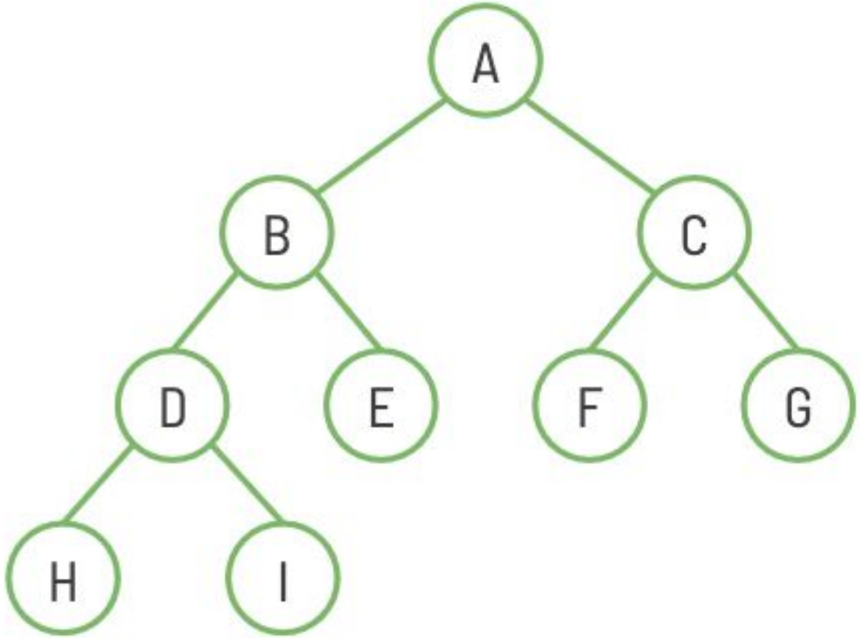
PSO 4

One minute Midterm vent session

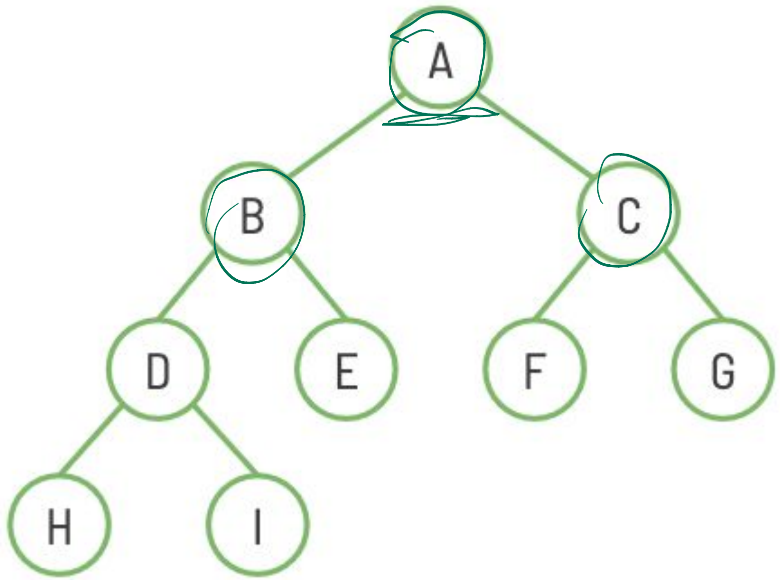
Binary Heaps

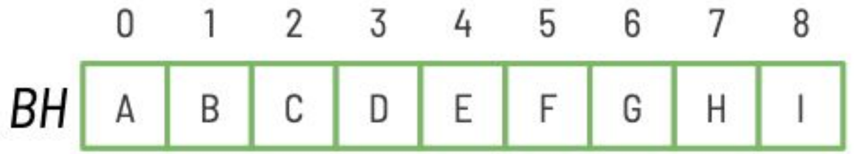


Max-heap (aka **Max Priority Queue**) if the key in each node is **larger than** or equal to the keys in that node's two children (if any).

Min-heap (aka **Min Priority Queue)** if the key in each node is **less than** or equal to the keys in that node's two children (if any).

Binary Heaps as Arrays





	0	1	2	3	4	5	6	7	8
BH	A	B	C	D	E	F	G	H	I

$$\text{leftchild}(i \in \mathbb{Z}_{\geq 0}) := 2i + 1$$

$$\text{rightchild}(i \in \mathbb{Z}_{\geq 0}) := 2i + 2$$

$$\text{parent}(i \in \mathbb{Z}^{+}) := \left\lfloor \frac{i-1}{2} \right\rfloor$$

Question 3

(Binary heap)

(1) If the binary heap is represented as an array, and the root is stored at index 0, where is the left child of the node at index $i = 23$ stored?

A. 45

B. 46

C. 47

D. 48

E. 49

General formula for this?

Question 3

(Binary heap)

(1) If the binary heap is represented as an array, and the root is stored at index 0, where is the left child of the node at index $i = 23$ stored?

A. 45

B. 46

[✓ C. 47]

D. 48

E. 49

<u>Binary Heap Cheatsheet</u>

left(i) = 2i + 1

right(i) = 2i + 2

(2) If the binary heap is represented as an array, and the root is stored at index 0, where is the parent of the node at index $i = 99$ stored?

A. 45

B. 46

C. 47

D. 48

[✓ E.] 49

Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

(2) If the binary heap is represented as an array, and the root is stored at index 0, where is the parent of the node at index $i = 99$ stored?

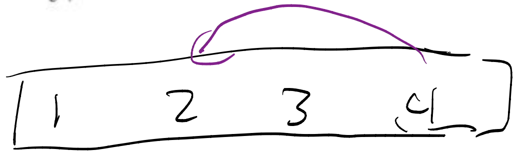
A. 45

B. 46

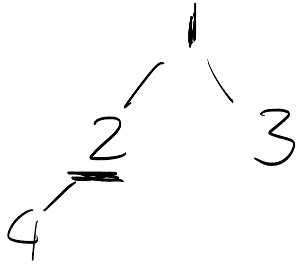
C. 47

D. 48

E. 49



parent(3) = 1



Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

parent(i) = ⌊(i - 1) / 2⌋

(3) If the binary heap is represented as an array of length $n = 99$, and the root is stored at index 0, where is the last non-leaf node stored?

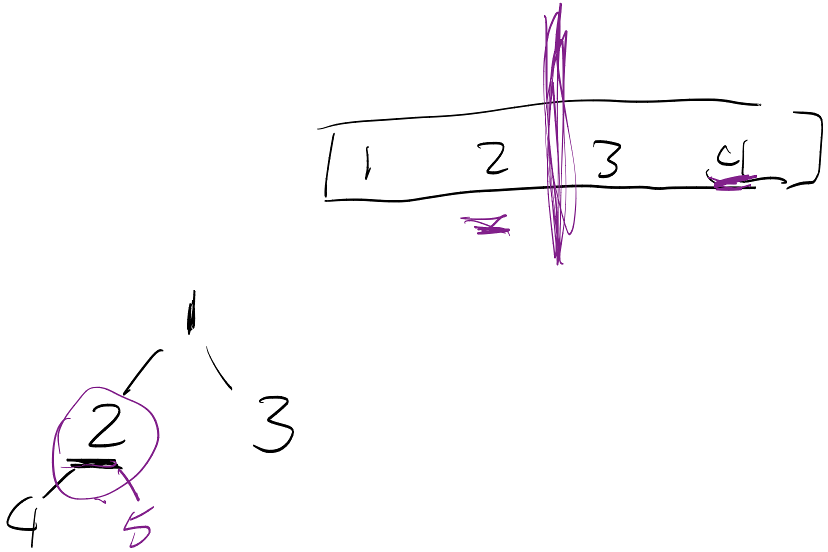
A. 45

B. 46

C. 47

[✓ D.] 48

E. 49



Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

parent(i) = ⌊(i - 1) / 2⌋

4649

(3) If the binary heap is represented as an array of length $n = 99$, and the root is stored at index 0, where is the last non-leaf node stored?

A. 45

B. 46

C. 47

[✓ D. 48]

E. 49

Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

parent(i) = $\lfloor(\overset{n}{i} - 1) / 2\rfloor$

lastNonLeaf() = ⌊(n / 2) - 1⌋

General intuition: There are ~n/2 leaves since these are **complete trees**

(4) If the binary heap is represented as an array of length $n = 99$, and you want to insert an element, how many different locations of the element are possible after insertion?

A. 5

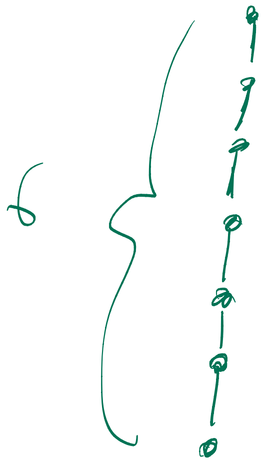
B. 6

[✓ C.] 7

D. 8

E. 9

hint: $\lfloor \log_2 99 \rfloor = 6$



Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

parent(i) = ⌊(i - 1) / 2⌋

lastNonLeaf() = ⌊(n / 2) - 1⌋

(4) If the binary heap is represented as an array of length $n = 99$, and you want to insert an element, how many different locations of the element are possible after insertion?

height() + 1

A. 5

B. 6

[✓ C. 7]

D. 8

E. 9

Binary Heap Cheatsheet

left(i) = 2i + 1

right(i) = 2i + 2

parent(i) = ⌊(i - 1) / 2⌋

lastNonLeaf() = ⌊(n / 2) - 1⌋

height() = ⌊lg n⌋

Question 2

(Heap sort) In the following questions, we consider Heap sort using **Heapify**.

(1) Show the array $\{3, 4, 1, 0, 9, 2\}$ as it goes through Heap sort (in the ascending order).

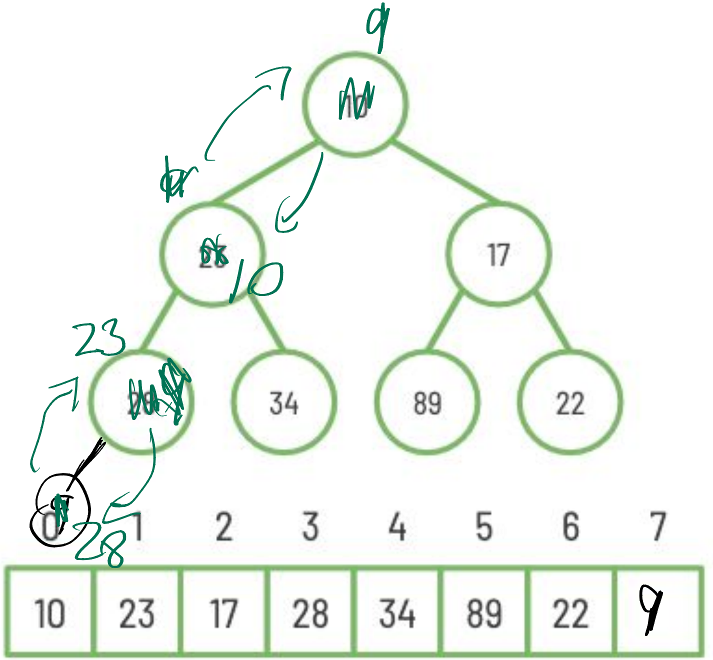
(2) Given K number of sorted (ascending ordered) arrays each having N/K elements in it, your task is to merge all these arrays to form a N-element final sorted array (also in the ascending order).

(2.1) Propose a simple solution to the problem which may run in $O(N \log(N))$ time.

(2.2) Can you propose a better algorithm to solve the problem? What is the time complexity of your proposed solution?

Heap Insertion

1. Insert at next leaf
2. Sift up



0	1	2	3	4	5	6	7
10	23	17	28	34	89	22	9

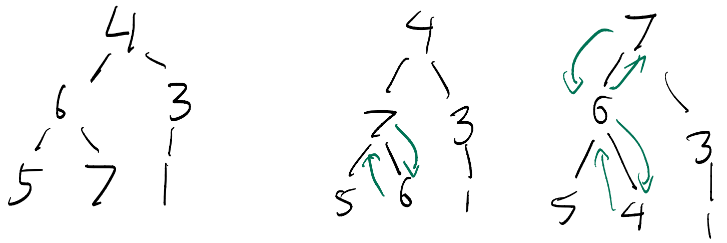
Demonstration: insert(9)

(Max) Heapify: Turning your arrays into Heaps

For each non-leaf node from the last to the first:

while it is less than its largest child, swap it downward

[4 6 3 5 7 1]



Demonstr. : Heapify [4 6 3 5 7 1]

[4 6 3 5 7 1]

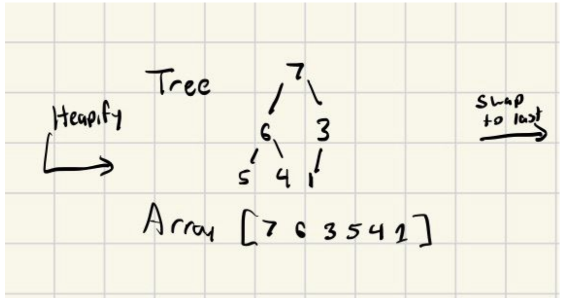
Heap Sort

AS SEEN ON

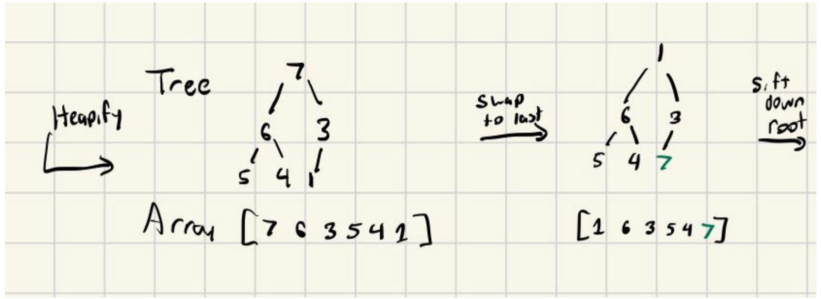
Andres’s Lecture

Idea: In a max heap, the max element is always at the root, sort backwards, from largest to smallest

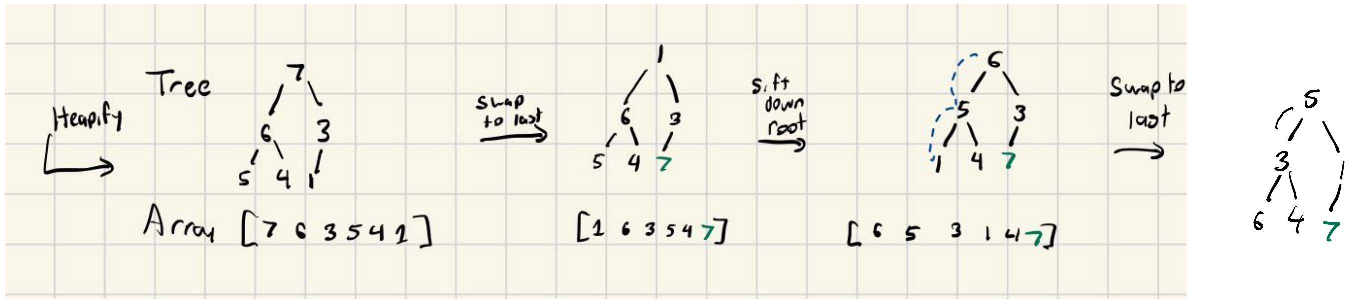
1. Heapify your array
2. Swap root with last leaf, excluding the elements you’ve already swapped
3. Fix heap by sifting down, excluding the elements you’ve already swapped
4. Repeat steps 2-4



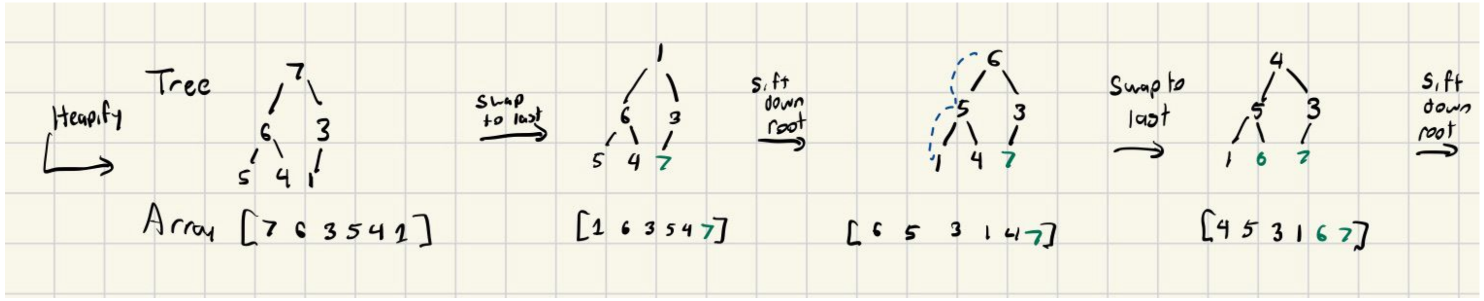
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



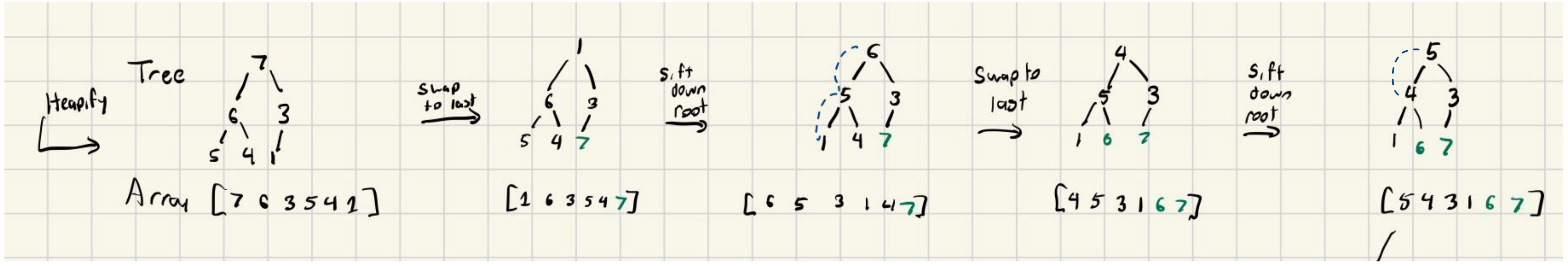
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



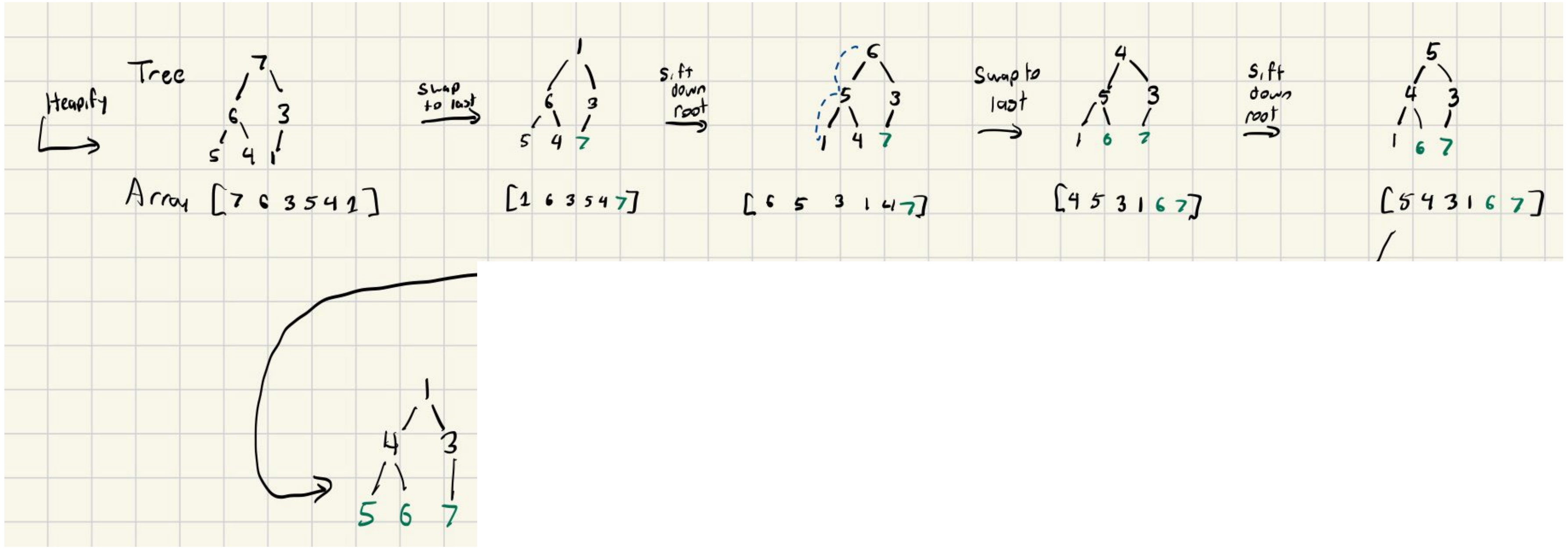
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



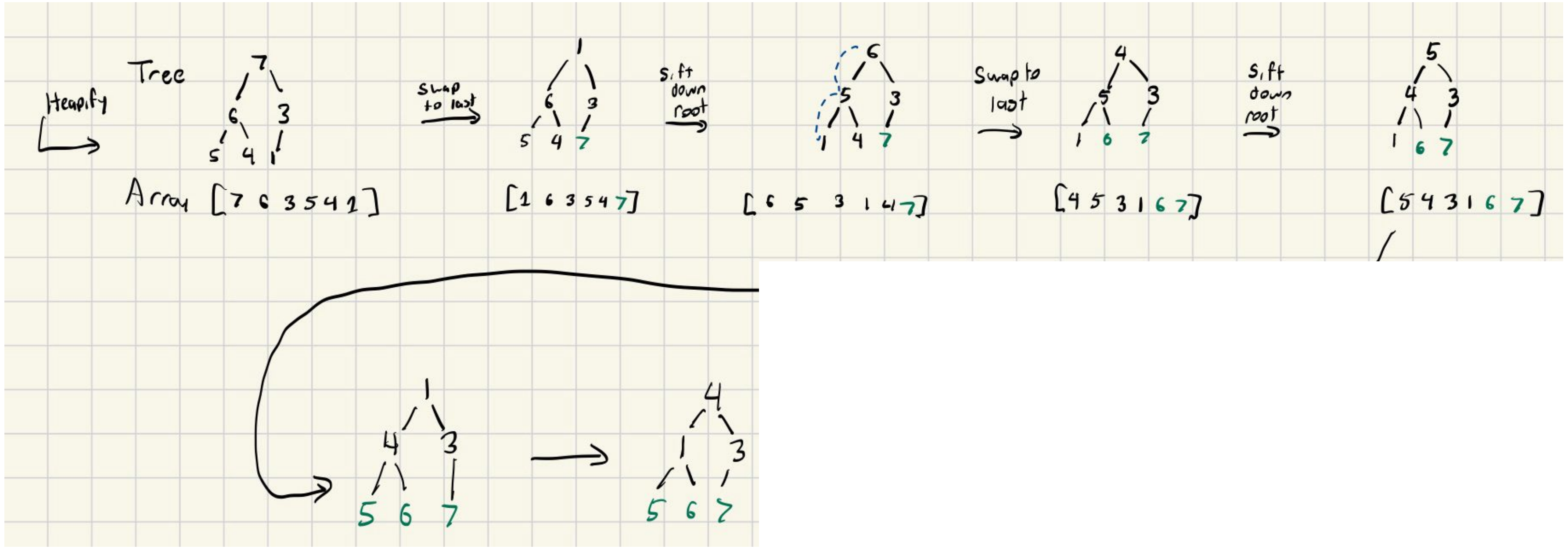
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



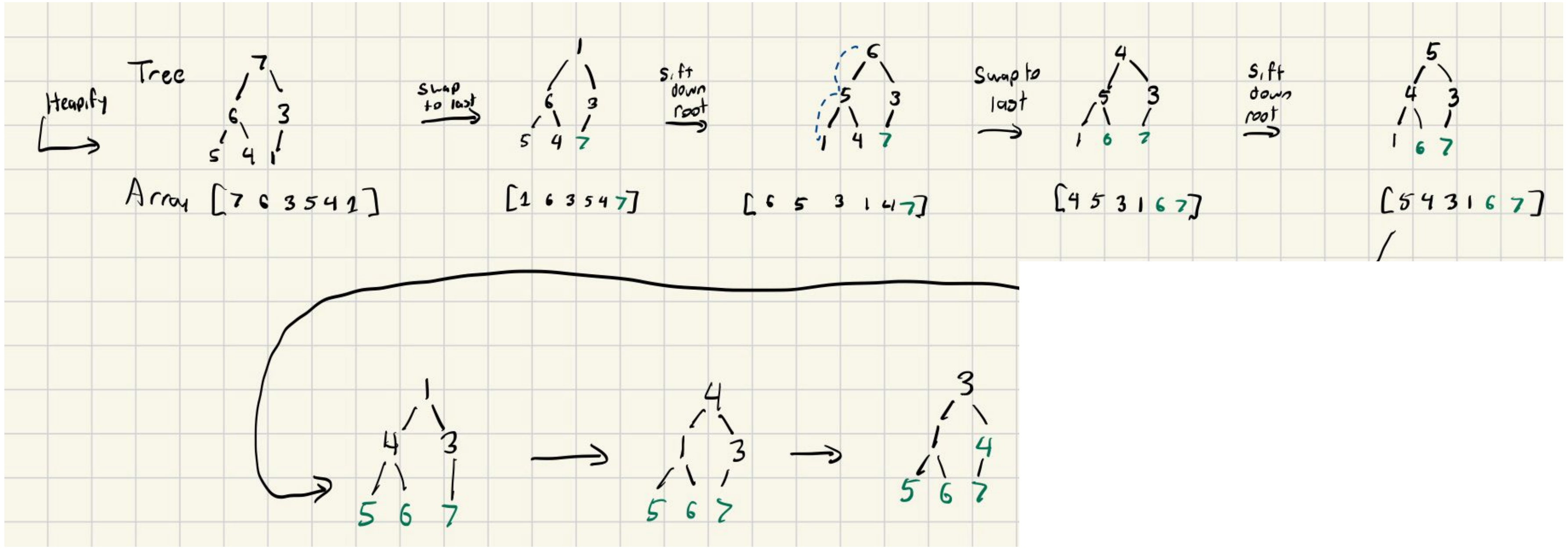
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



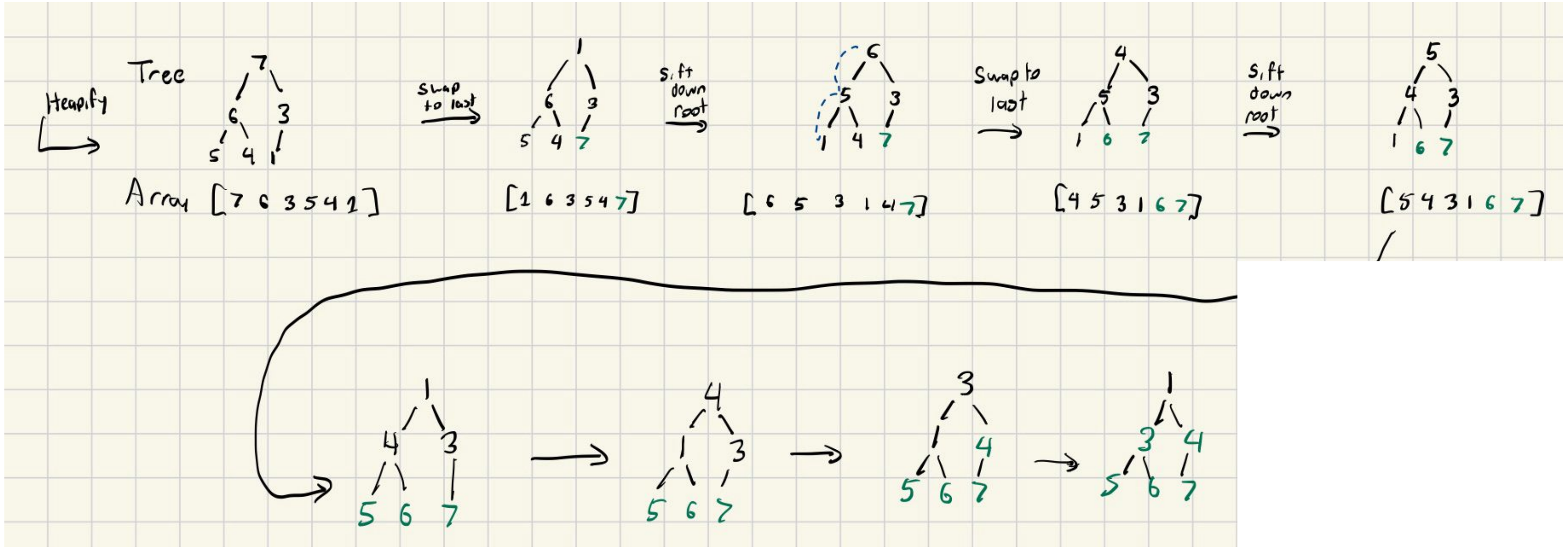
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



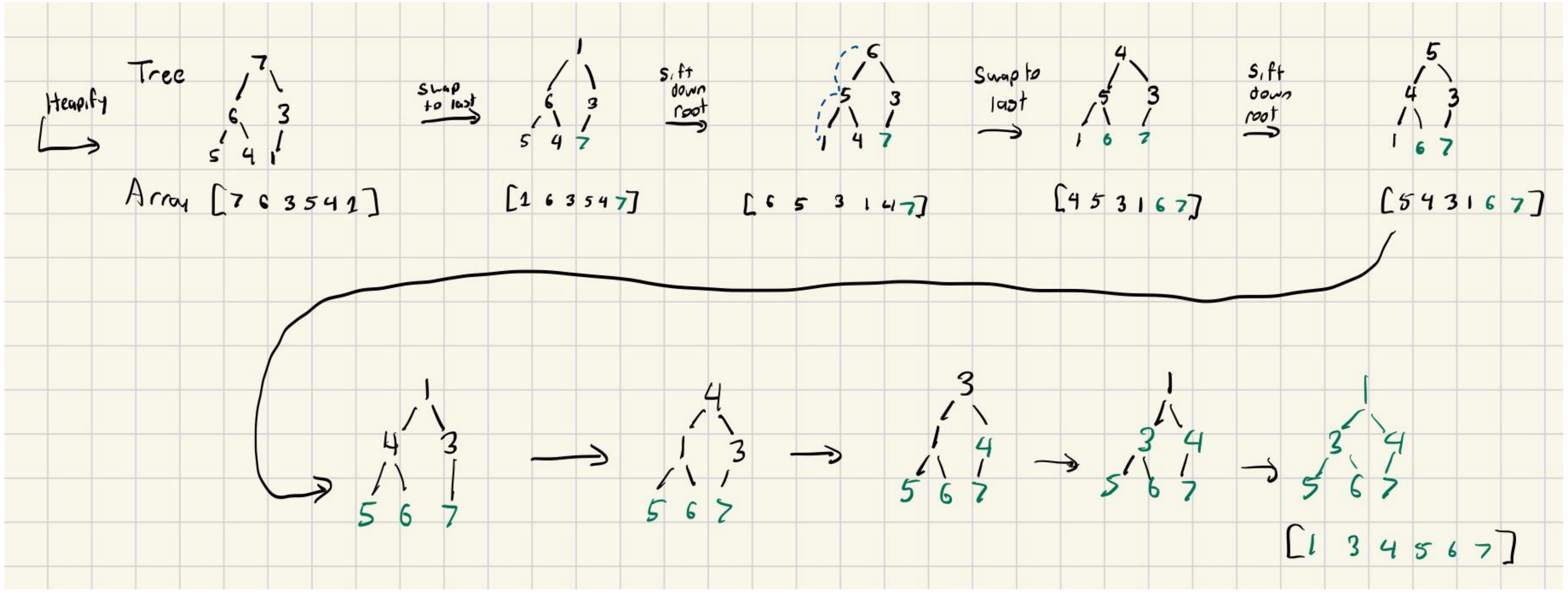
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



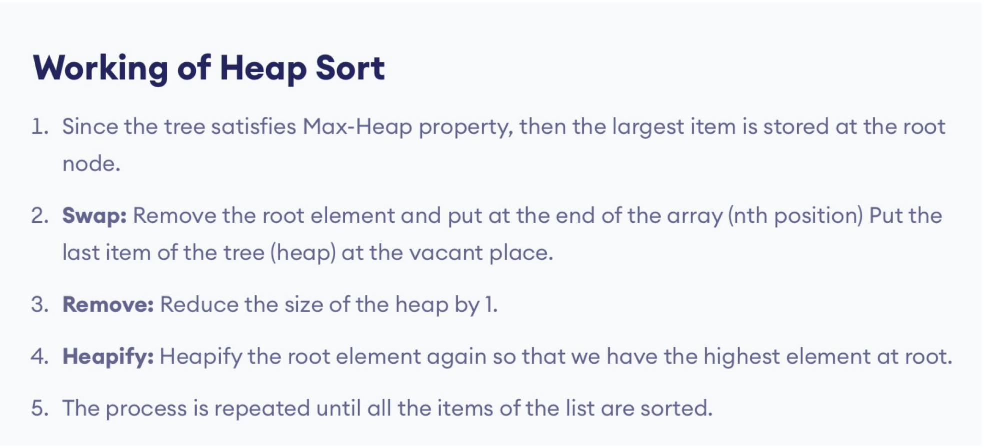
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4



1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4

$O(n \log n)$

Exercise: Equivalent Heap Sorts



Working of Heap Sort

1. Since the tree satisfies Max-Heap property, then the largest item is stored at the root node.
2. **Swap:** Remove the root element and put at the end of the array (nth position) Put the last item of the tree (heap) at the vacant place.
3. **Remove:** Reduce the size of the heap by 1.
4. **Heapify:** Heapify the root element again so that we have the highest element at root.
5. The process is repeated until all the items of the list are sorted.

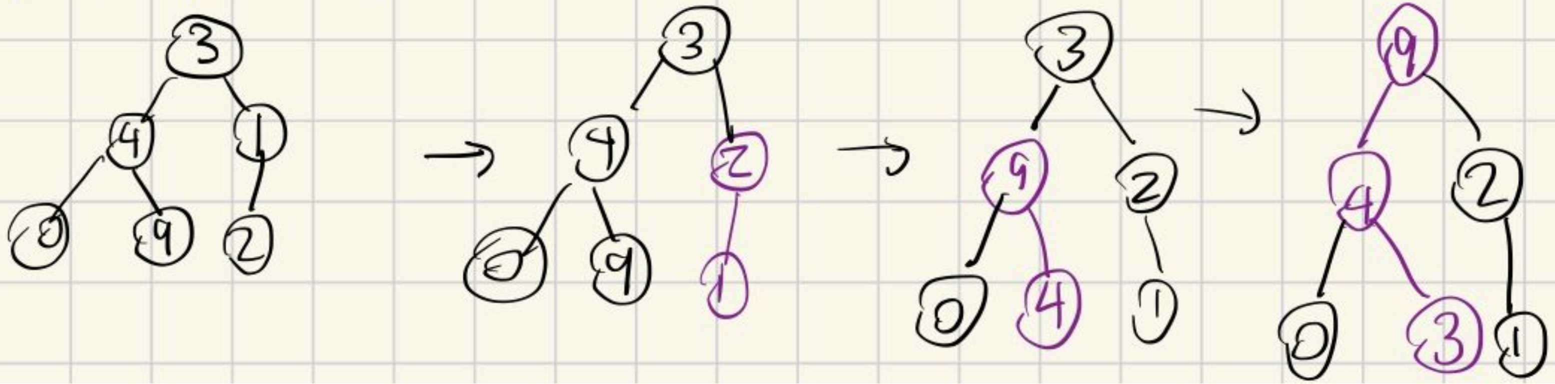
1. Heapify your array
2. Swap root with last leaf, excluding the elements you've already swapped
3. Fix heap by sifting down, excluding the elements you've already swapped
4. Repeat steps 2-4

(Heap sort) In the following questions, we consider Heap sort using **Heapify**.

(1) Show the array $\{3, 4, 1, 0, 9, 2\}$ as it goes through Heap sort (in the ascending order).

1) [3, 4, 1, 0 9, 2]

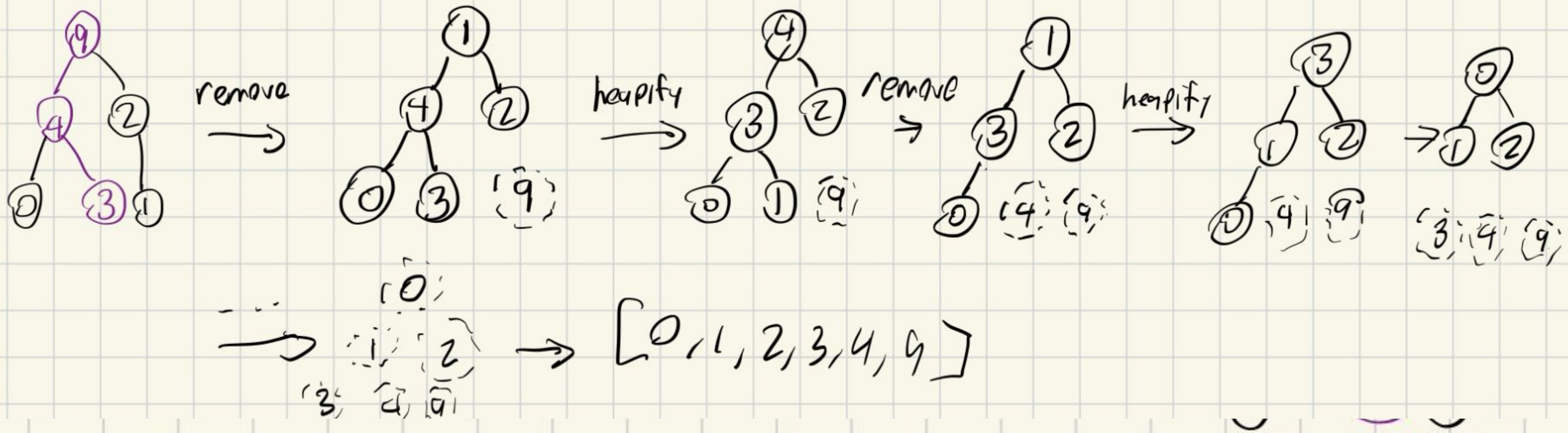
step1) array to heap



(Heap sort) In the following questions, we consider Heap sort using **Heapify**.

(1) Show the array $\{3, 4, 1, 0, 9, 2\}$ as it goes through Heap sort (in the ascending order).

Step 2) The sort: remove root, replace by last leaf, reduce heapsize by 1, heapify, repeat.



$\rightarrow [0, 1, 2, 3, 4, 9]$

Heap Summary Costs

For a heap with 🎺 items,

Heapify: O(🎺)

Add/Pop: O(log 🎺)

Heap Sort: O(🎺log 🎺)

(2) Given K number of sorted (ascending ordered) arrays each having N/K elements in it, your task is to merge all these arrays to form a N-element final sorted array (also in the ascending order).

(2.1) Propose a simple solution to the problem which may run in $O(N\log(N))$ time.

(2) Given K number of sorted (ascending ordered) arrays each having N/K elements in it, your task is to merge all these arrays to form a N-element final sorted array (also in the ascending order).

(2.1) Propose a simple solution to the problem which may run in $O(N\log(N))$ time.

Just run merge sort on the combined array

(2) Given K number of sorted (ascending ordered) arrays each having N/K elements in it, your task is to merge all these arrays to form a N-element final sorted array (also in the ascending order).

(2.1) Propose a simple solution to the problem which may run in $O(N\log(N))$ time.

(2.2) Can you propose a better algorithm to solve the problem? What is the time complexity of your proposed solution?

Example (N = 12, K = 3):

1	3	5	7

2	4	5	5

9	10	11	12

[1 2 3 4 5 5 5 7 9 10
11 12]

Can I use a heap somehow?

Idea: index-wise heap sorting

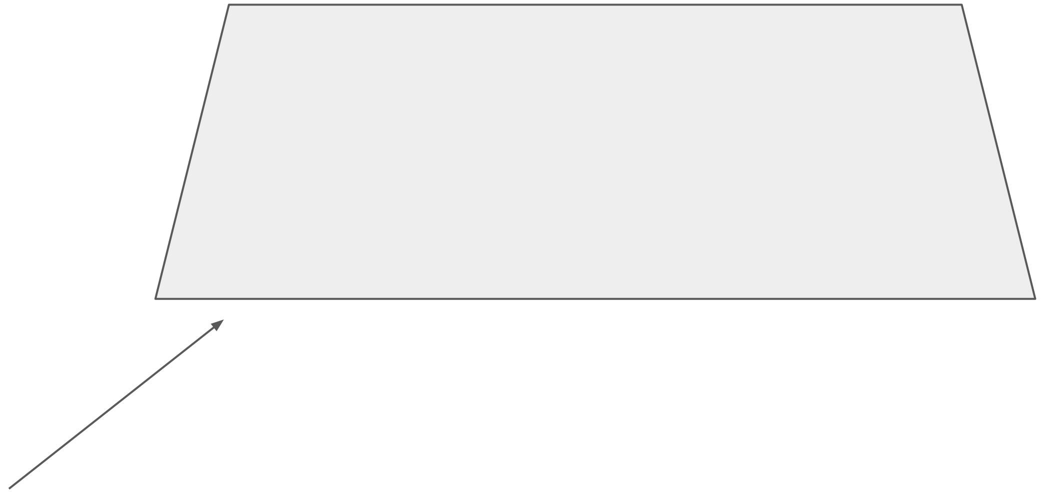
Example (N = 12, K = 3):

1	3	5	7

2	4	5	5

9	10	11	12

Iteratively sort and add items to the heap

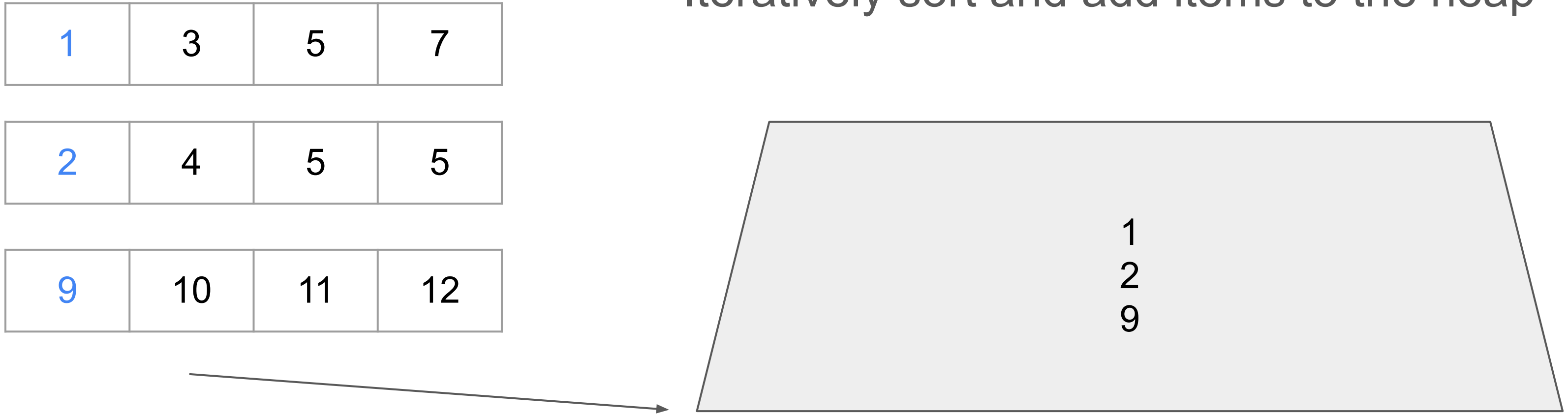


This is a heap (as seen in the wild)

Idea: index-wise heap sorting

Example (N = 12, K = 3):

Iteratively sort and add items to the heap



1. Add first index elements to heap

Idea: index-wise heap sorting

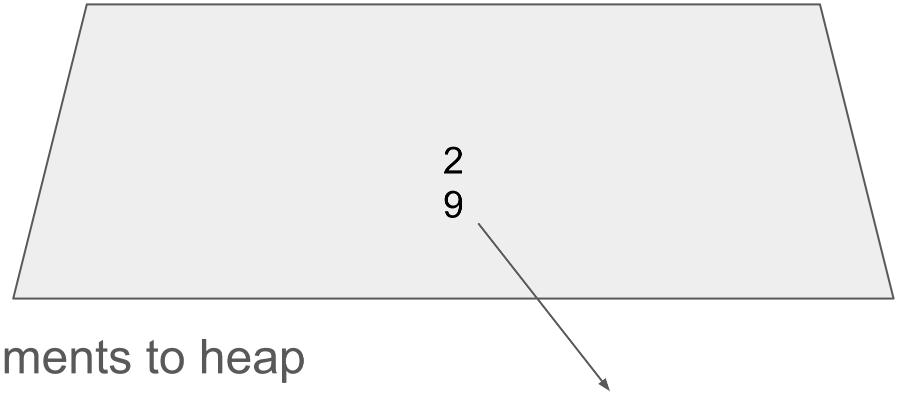
Example (N = 12, K = 3):z

1	3	5	7

2	4	5	5

9	10	11	12

Iteratively sort and add items to the heap



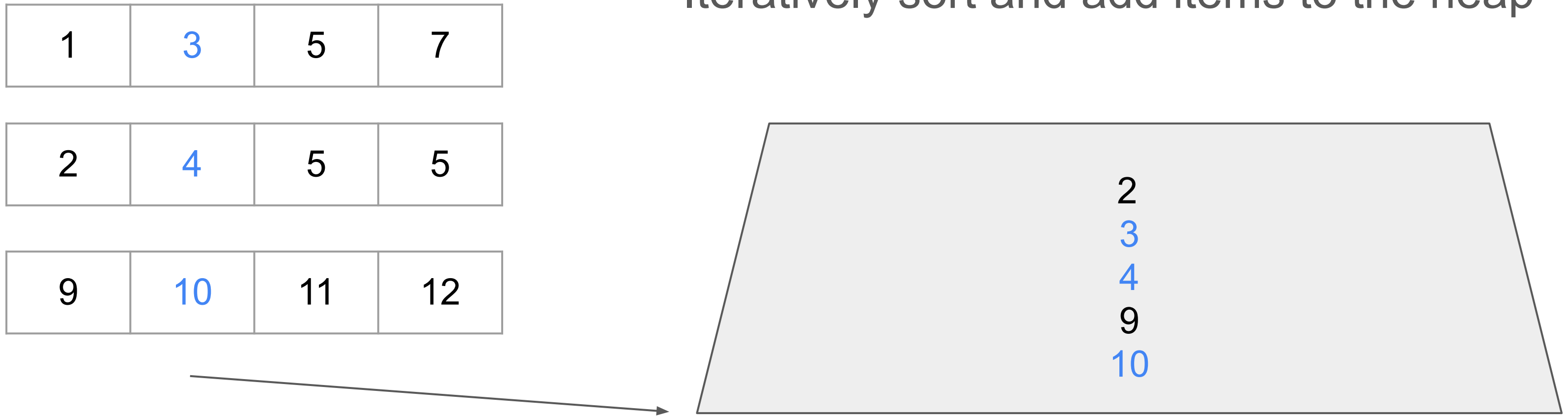
1. Add first index elements to heap
2. Pop heap and append to res array

res = 1

Idea: index-wise heap sorting

Example (N = 12, K = 3):z

Iteratively sort and add items to the heap



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

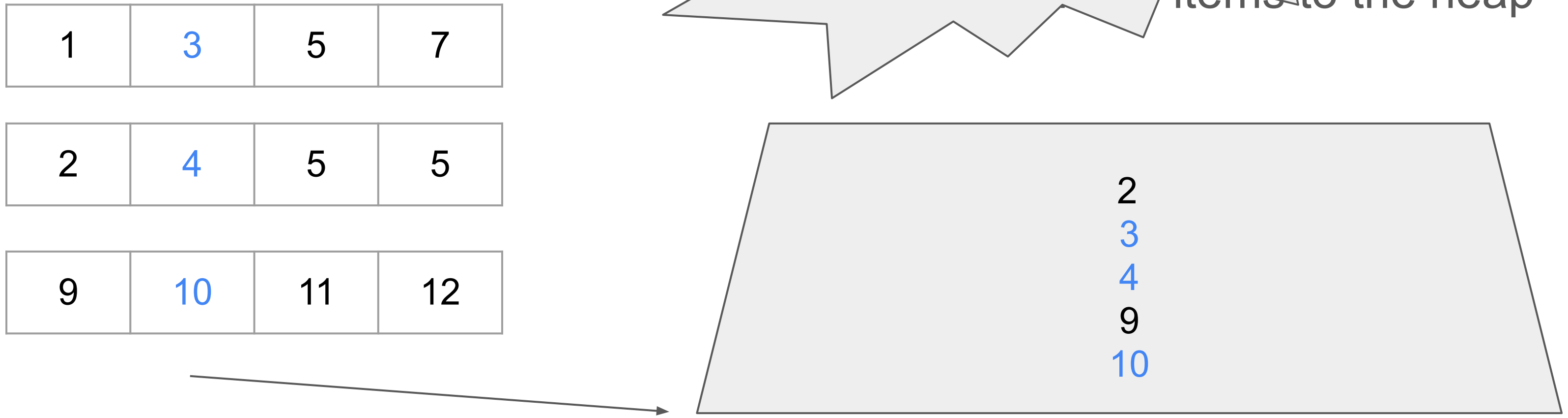
res = 1

Idea: index-wise heap sorting

Problem
Heap size at most N
Overall will be O(NlogN)

Example (N = 12, K = 3):z

It items to the heap



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

res = 1

Keeping our heap to size K

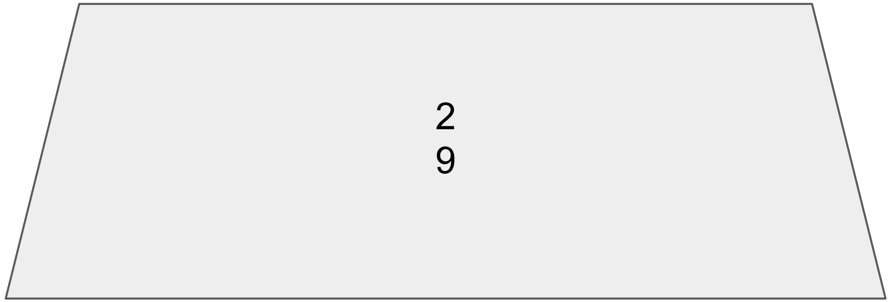
Example (N = 12, K = 3):z

1	3	5	7

2	4	5	5

9	10	11	12

Iteratively sort and add items to the heap



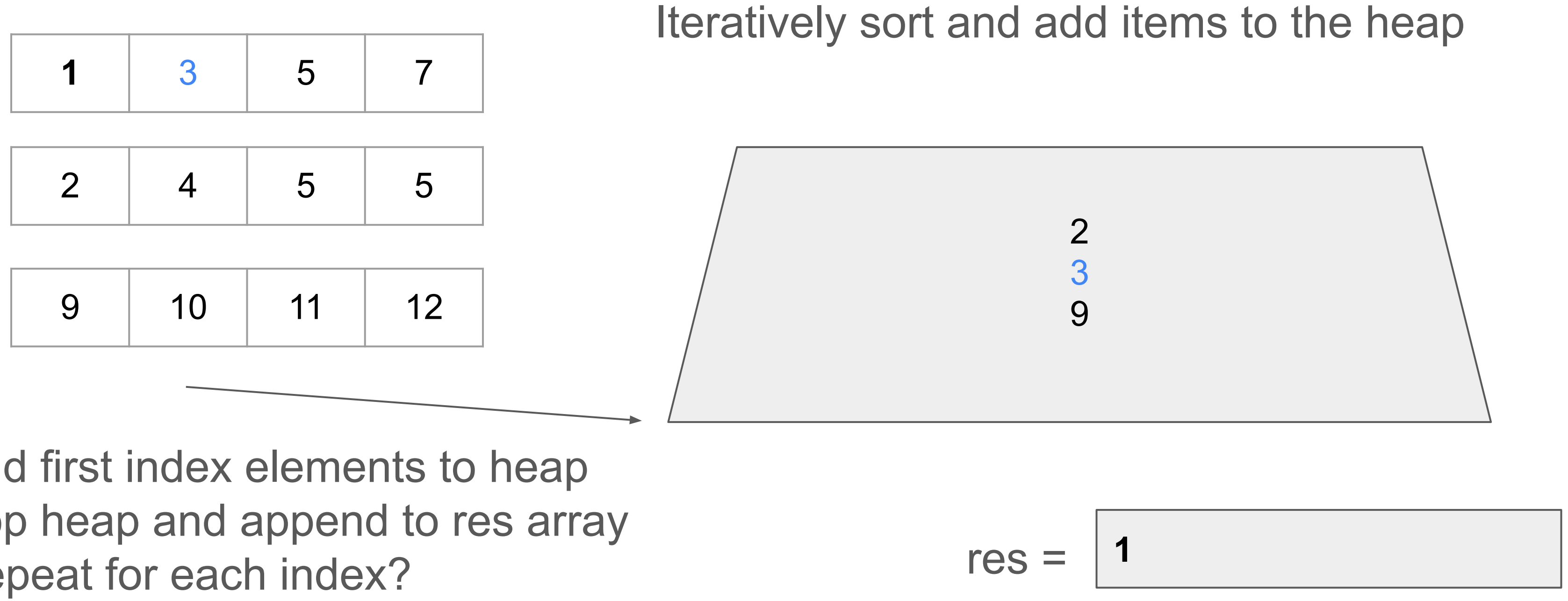
1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

Only add next index element from popped array

res = **1**

Keeping our heap to size K

Example (N = 12, K = 3):z



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

Only add next index element from popped array

Keeping our heap to size K

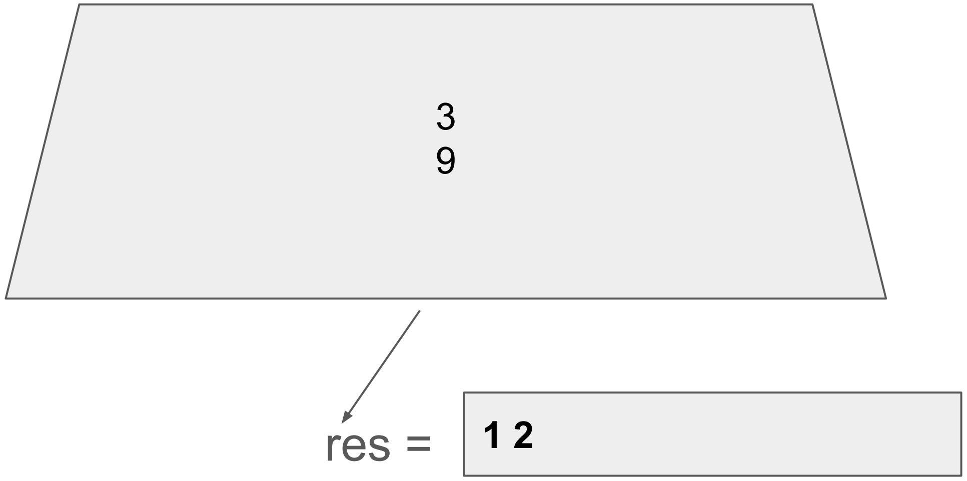
Example (N = 12, K = 3):z

1	3	5	7

2	4	5	5

9	10	11	12

Iteratively sort and add items to the heap



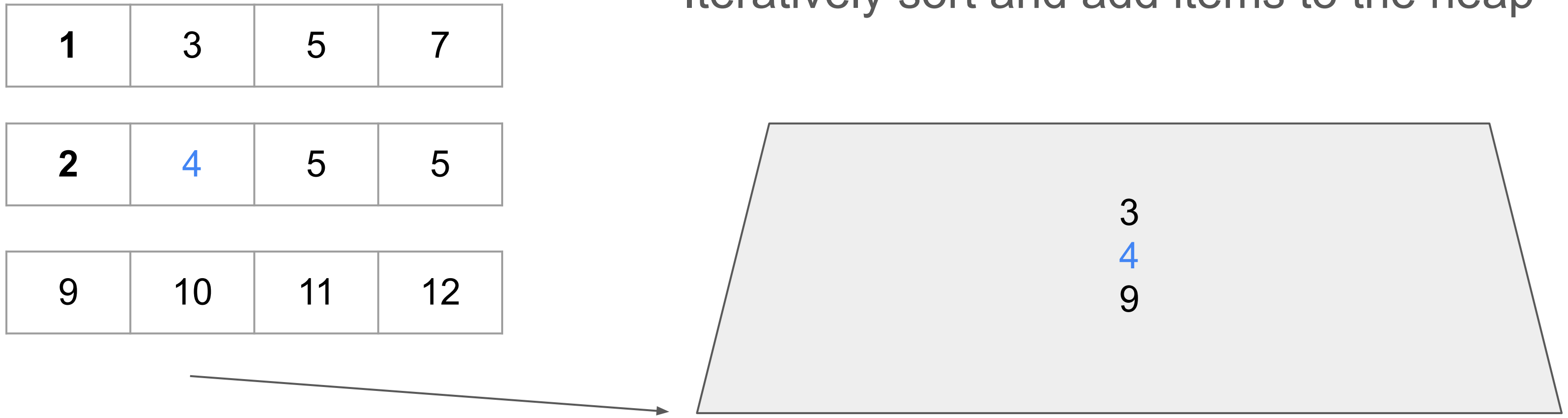
1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

Only add next index element from popped array

Keeping our heap to size K

Example (N = 12, K = 3):z

Iteratively sort and add items to the heap



res = **1 2**

1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

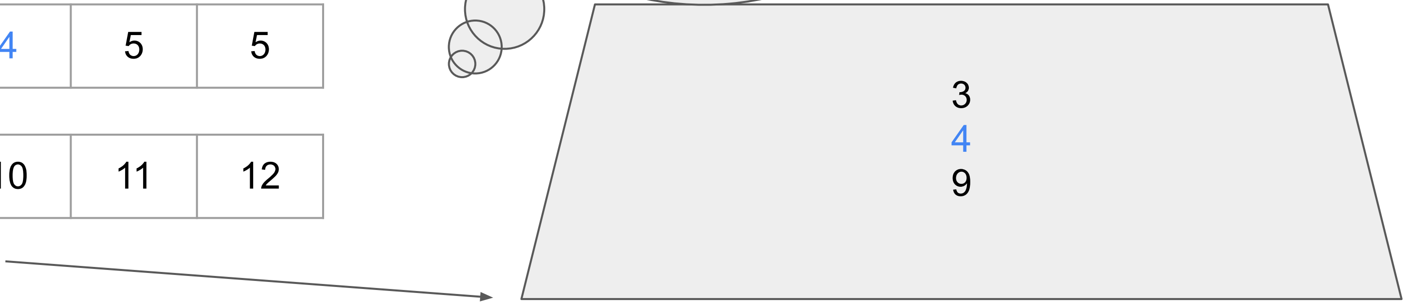
Keeping our heap to

Okay.. but how do we know:
1. which array the popped element belongs to?
2. Its index in the array

Example (N = 12, K =

s to the heap

1	3	5	7
2	4	5	5
9	10	11	12



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

Only add next index element from popped array

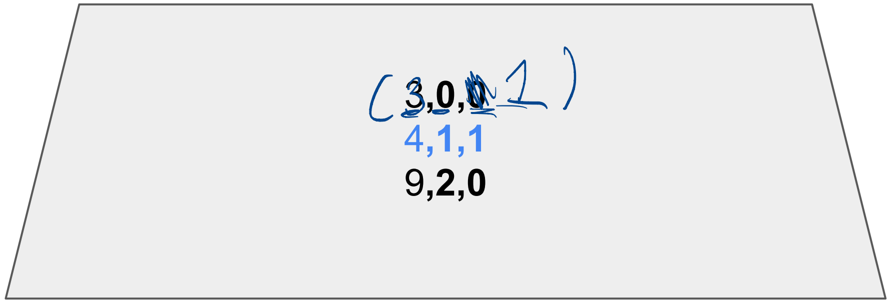
res = **1 2**

Store the array number and the index!

Example (N = 12, K = 3):z

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

Iteratively sort and add items to the heap **of the form x,array #,index**



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

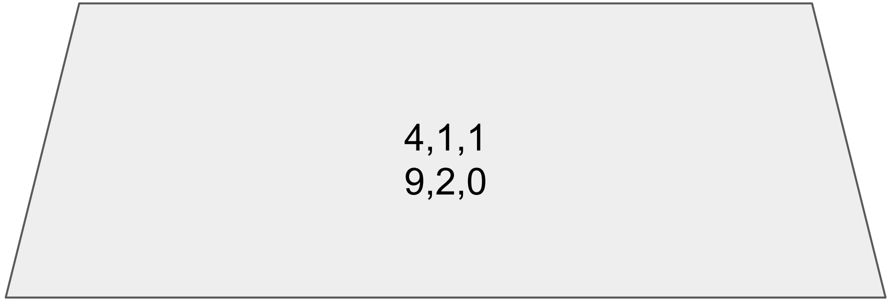
res = **1 2**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	5	7
1	**2**	4	5	5
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



res = **1 2 3**

1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

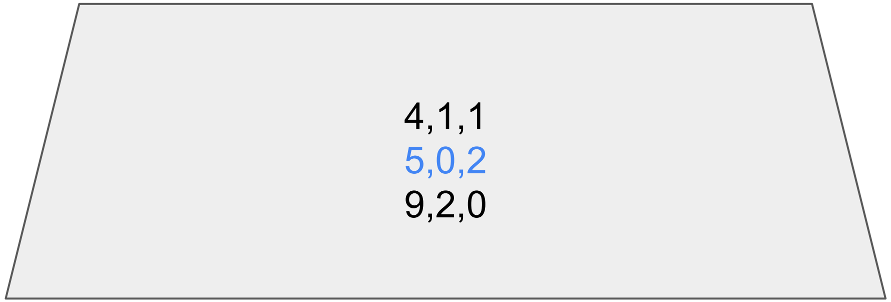
Only add next index element from popped array

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	5	7
1	**2**	4	5	5
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



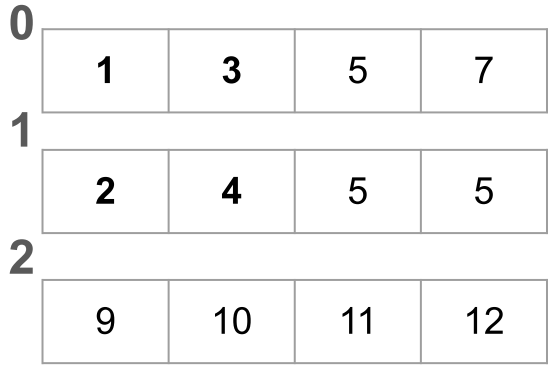
res = **1 2 3**

1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

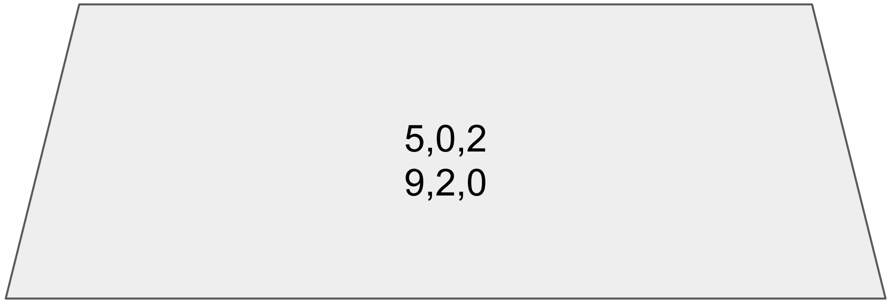
Only add next index element from popped array

Store the index

Example (N = 12, K = 3):z



Iteratively sort and add items to the heap **of the form x,array #,index**



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

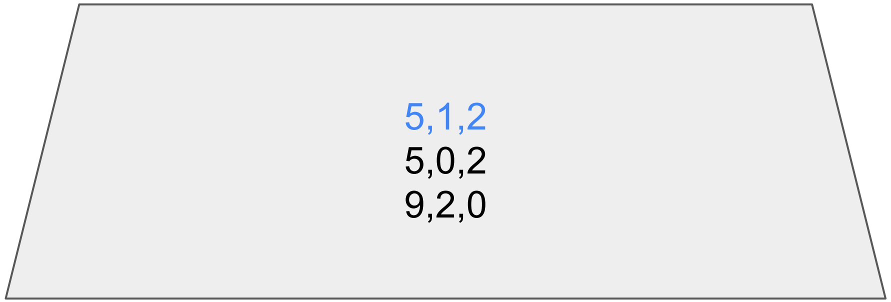
res = **1 2 3 4**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	5	7
1	**2**	**4**	5	5
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**

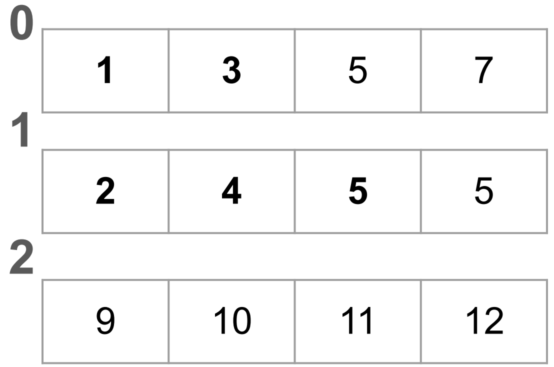


res = **1 2 3 4**

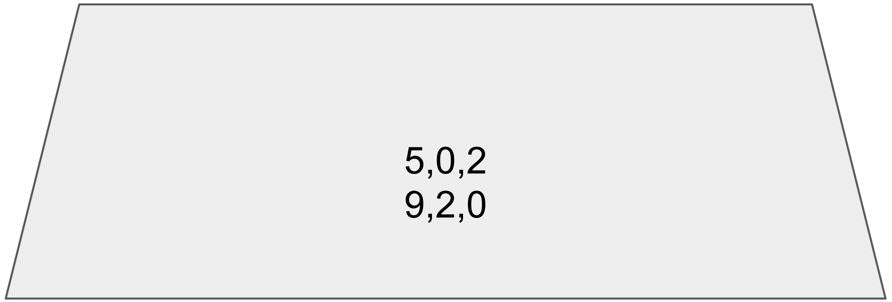
1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?
   Only add next index element from popped array

Store the index

Example (N = 12, K = 3):z



Iteratively sort and add items to the heap **of the form x,array #,index**



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?
   Only add next index element from popped array

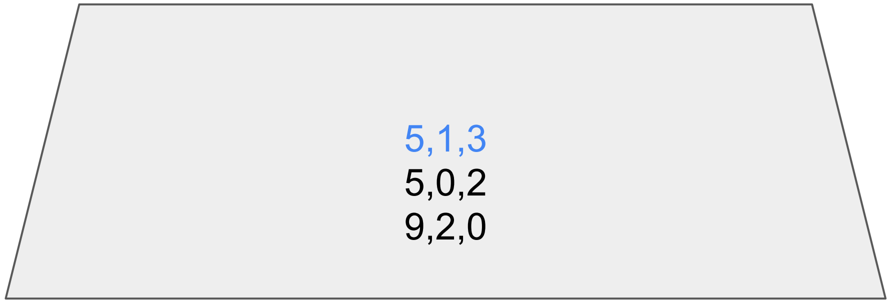
res = **1 2 3 4 5**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	5	7
1	**2**	**4**	**5**	5
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



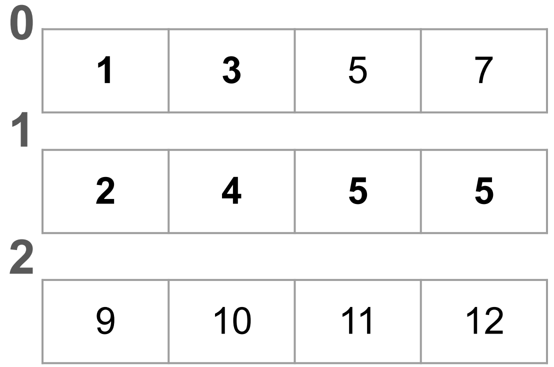
1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

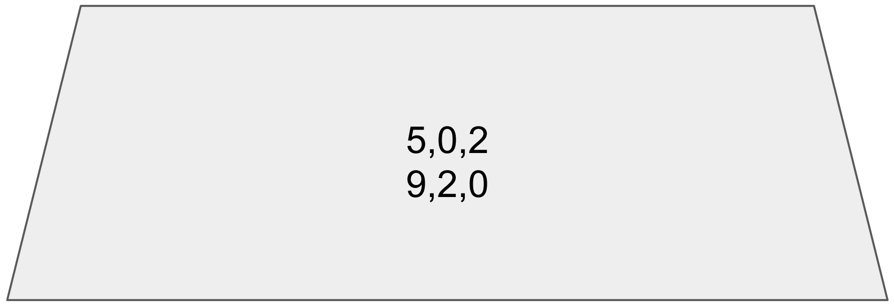
res = **1 2 3 4 5**

Store the index

Example (N = 12, K = 3):z



Keep sorting the remaining arrays



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?
   Only add next index element from popped array

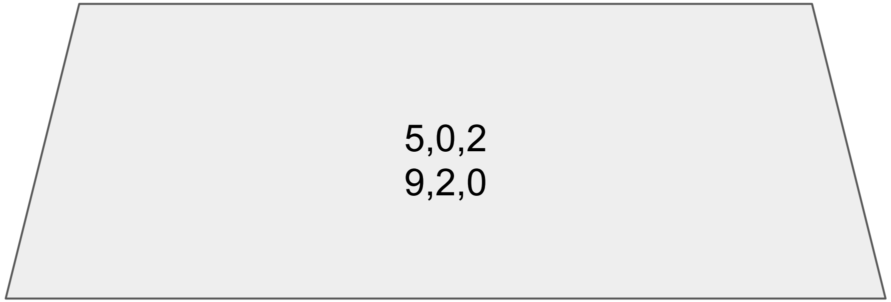
res = **1 2 3 4 5 5**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	5	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?
   Only add next index element from popped array

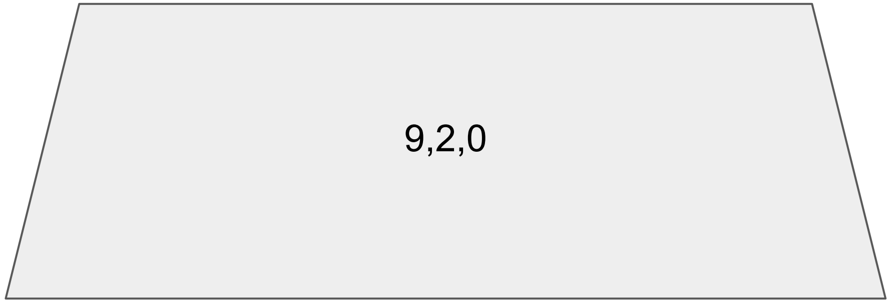
res = **1 2 3 4 5 5**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	**5**	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

Only add next index element from popped array

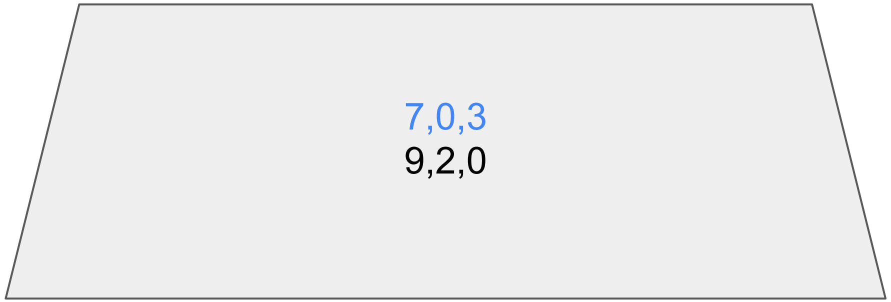
res = **1 2 3 4 5 5 5**

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	**5**	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



res = **1 2 3 4 5 5 5**

1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

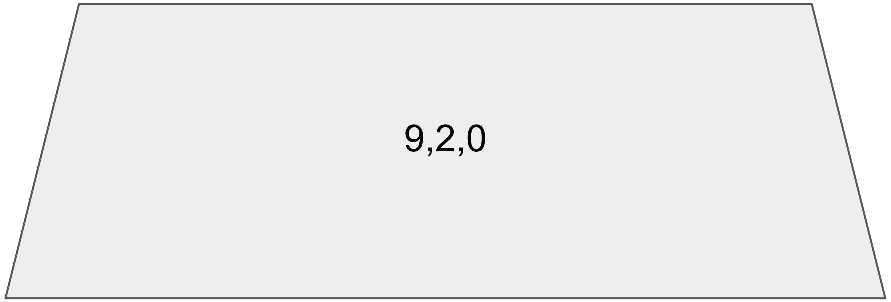
   Only add next index element from popped array

Store the index

Example (N = 12, K = 3):z

0	**1**	**3**	**5**	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12

Iteratively sort and add items to the heap **of the form x,array #,index**



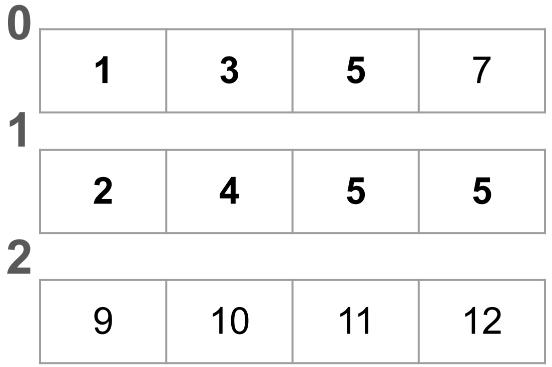
1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

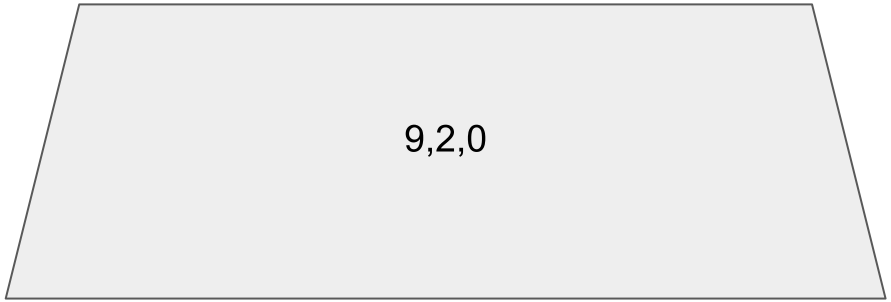
res = **1 2 3 4 5 5 5 7**

Store the index

Example (N = 12, K = 3):z



Add everything left from last array to res



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

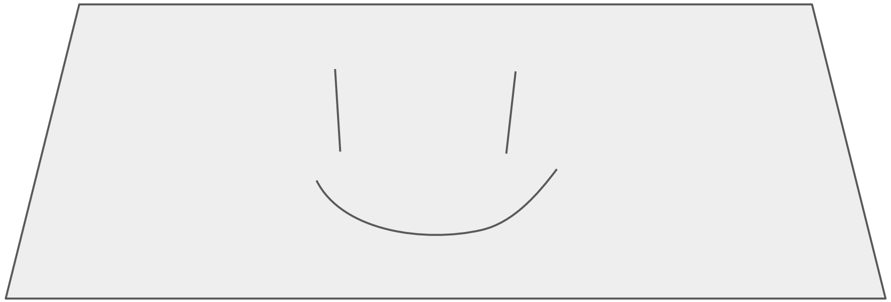
res = 1 2 3 4 5 5 5 7

Time complexity?

Example (N = 12, K = 3):z

0	**1**	**3**	**5**	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12

Add everything left from last array to res



1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

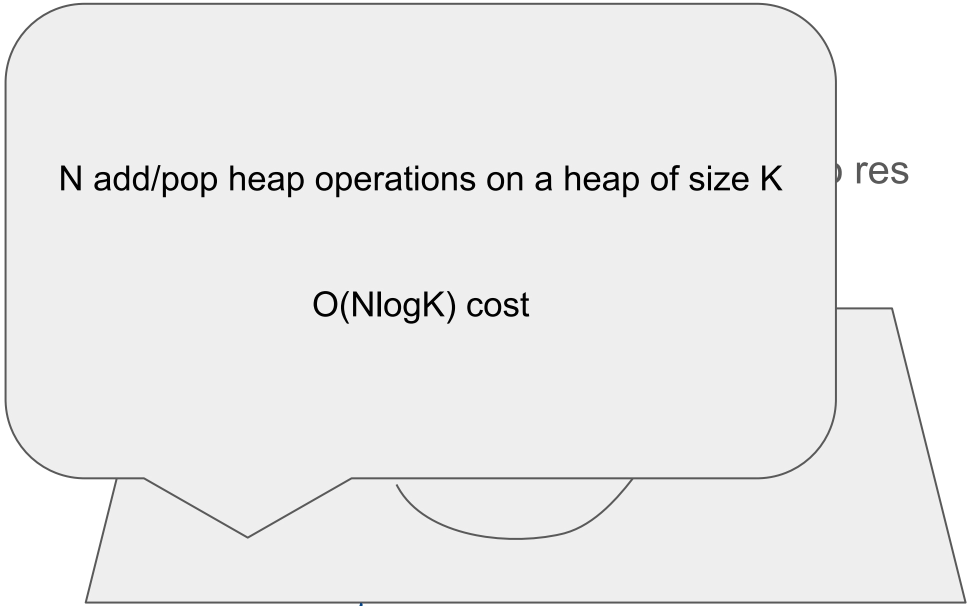
   Only add next index element from popped array

res = **1 2 3 4 5 5 5 7** 9 10 11 12

Time complexity?

Example (N = 12, K = 3):z

0	**1**	**3**	**5**	7
1	**2**	**4**	**5**	**5**
2	9	10	11	12



N add/pop heap operations on a heap of size K

O(NlogK) cost

N/K

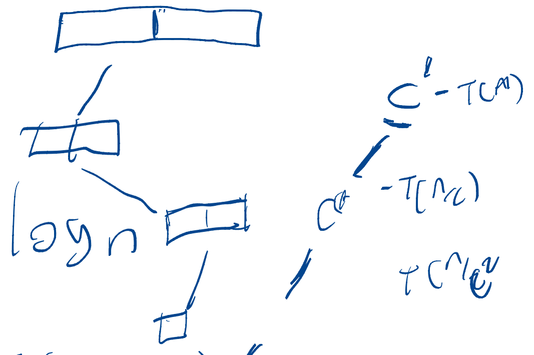
res = **1 2 3 4 5 5 5 7** 9 10 11 12

1. Add first index elements to heap
2. Pop heap and append to res array
3. Repeat for each index?

   Only add next index element from popped array

Thank you!

Bonus content after this slide..



$$\underbrace{T(n) = \underline{T(n/c)} + \underline{c'}}_{\text{for } c, c' \text{ constants.}}$$

Cost at each level: c'

levels: $\log_c(n)$

$c' \log_c(n)$

$\in O(\log n)$

Are you the root of my heap?
Because you're my #1 priority

From:
To:

Your photo here

There exists a constant c and $n_0 > 0$ such that

$$\text{myAttractionToYou}(n) \geq g(n)$$

for all non-constant $g(n)$ and $n \geq n_0$.

From:
To:

Your photo here

$$\lim_{n \to \infty} \frac{g(n)}{\mathrm{myAttractionToYou}(n)} = 0$$

for all non-constant g(n).

From:
To:

Your photo here

Are you a linked list? Because I love you from head to tail

From:
To:

Your photo here

Be-leaf me, I'm not a complete binary tree without you at my (left-most) side

From:
To:

Your photo here

Roses are red,
Array resize is amortized,
Sorry, I got lost in your amber eyes

From:
To:

Your photo here

I’m approaching you, asymptotically

From:
To:

Your photo here