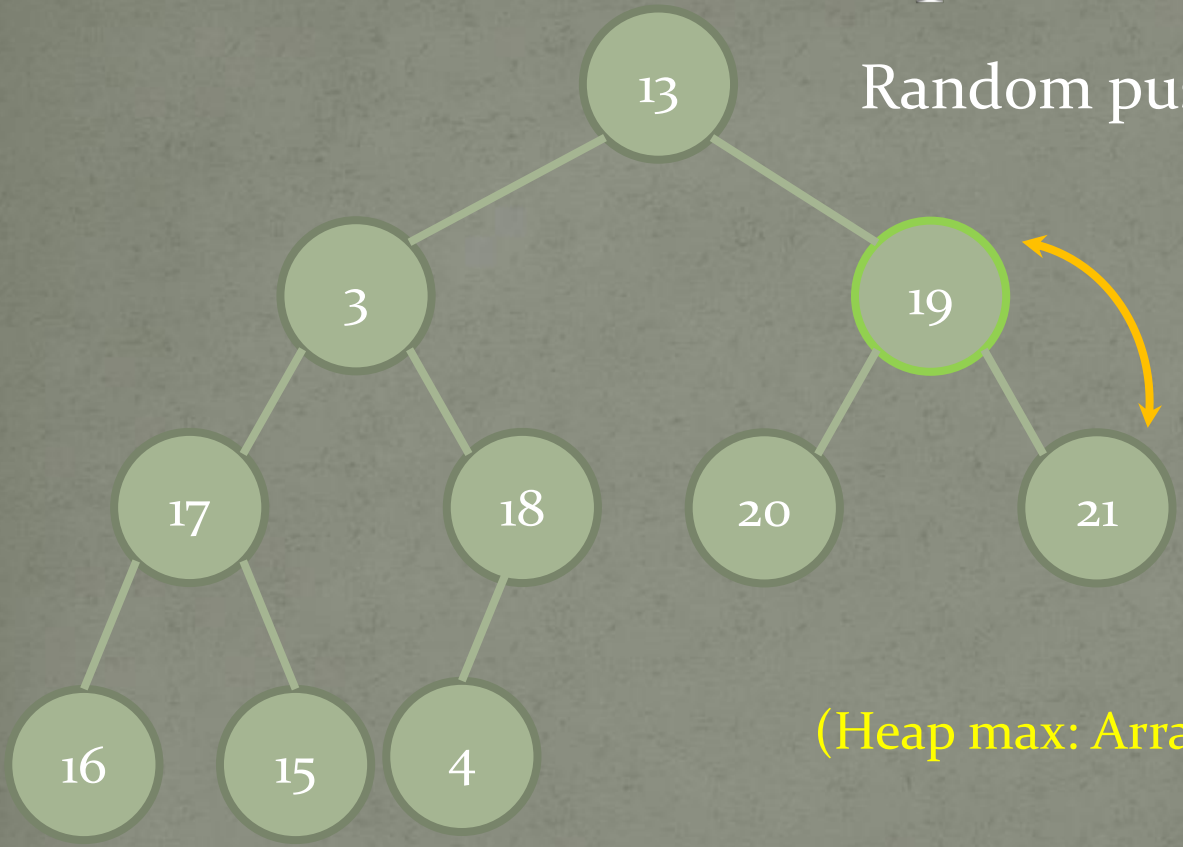
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	19	17	18	20	21	16	15	4	

Function: makeHeap()



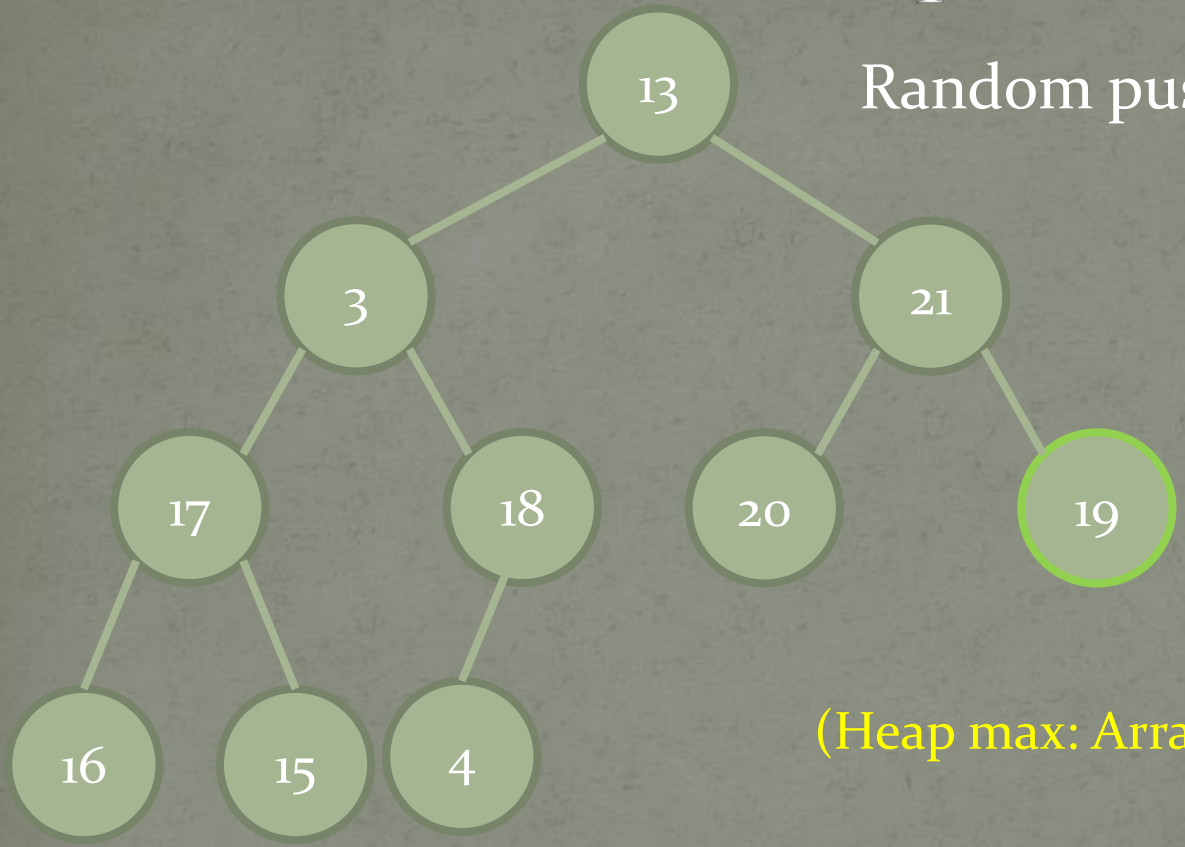
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	19	17	18	20	21	16	15	4	

Function: makeHeap()



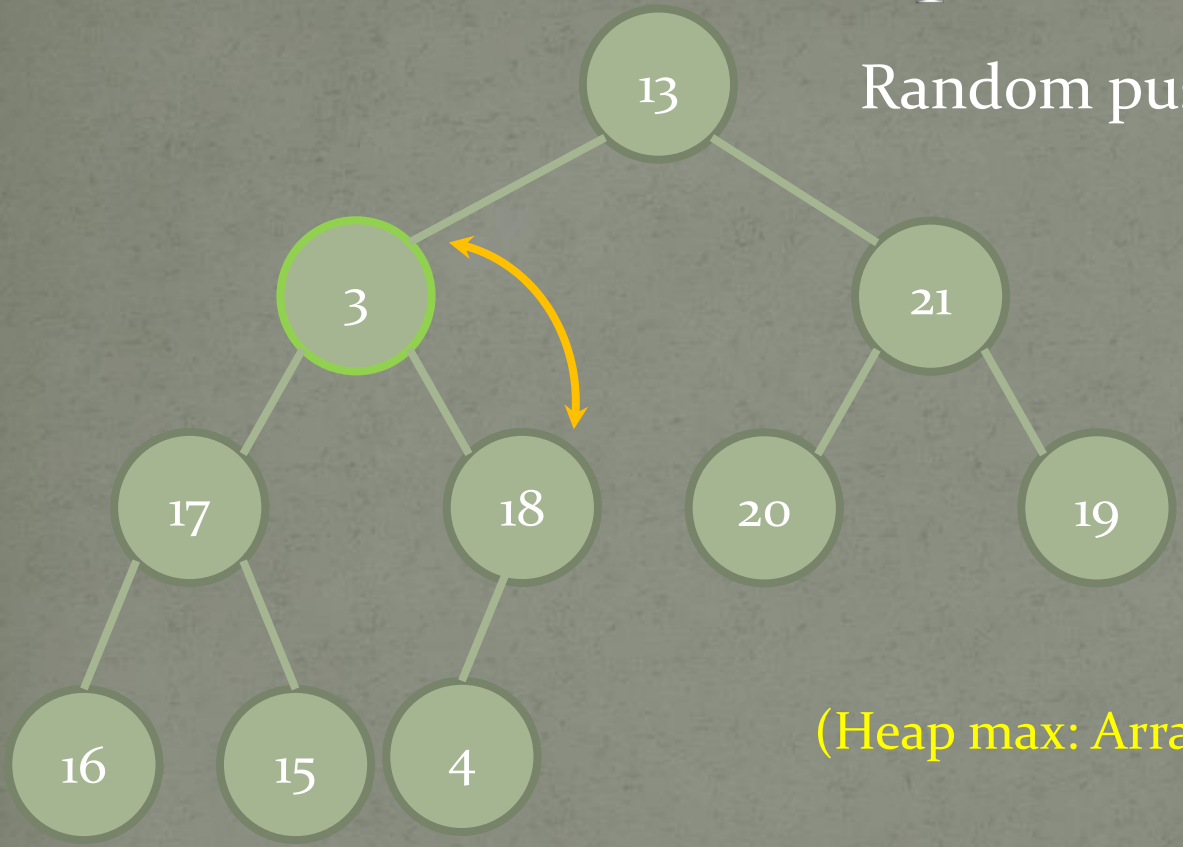
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	21	17	18	20	19	16	15	4	

Function: makeHeap()



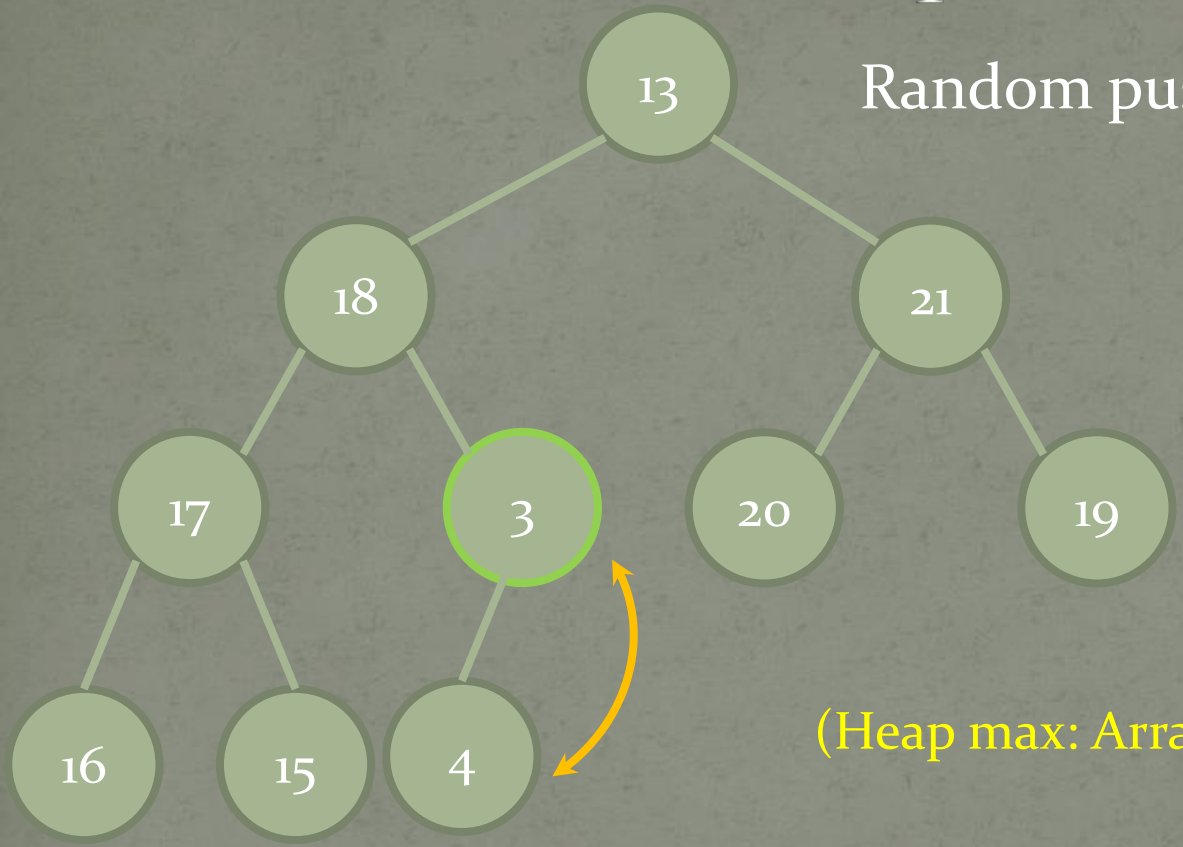
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	3	21	17	18	20	19	16	15	4	

Function: makeHeap()



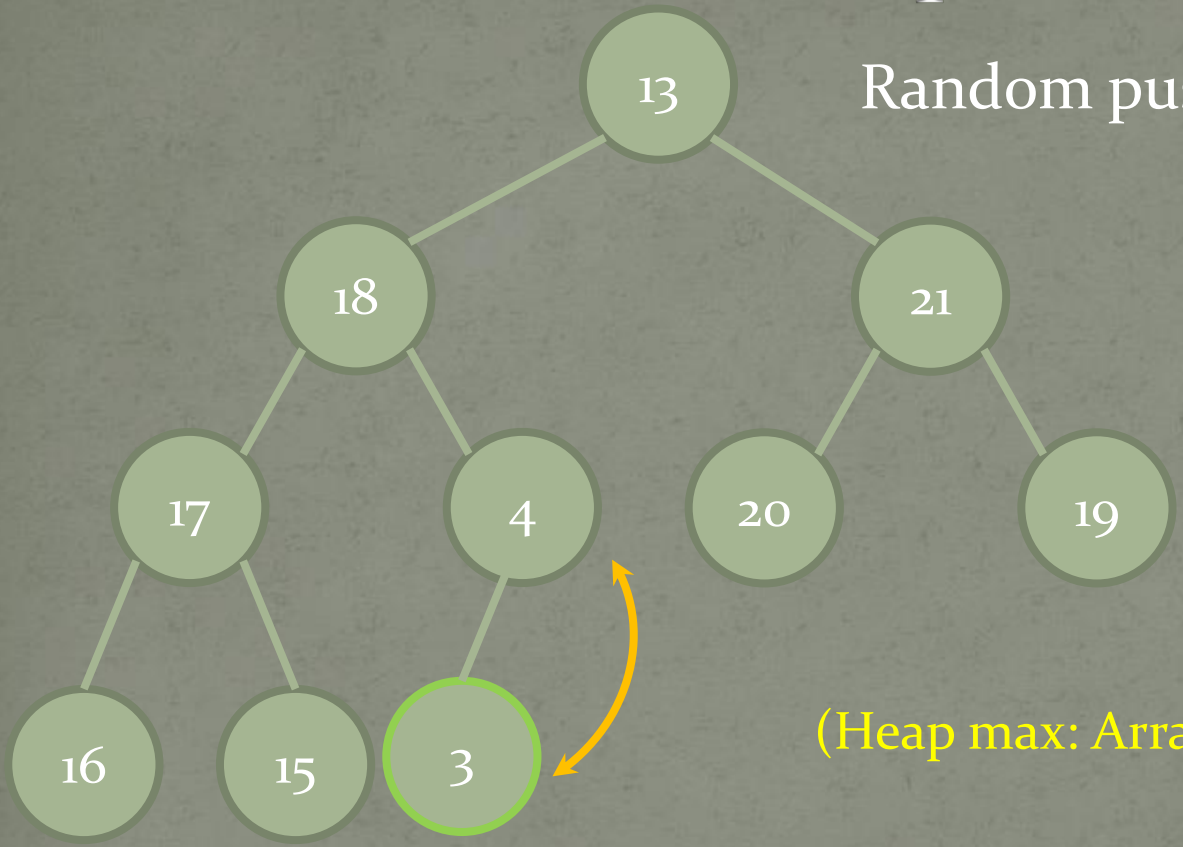
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	18	21	17	3	20	19	16	15	4	

Function: makeHeap()



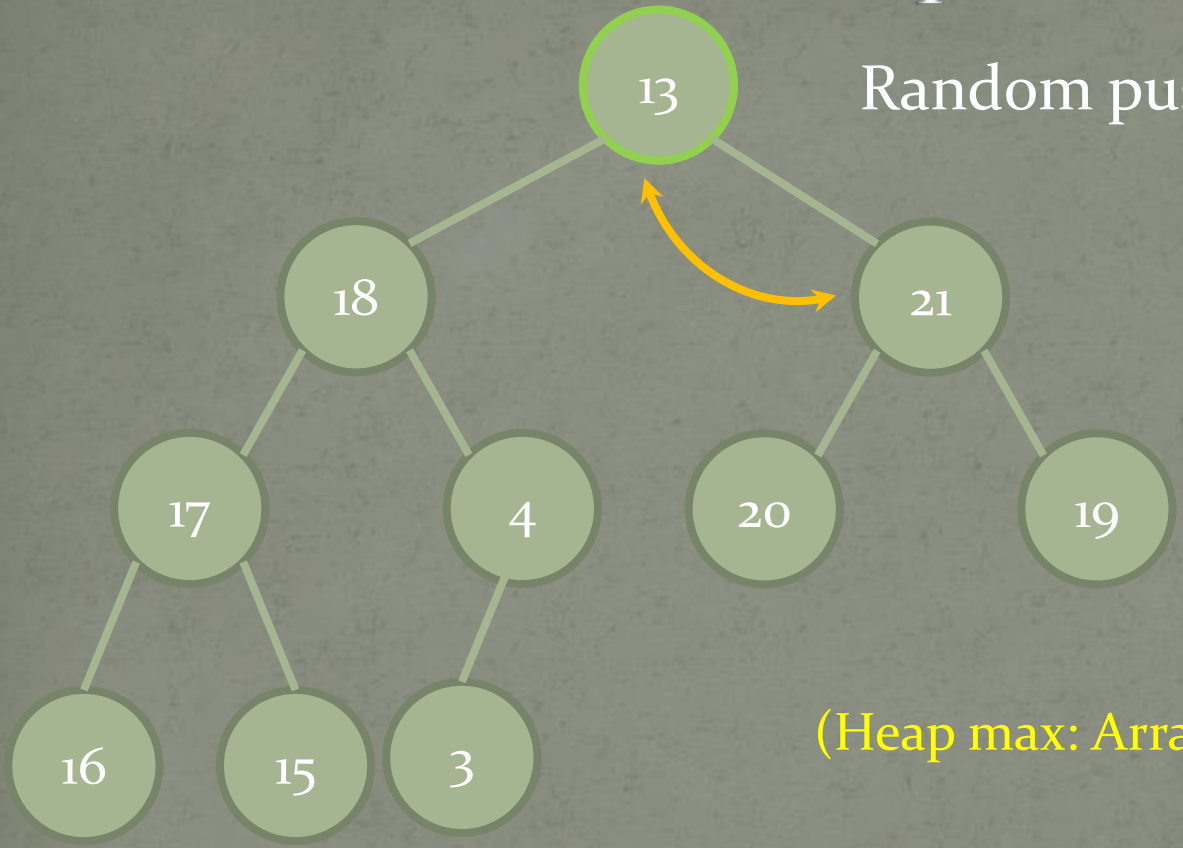
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	18	21	17	4	20	19	16	15	3	

Function: makeHeap()



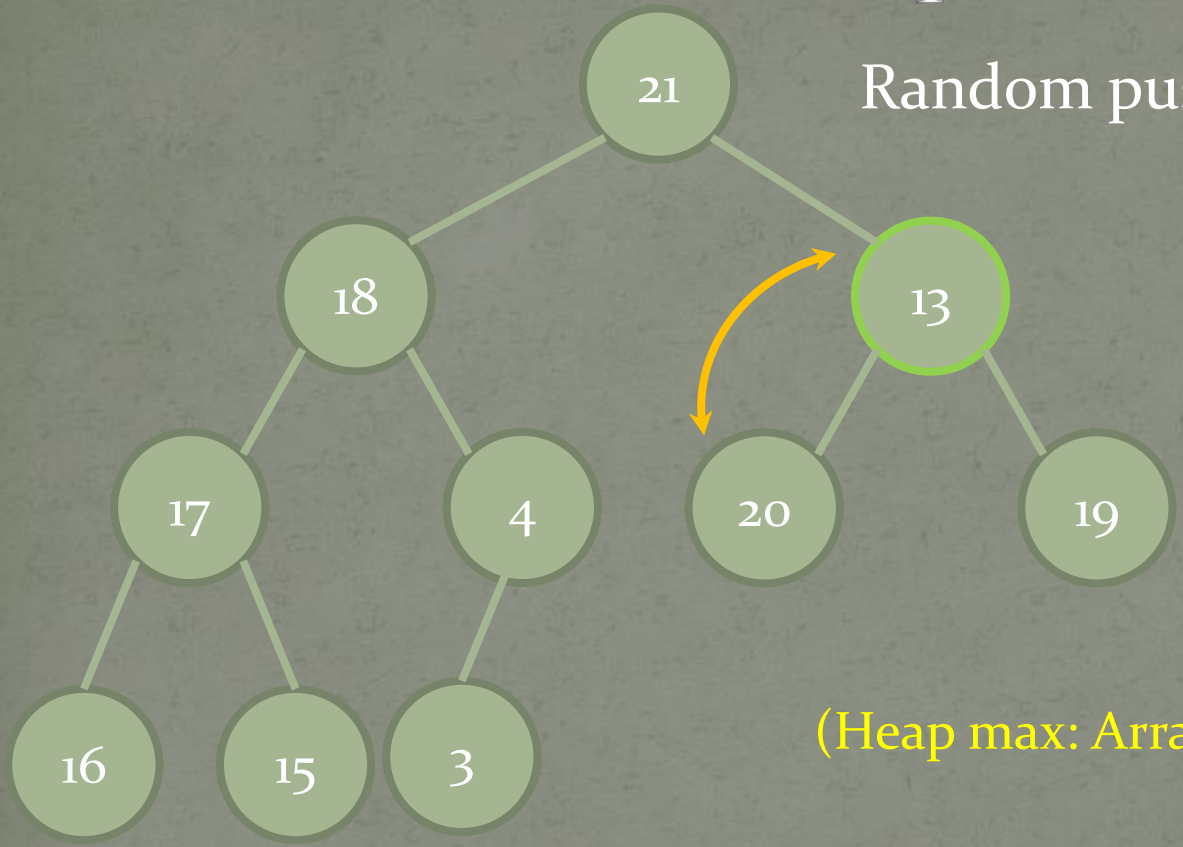
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	13	18	21	17	4	20	19	16	15	3	

Function: makeHeap()



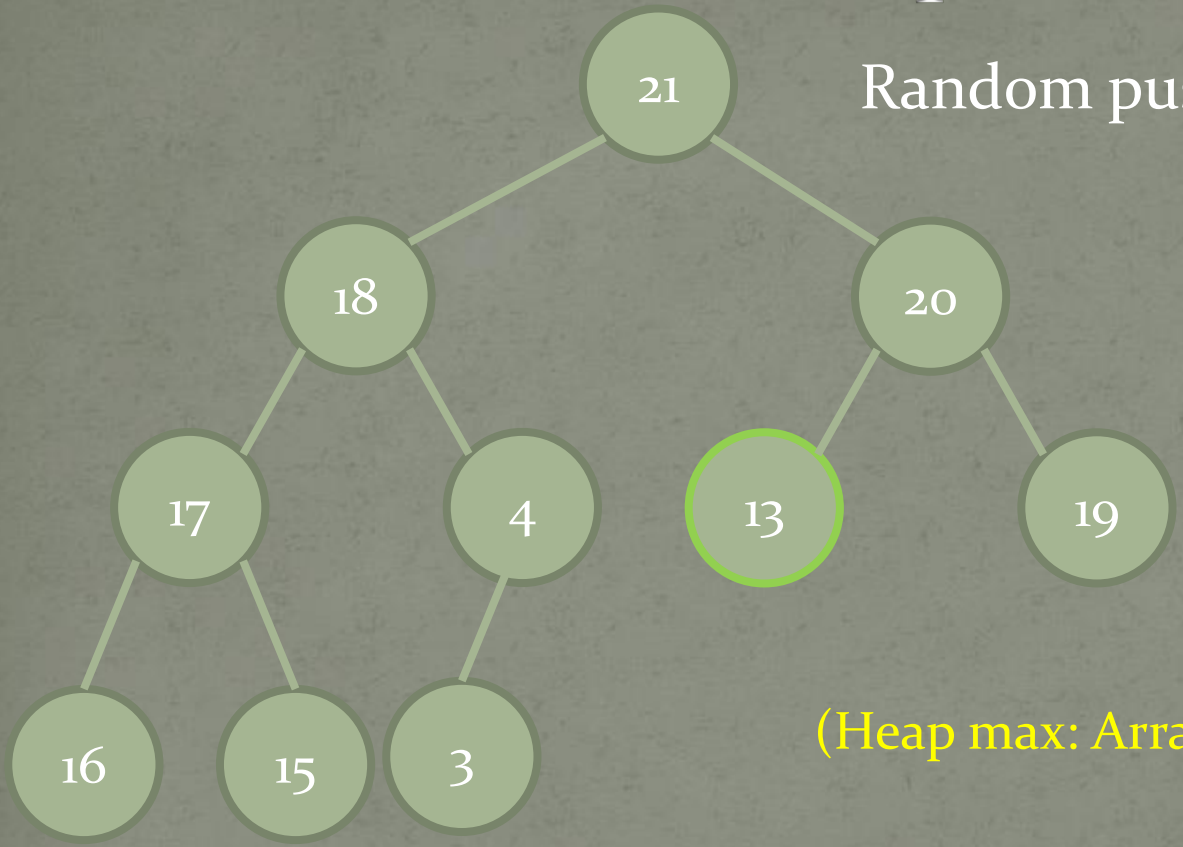
Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	21	18	13	17	4	20	19	16	15	3	

Function: makeHeap()



Random pushing of new items

Father = floor(i/2)
Left child = 2*i
Right child = 2*i+1

(Heap max: Array[i]<=Array[Father[i]])

0	1	2	3	4	5	6	7	8	9	10	11
	21	18	20	17	4	13	19	16	15	3	