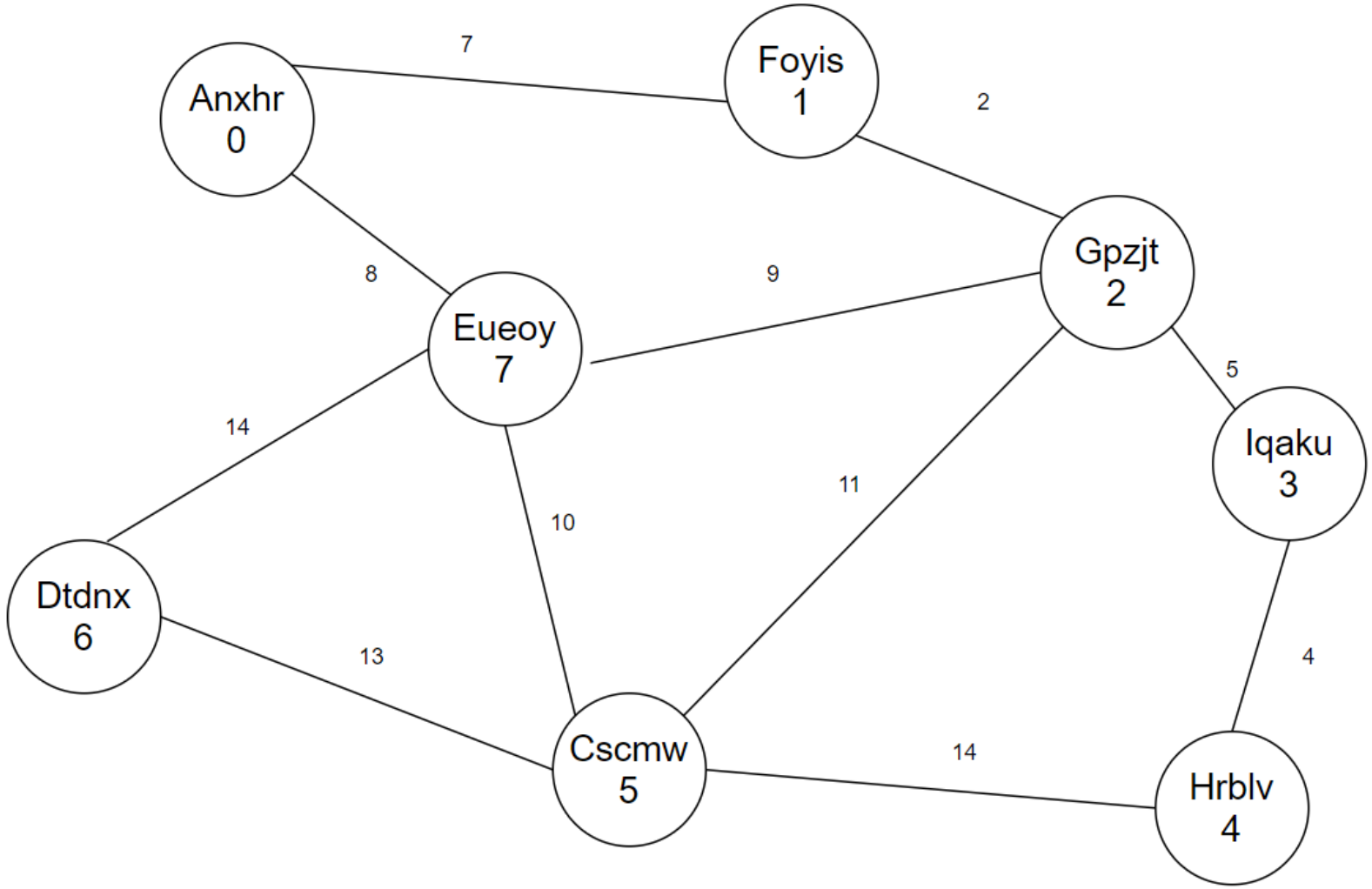
CSE 2320

Week of 08/10/2020

Instructor : Donna French

Coding Assignment 3

```
Anxhr,1,7,7,8
Foyis,0,7,2,2
Gpzjt,1,2,7,9,5,11,3,5
Iqaku,2,5,4,4
Hrblv,5,14,3,4
Cscmw,6,13,7,10,2,11,4,14
Dtdnx,5,13,7,14
Eueoy,0,8,6,14,5,10,2,9
```



Coding Assignment 3

Test GTA - Run student's program with the GraphGrade.txt file provided by the instructor. Confirm that the program outputs the correct adjacency matrix and vertex array (correct values will be provided by the instructor).	30 pts Full Marks	0 pts No Marks	30 pts

Coding Assignment 3

```
student@cse1325:/media/sf_VM2320$ ./a.out GraphGrade.txt

   -1       7      -1      -1      -1      -1      -1       8
    7      -1       2      -1      -1      -1      -1      -1
   -1       2      -1       5      -1      11      -1       9
   -1      -1       5      -1       4      -1      -1      -1
   -1      -1      -1       4      -1      14      -1      -1
   -1      -1      11      -1      14      -1      13      10
   -1      -1      -1      -1      -1      13      -1      14
    8      -1       9      -1      -1      10      14      -1

What is the starting vertex? Dtdnx


I       L       D       P       V
0       Anxhr   22      7       1
1       Foyis   25      2       1
2       Gpzjt   23      7       1
3       Iqaku   28      2       1
4       Hrblv   27      5       1
5       Cscmw   13      6       1
6       Dtdnx   0       -1      1
7       Eueoy   14      6       1

What is the destination vertex? Iqaku
The path from Dtdnx to Iqaku is Dtdnx->Eueoy->Gpzjt->Iqaku and has a length of 28
```

30 points

1 point awarded per correct line of adjacency matrix = 8 points

2 points awarded per correct line of vertex array – 1 point for Distance and 1 point for Previous = 16 points

2 points for correct vertices in path
2 points for path in correct order
2 points for correct path length

OLQ9

Question 1

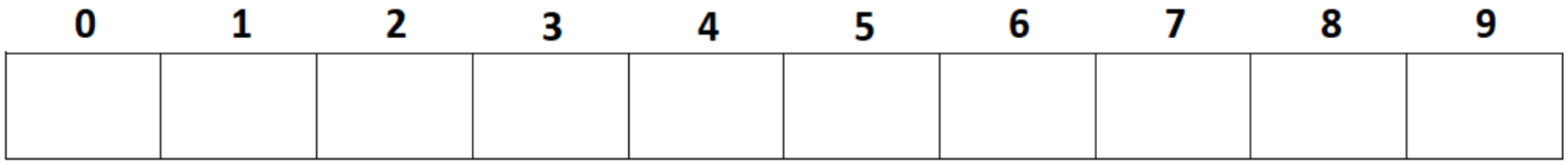
1 pts

Given a list of items to be stored in the hash table and the hashing function, show what the array will look like after all items have been hashed and stored when using open addressing. The array in use for this question has 11 elements.

List : Jan, Tim, Mia and Sue.

Hash function : add ASCII values and MOD with array size to get an array index.

Jan(281), Tim(296), Mia(279), Sue(301)



Jan [Choose]

Sue [Choose]

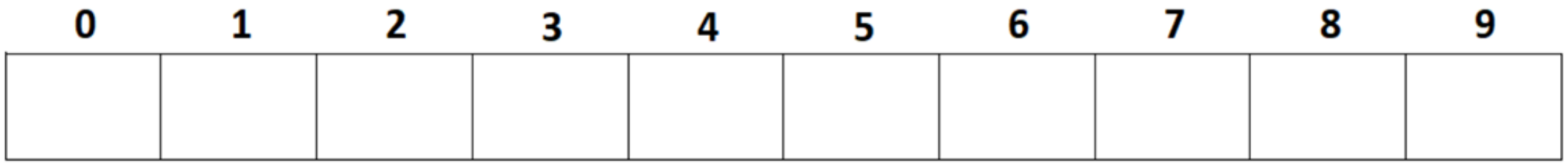
Tim [Choose]

Mia [Choose]

[Choose]
Array element 2
Array element 5
Array element 9
Array element 7
Array element 6
Array element 3
Array element 8
Array element 0
Array element 1
Array element 4

OLQ9

Jan(281), Tim(296), Mia(279), Sue(301)



Jan	Array element 1
Sue	Array element 2
Tim	Array element 6
Mia	Array element 9

OLQ9

Question 1

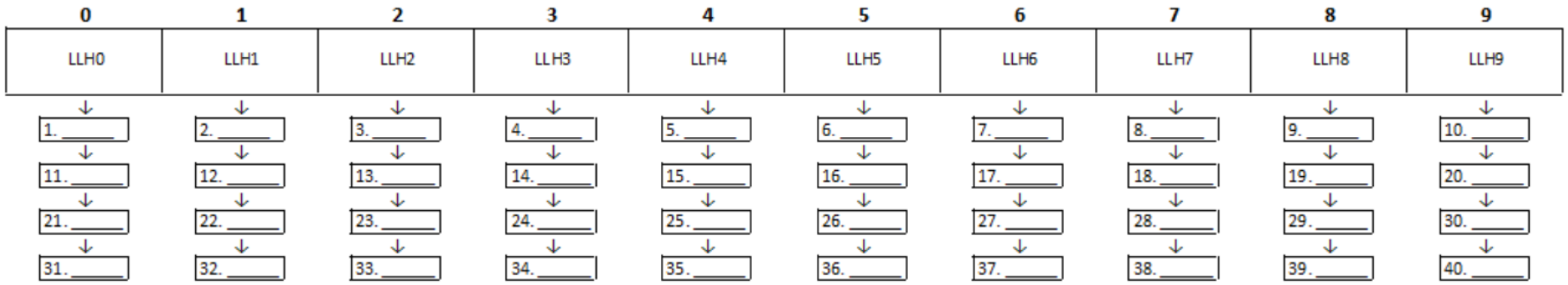
1 pts

Given a list of items to be stored in the hash table and the hashing function, show what the array will look like after all items have been hashed and stored when using separate chaining. The array in use for this question has 11 elements.

List : Jan, Tim, Mia and Sue.

Hash function : add ASCII values and MOD with array size to get an array index.

Jan(281), Tim(296), Mia(279), Sue(301)



Jan [Choose]

Tim [Choose]

Mia [Choose]

Sue [Choose]

Check for a scroll bar on the drop down box.

OLQ9

Numeric keys

Use MOD with the size of the array to get a key that is within the bounds of the array

Alphanumeric keys

Add the ASCII values of a string and use MOD with the size of the array to get a key that is within the bounds of the array

Folding Method

Rather than just adding the numbers of a phone number, for example, add the area code and then the prefix and then the exchange and use those 3 numbers as the array index.

214 772 2387 folds to become index 772.

Deleting from a Heap

If you had a stack of cans like this one, how would you remove a can?

Without making a mess or knocking them over?

This is not a game of Jenga

so we would take the top can.

Deleting from a Heap

We delete from a heap the same way – we take the top element.

For a max heap, the top element/root is the biggest element in the heap.

For a min heap, the top element/root is the smallest element in the heap.

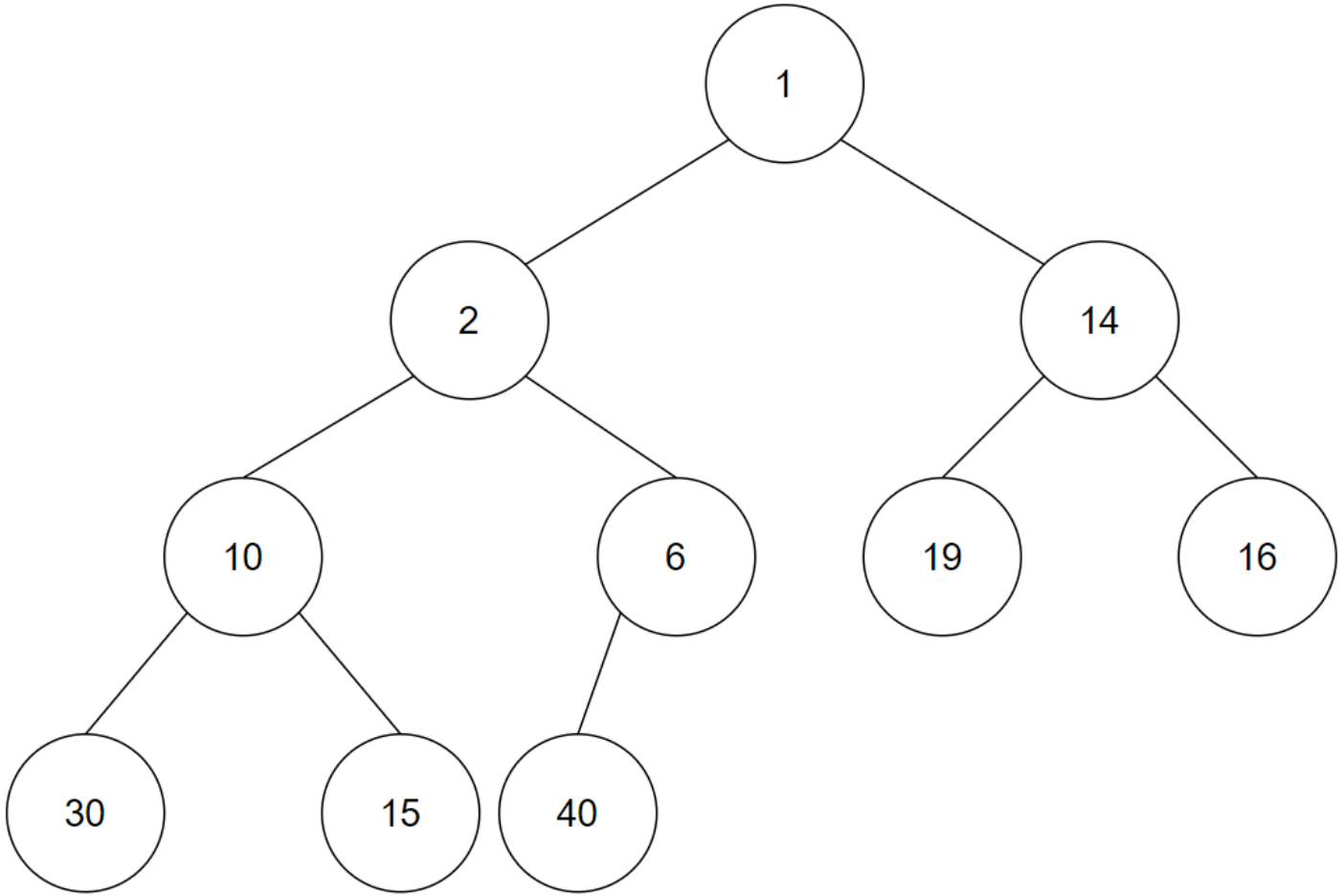
Deleting from a Min Heap

So what happens when we remove the root?

We need a new root?

Do we pick one of the root's children?

No.



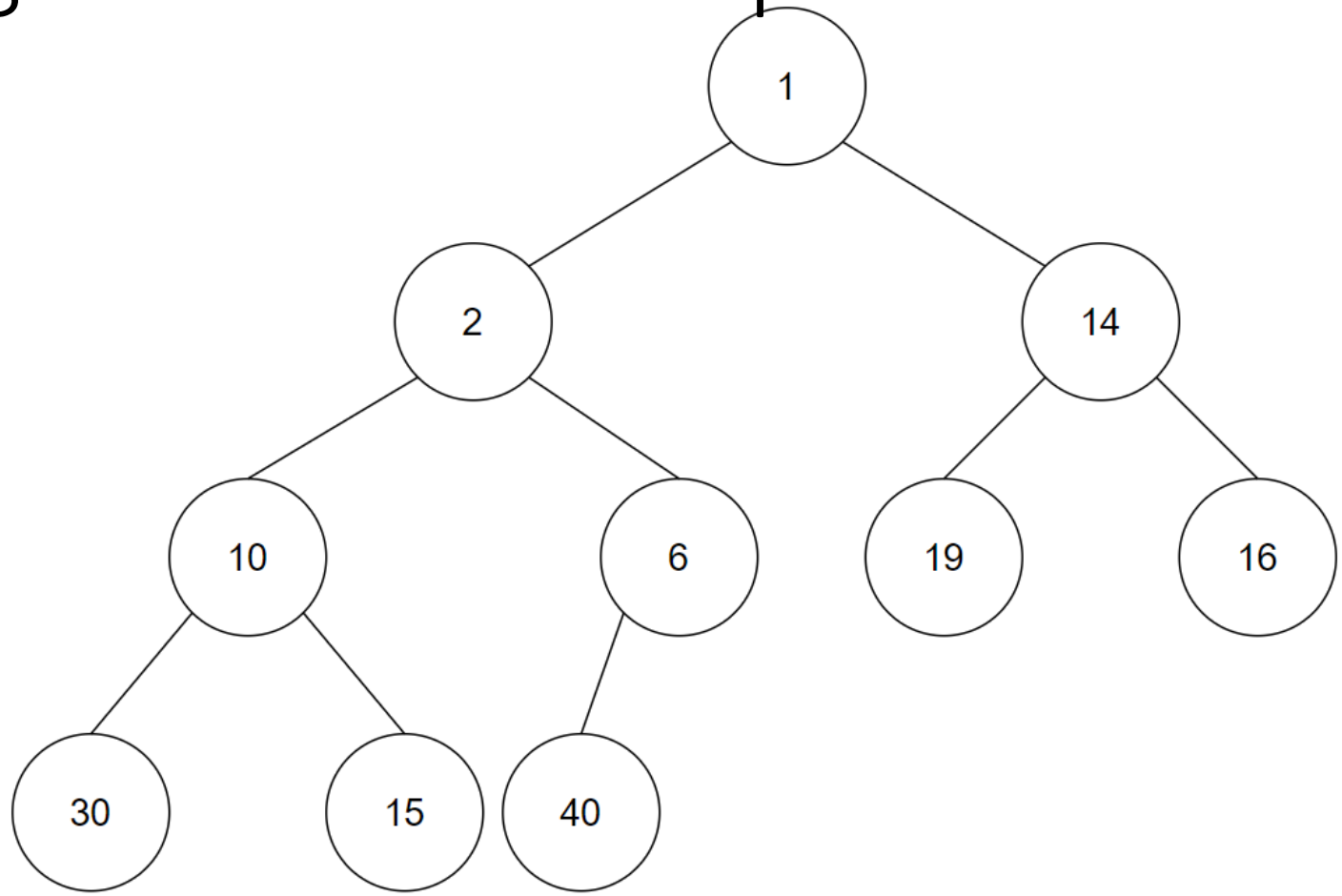
Deleting from a Min Heap

Let's look at the array again.

If we take the value out of the 0^{th} element, what is the easiest replacement value from the array?

If we take element 1, then we'll just have another gap.

We've already said that we don't like gaps in our arrays.



0	1	2	3	4	5	6	7	8	9
	2	14	10	6	19	16	30	15	40

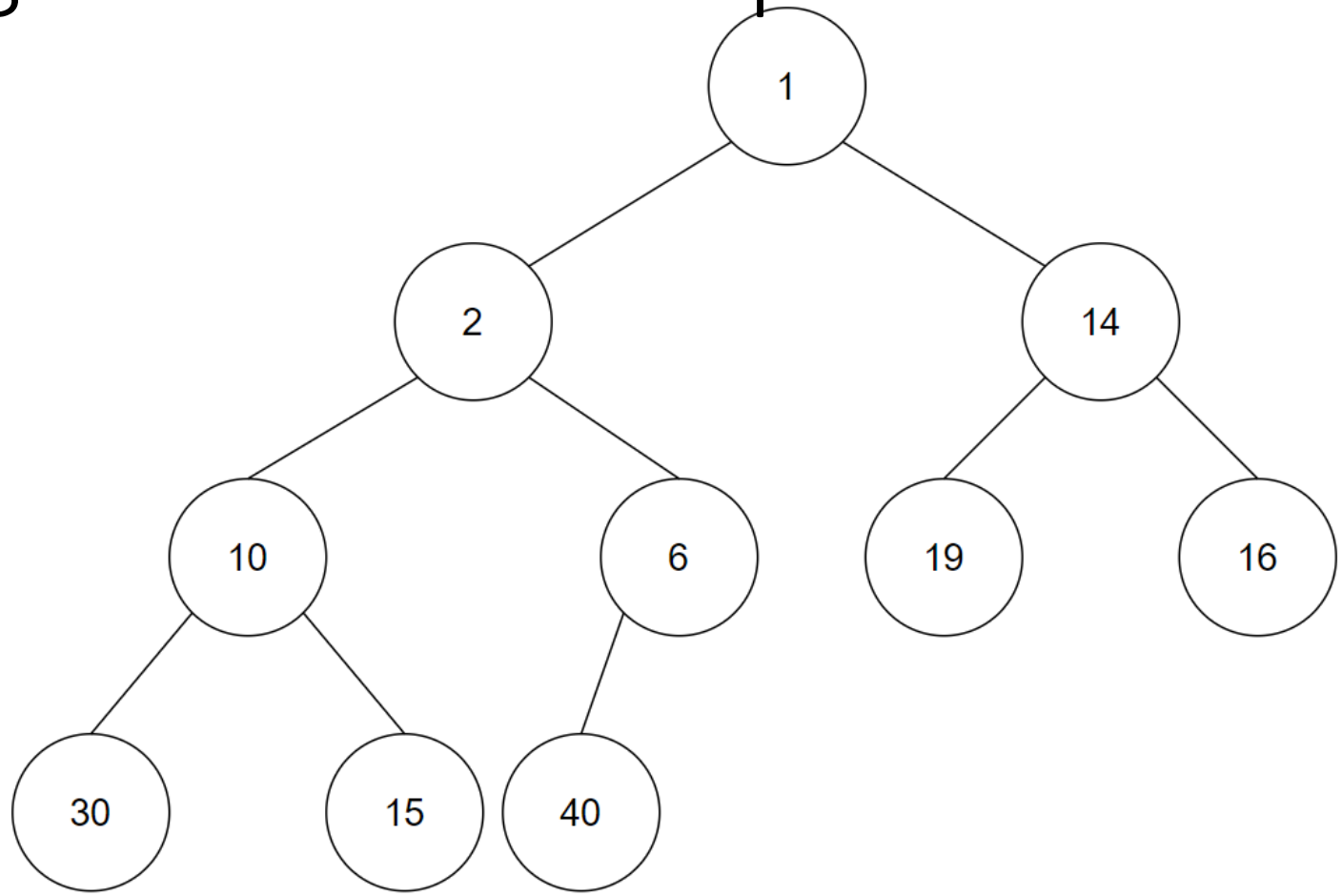
Deleting from a Min Heap

What if we move the last element to the 0^{th} spot?

It's OK to have spaces at the END of the array.

What will the heap look like?

Do we have a problem?



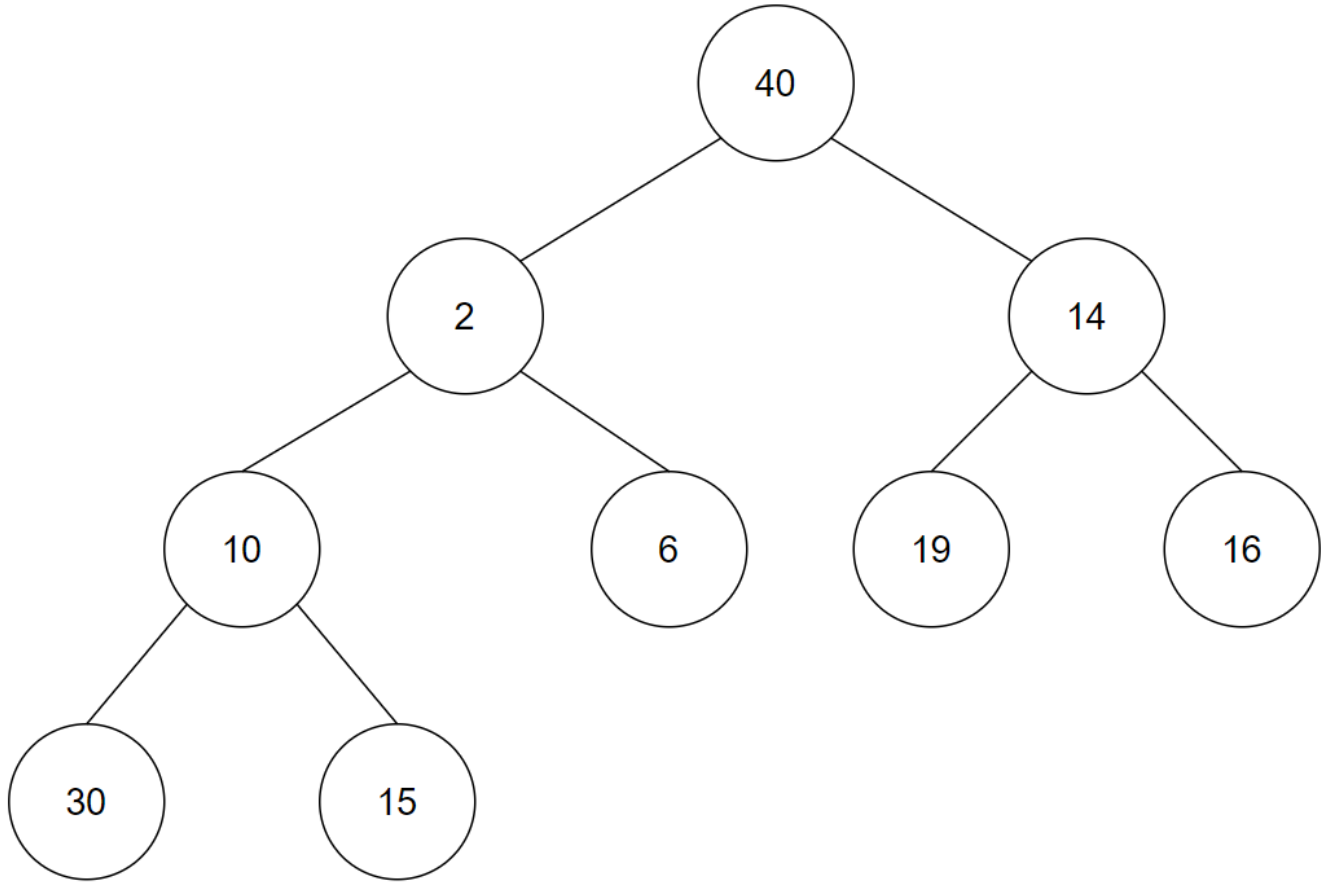
0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	

Deleting from a Min Heap

Do we have a problem?

We have lost our heap property again.

The tree is still complete but 40 is too big to be the root of this min heap.



0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	

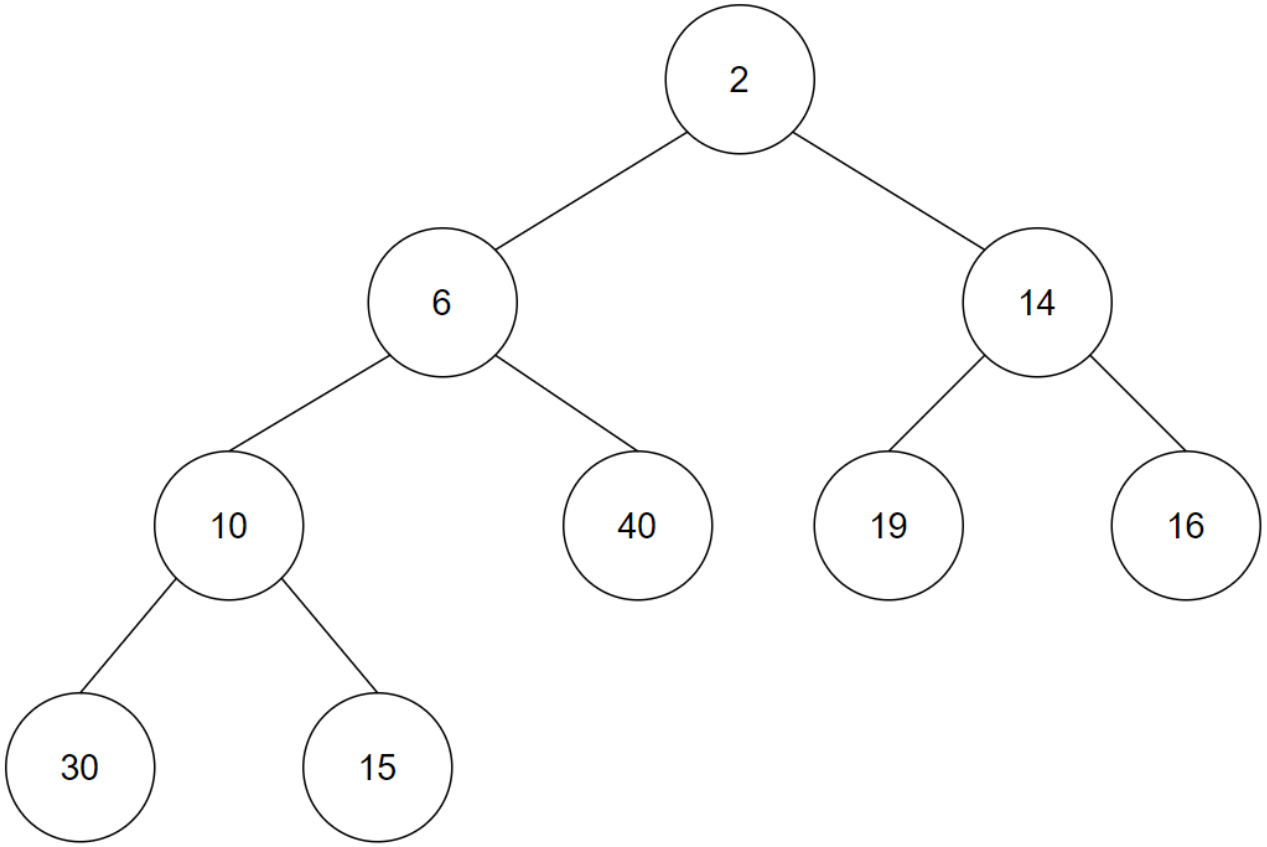
Deleting from a Min Heap

We are back to a min heap

Complete tree

Parents smaller than children

Root is smallest value.



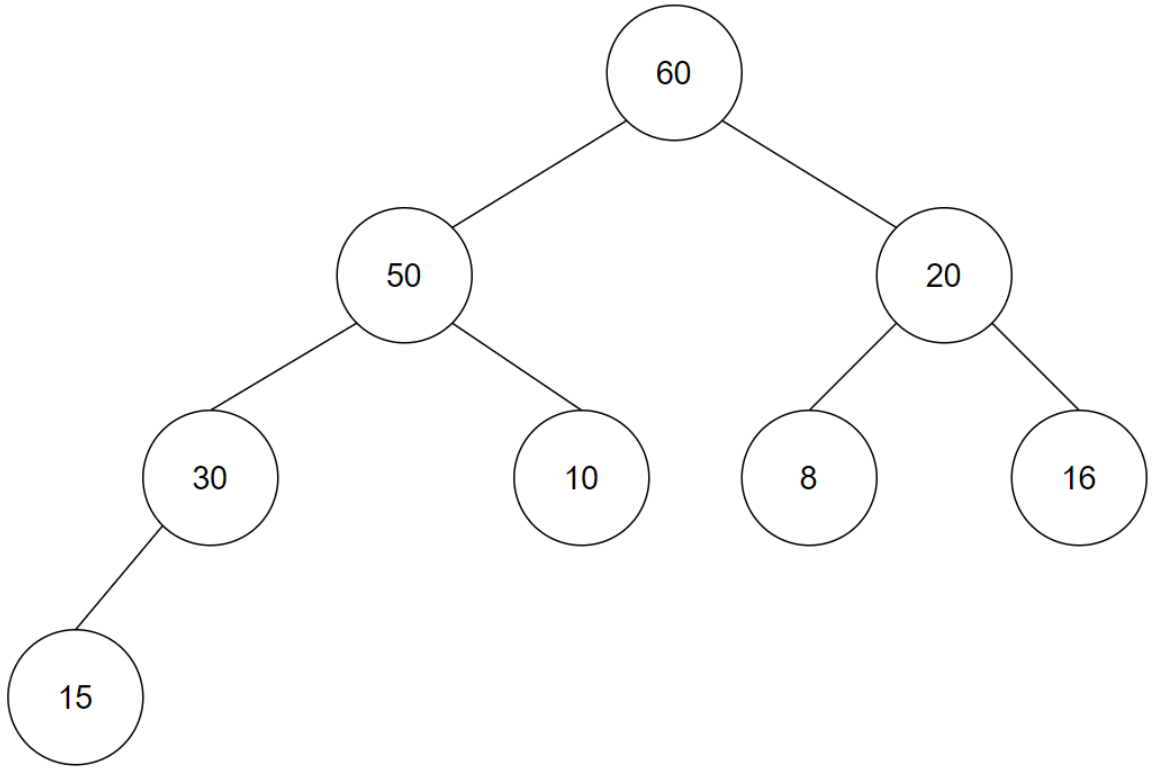
0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	

Deleting from a Max Heap

Let's try the same technique with a max heap.

We want remove 60.

We take 60 out and swap it with the last value of 15.



0	1	2	3	4	5	6	7
60	50	20	30	10	8	16	15

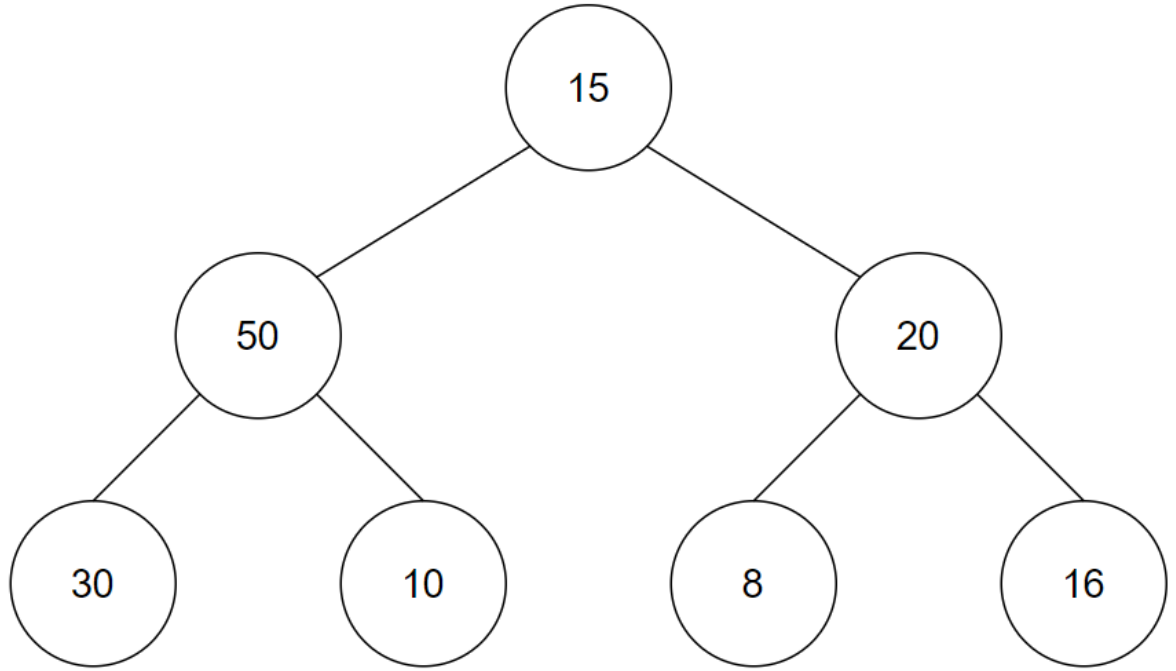
Deleting from a Max Heap

Now let's fix our heap.

Swap 15 and 50

Swap 15 and 30

Are we OK now?

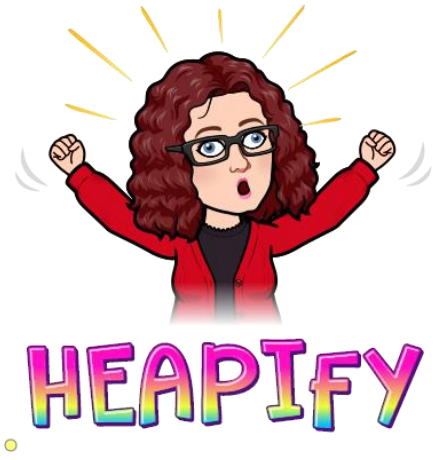


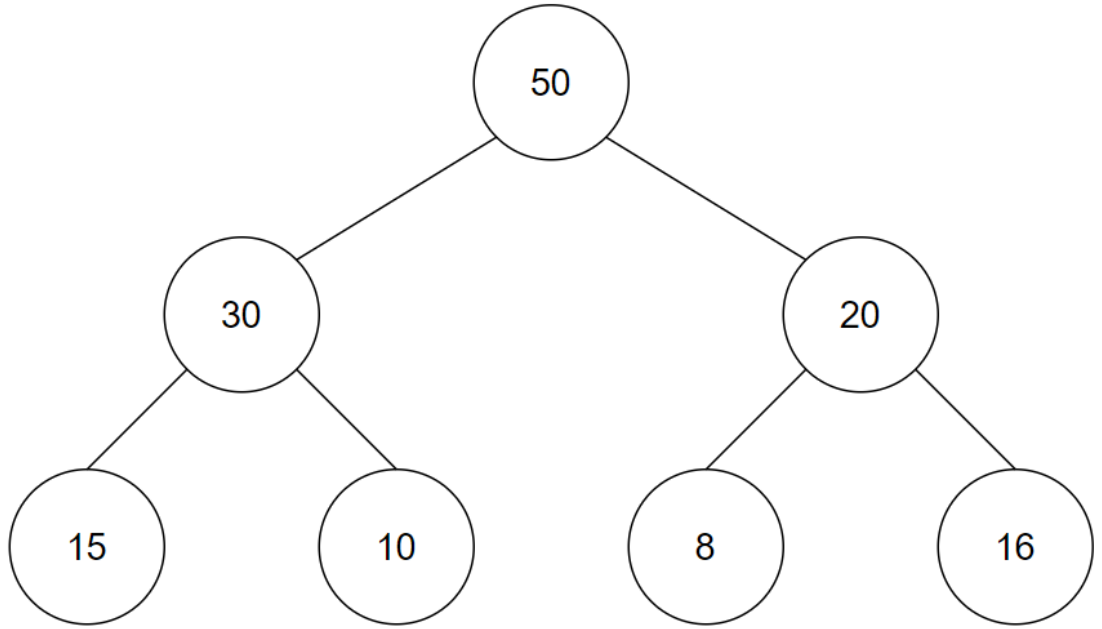
0	1	2	3	4	5	6	7
15	50	20	30	10	8	16	

Deleting from a Max Heap

Yes!

Our tree is complete and we have maintained our heap property.





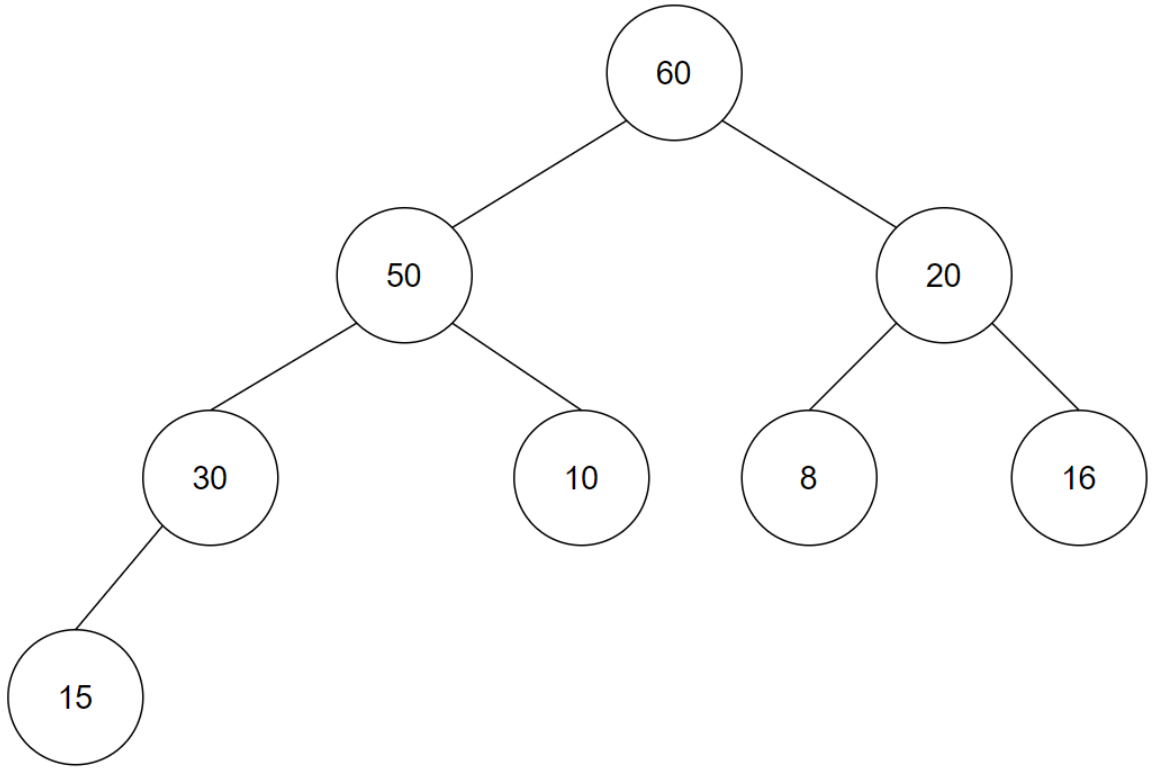
0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	

Sorting Using a Max Heap

What happens if we keep deleting the root from the heap?

When we delete the root, we are just taking it out of index 0 – we are not removing array element 0 – doing so just allows us to put the last element of the array in index 0.

What if we stash that 60 in that spot we just emptied?



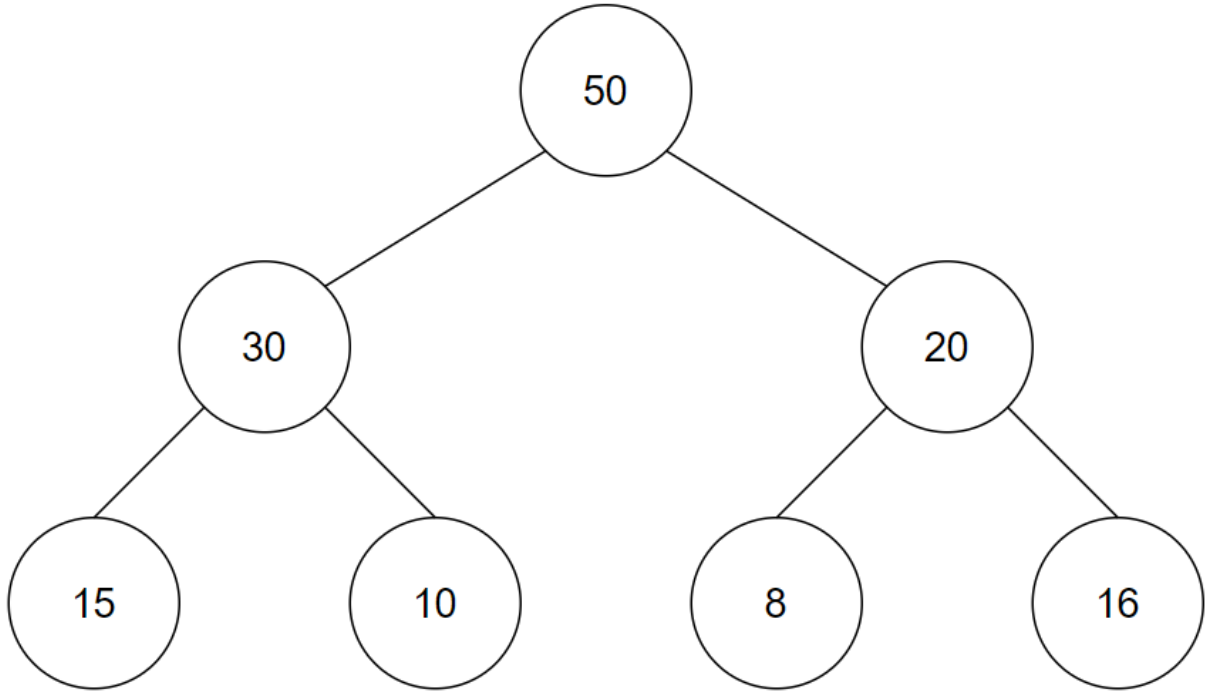
0	1	2	3	4	5	6	7
60	50	20	30	10	8	16	15

Sorting Using a Max Heap

So after heapifying, our heap is back to a good state and we make note that our array ends at element 6.

We are just using the space in element 7 to store that 60.

So what happens if we repeat this process?



0	1	2	3	4	5	6	7
50	30	20	15	10	8	16	60

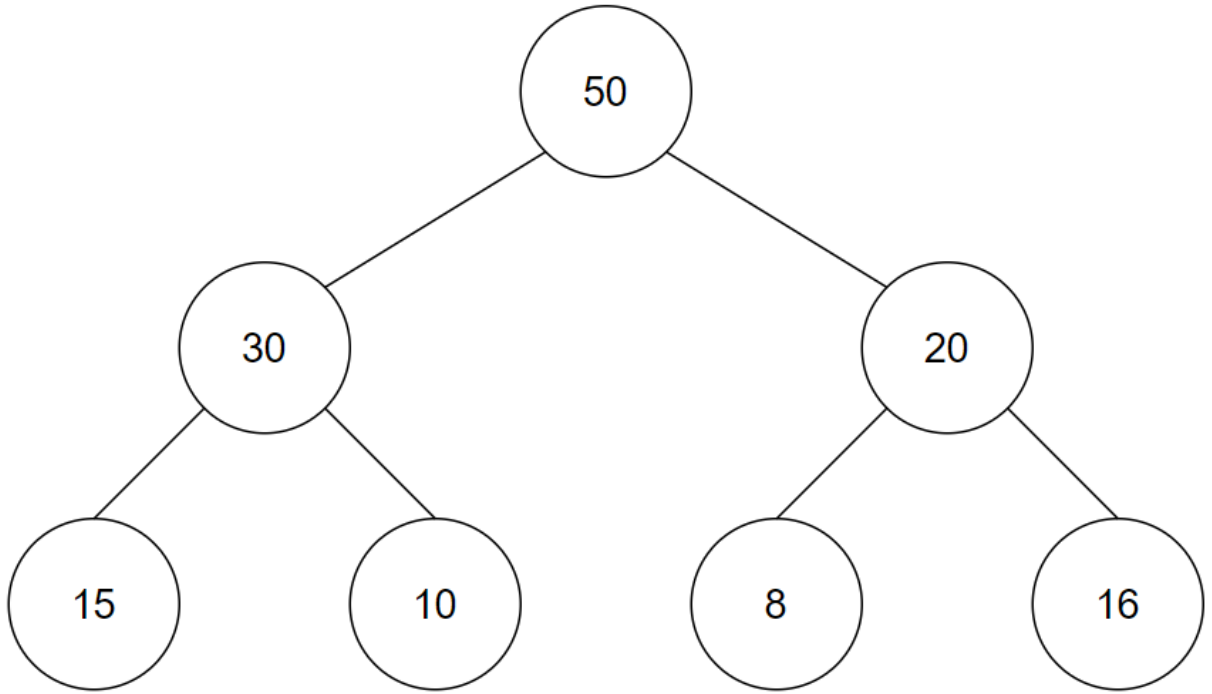
Sorting Using a Max Heap

We swap the 50 and the 16.

Heapify time!!

We make note that the end of our array is now element 5.

This also means that we only have 6 elements in our heap.

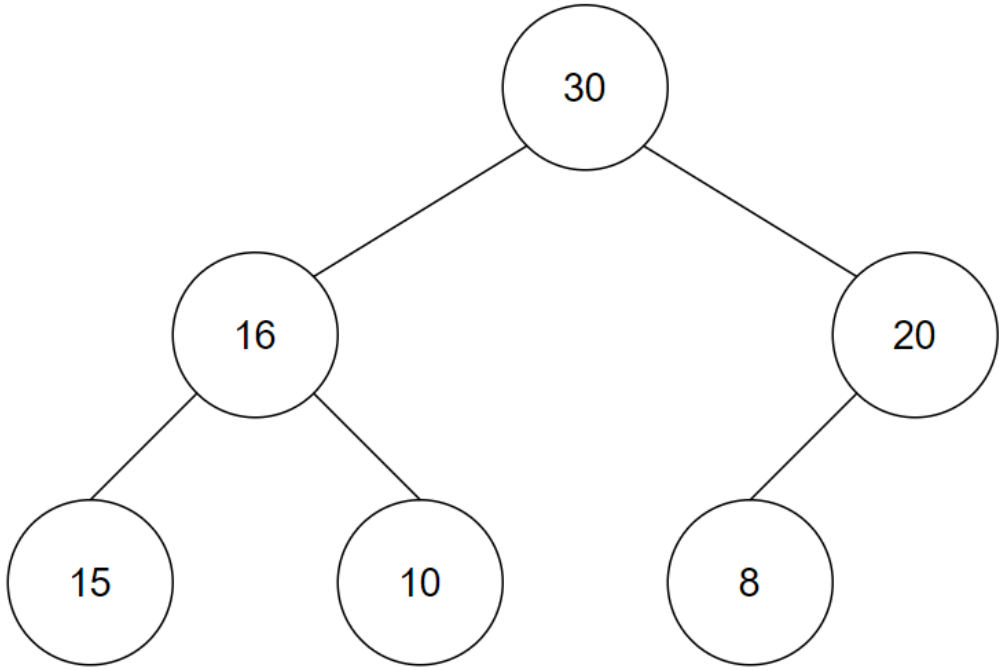


0	1	2	3	4	5	6	7
16	30	20	15	10	8	50	60

Sorting Using a Max Heap

Repeat.

Swap 30 and 8 and heapify.



0	1	2	3	4	5	6	7
30	16	20	15	10	8	50	60

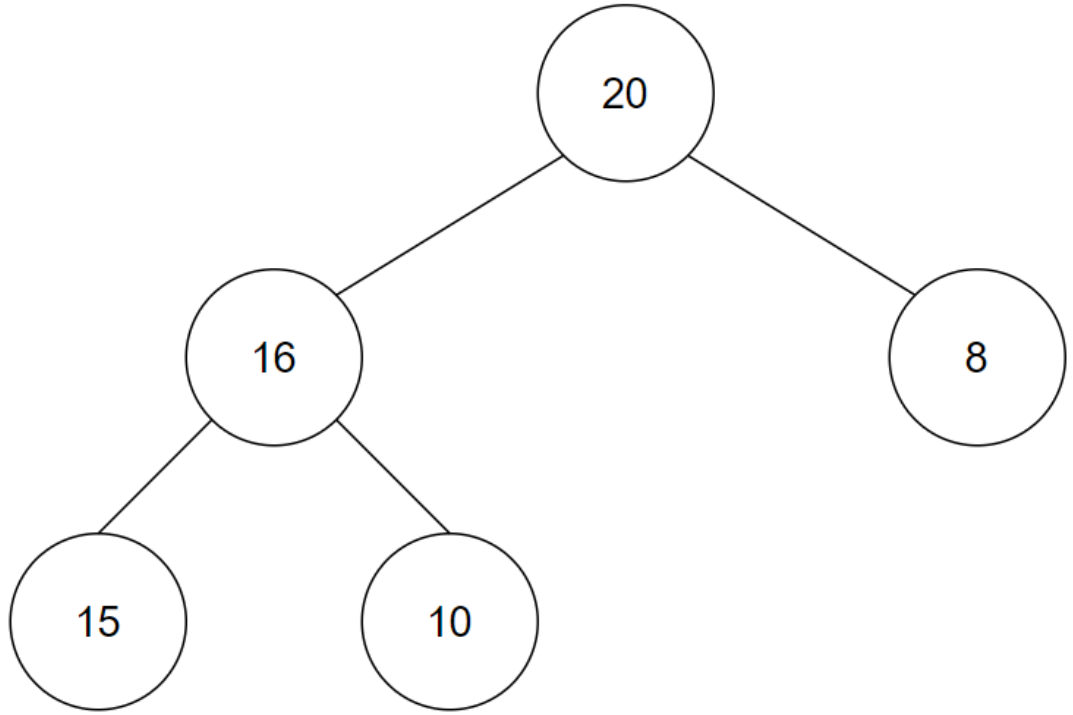
Sorting Using a Max Heap

Repeat.

Swap 20 and 10

Why not swap 20 and 8?

Heapify.



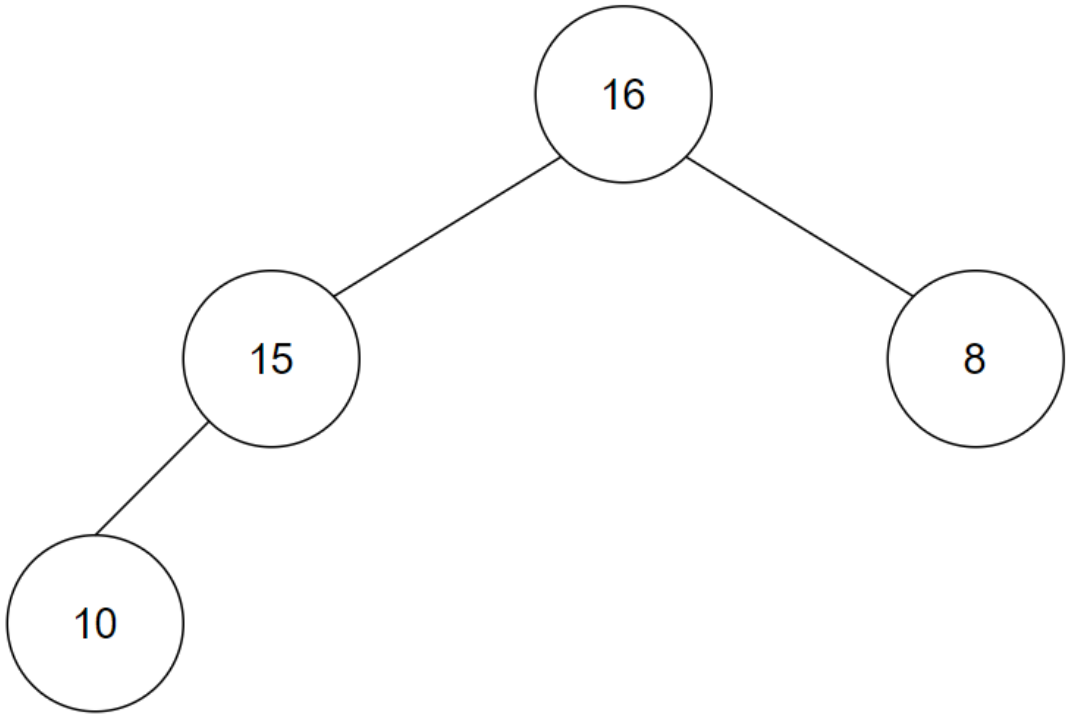
0	1	2	3	4	5	6	7
20	16	8	15	10	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 16 and 10.

Heapify.



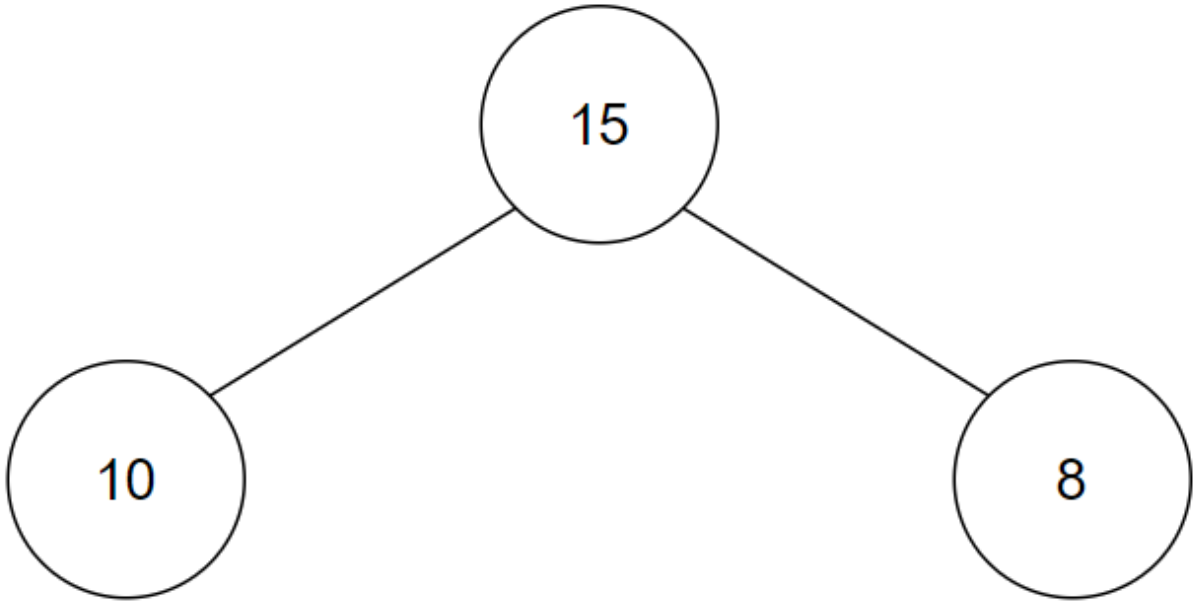
0	1	2	3	4	5	6	7
16	15	8	10	20	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 8 and 15.

Heapify



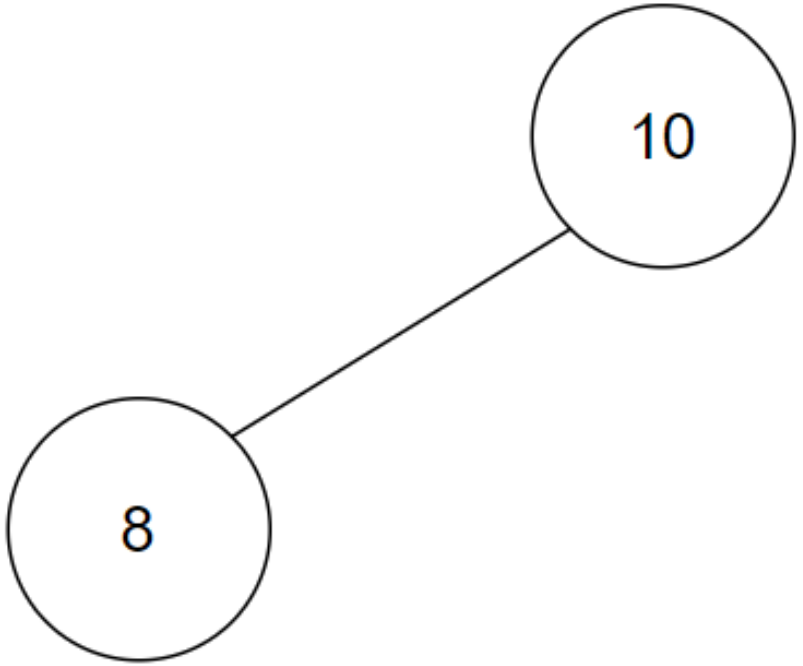
0	1	2	3	4	5	6	7
15	10	8	16	20	30	50	60

Sorting Using a Max Heap

Repeat.

Swap 10 and 8.

Do we need to heapify?

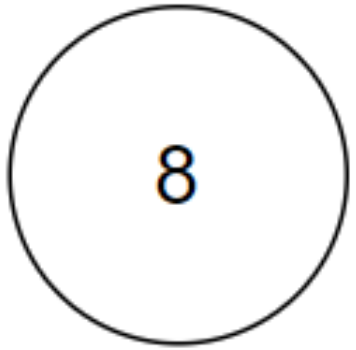


0	1	2	3	4	5	6	7
10	8	15	16	20	30	50	60

Sorting Using a Max Heap

No.

Final node maintains the heap property.



What is in our array now?

It is sorted.

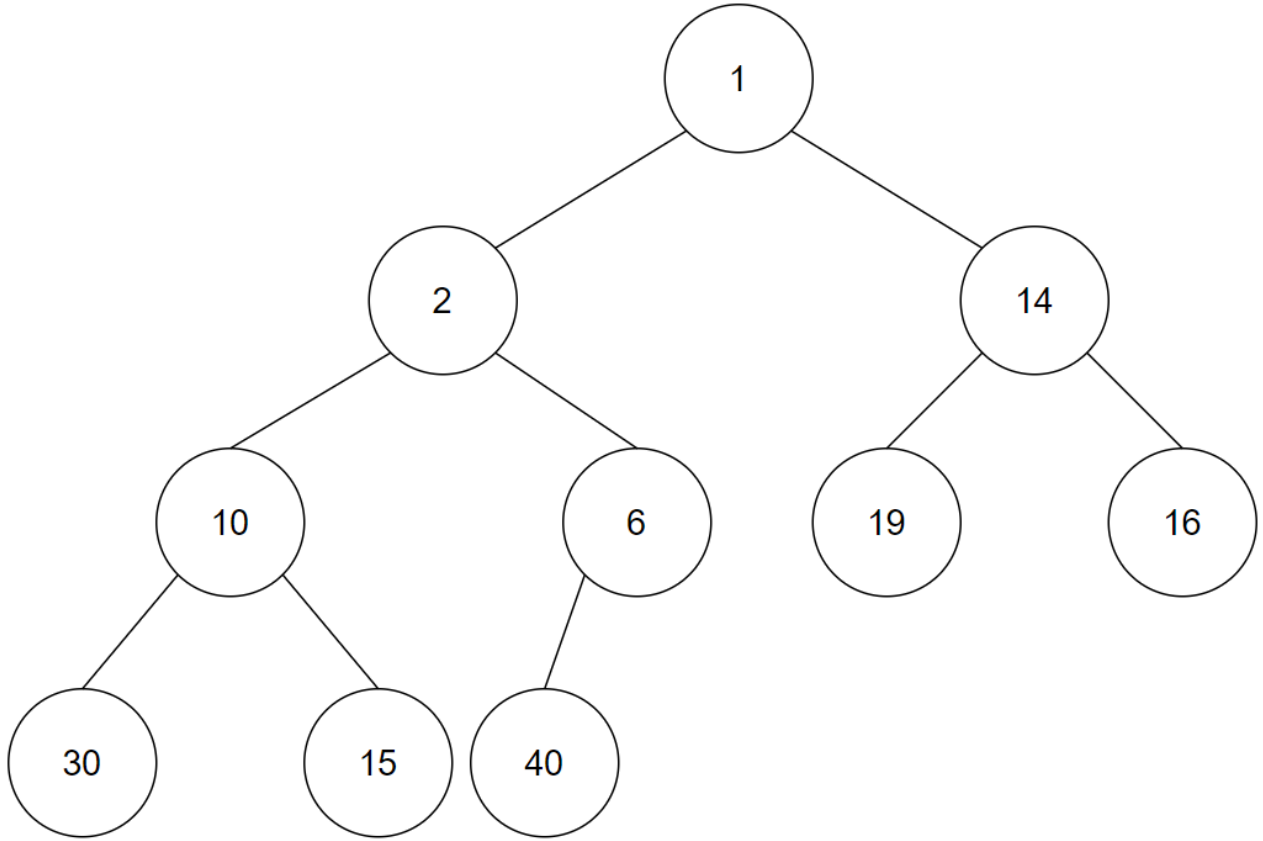
0	1	2	3	4	5	6	7
8	10	15	16	20	30	50	60

Sorting Using a Min Heap

Does the process work differently for a Min Heap?

Will the result be different?

Let's try it.

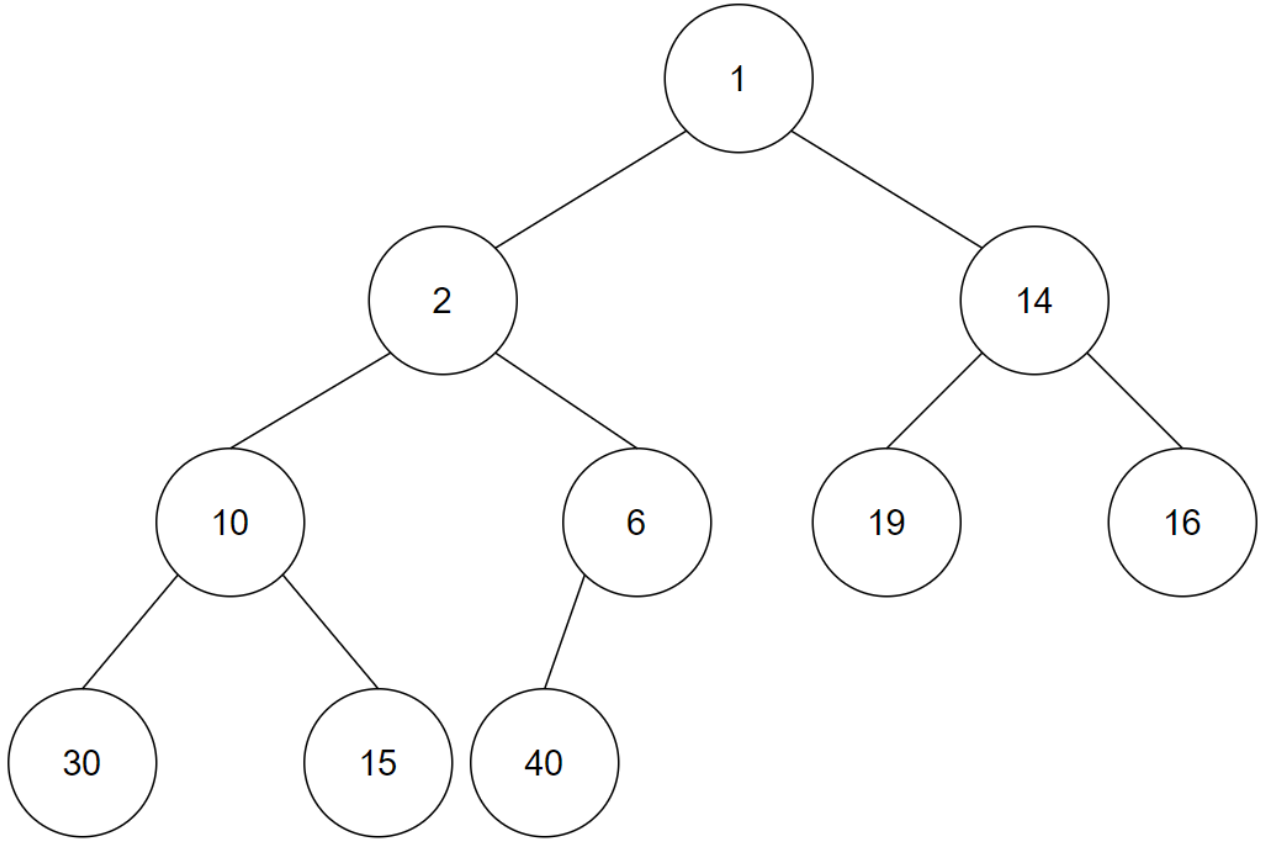


0	1	2	3	4	5	6	7	8	9
1	2	14	10	6	19	16	30	15	40

Sorting Using a Min Heap

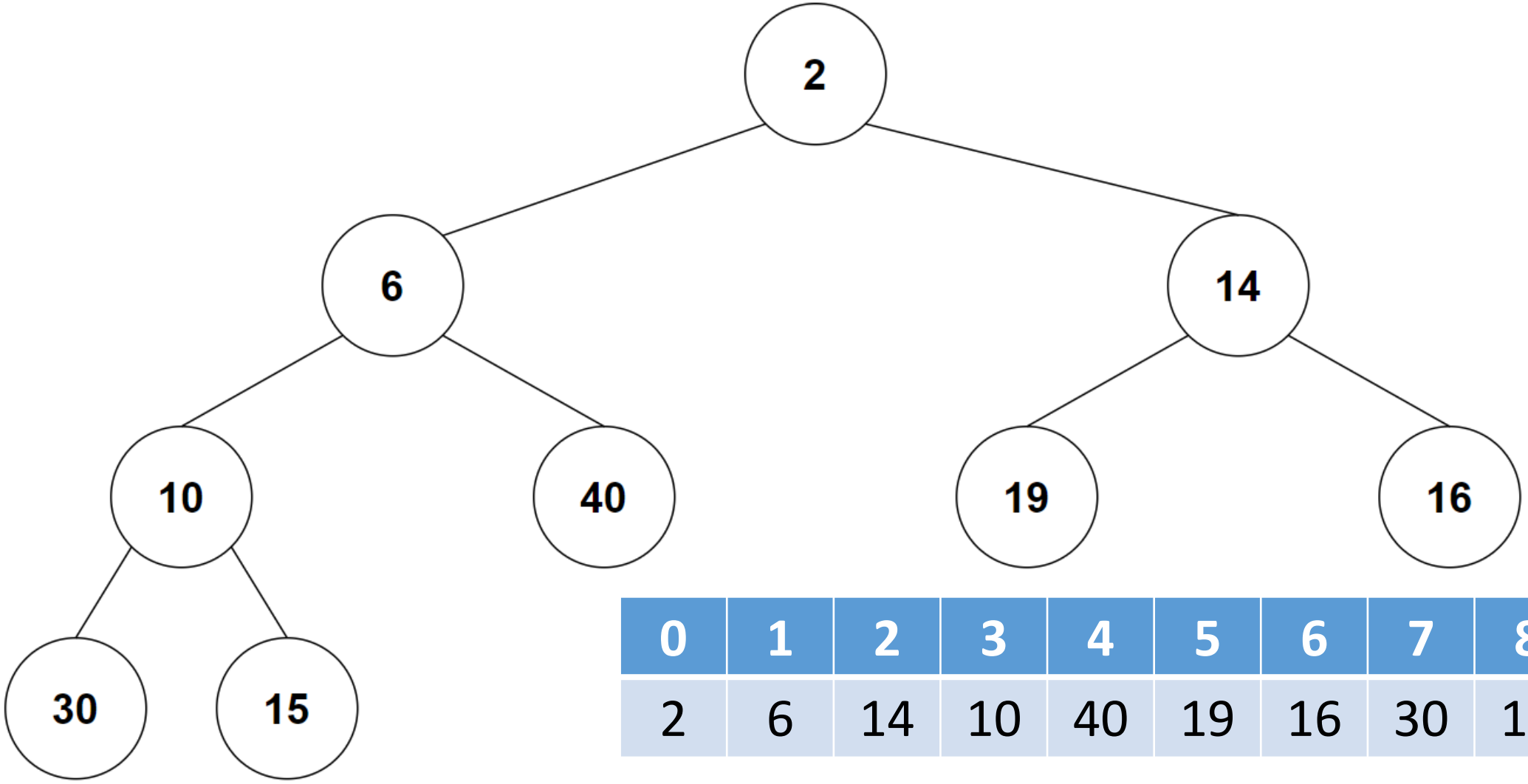
Swap 1 and 40

Heapify!



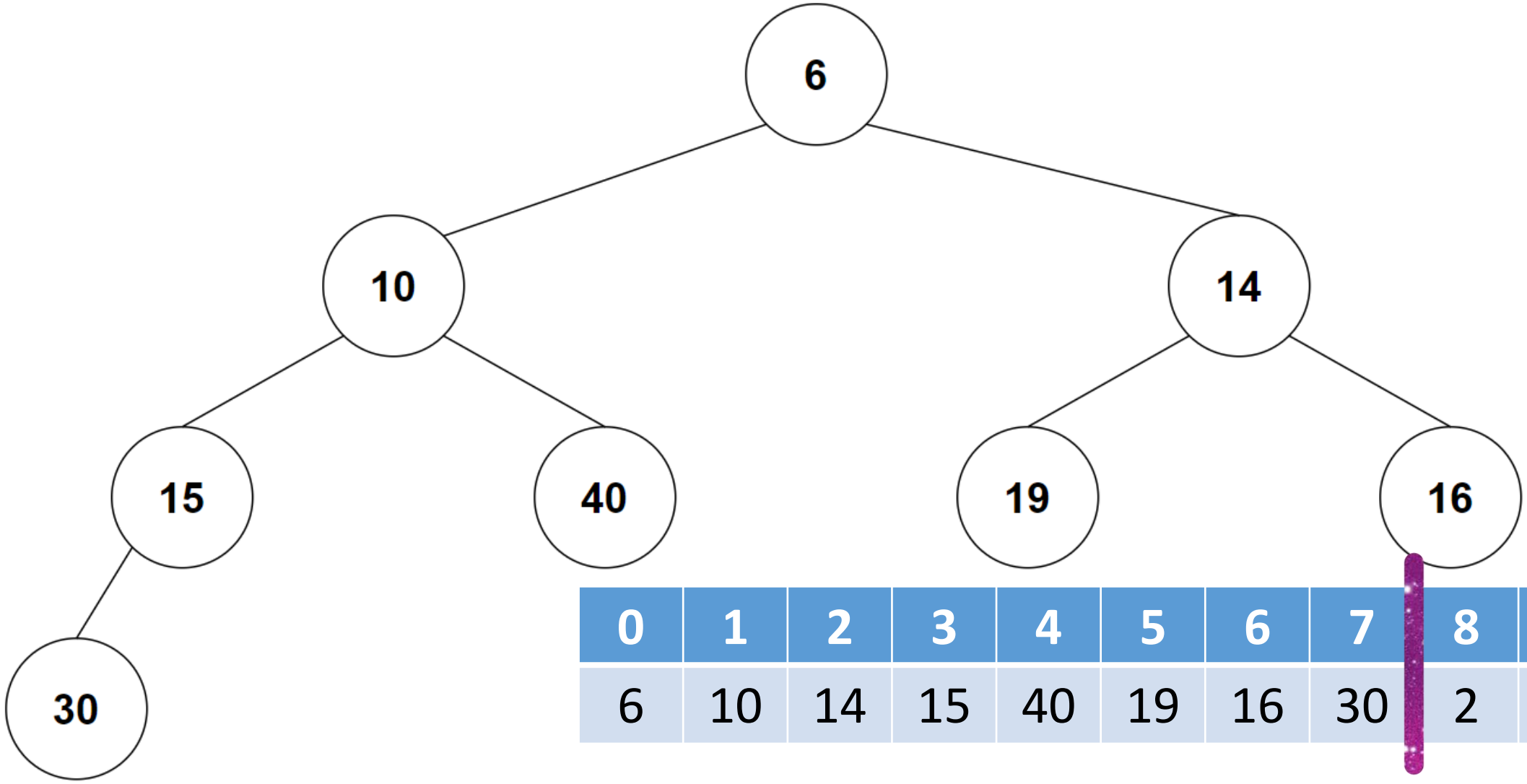
0	1	2	3	4	5	6	7	8	9
40	2	14	10	6	19	16	30	15	1

Sorting Using a Min Heap



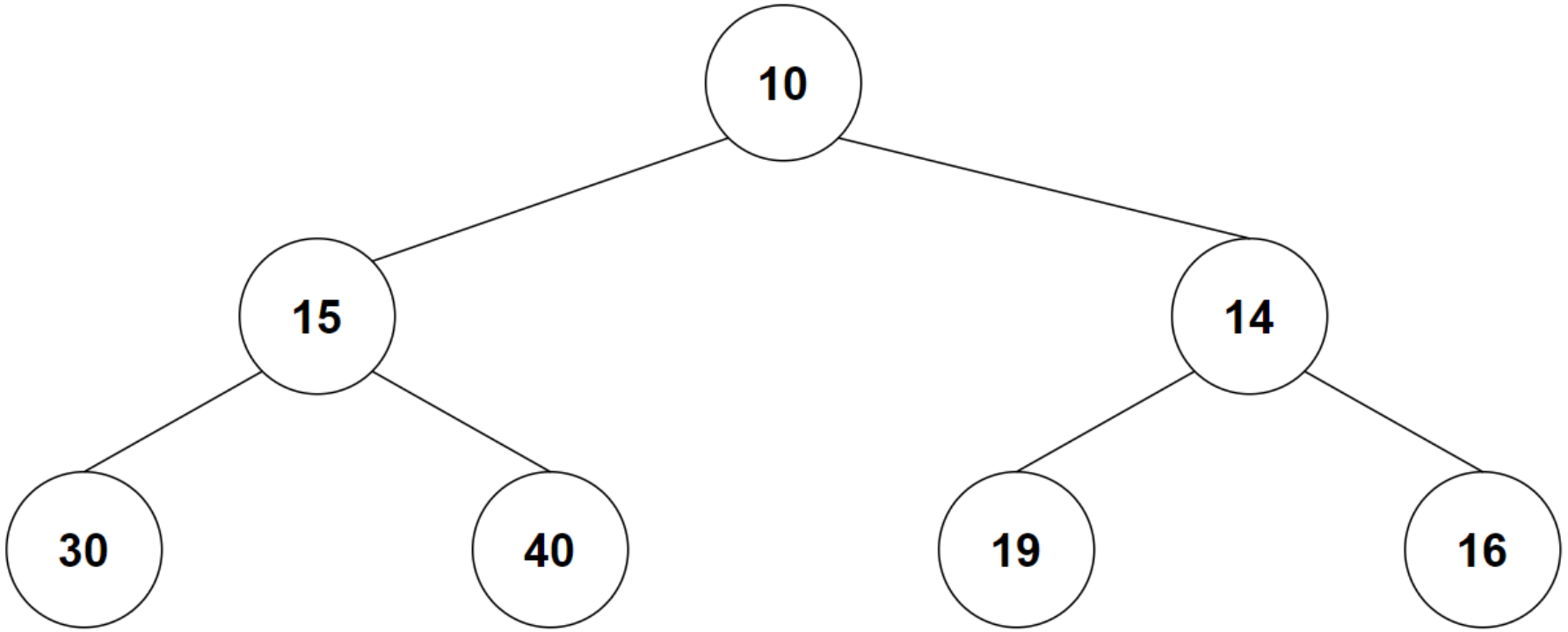
0	1	2	3	4	5	6	7	8	9
2	6	14	10	40	19	16	30	15	1

Sorting Using a Min Heap



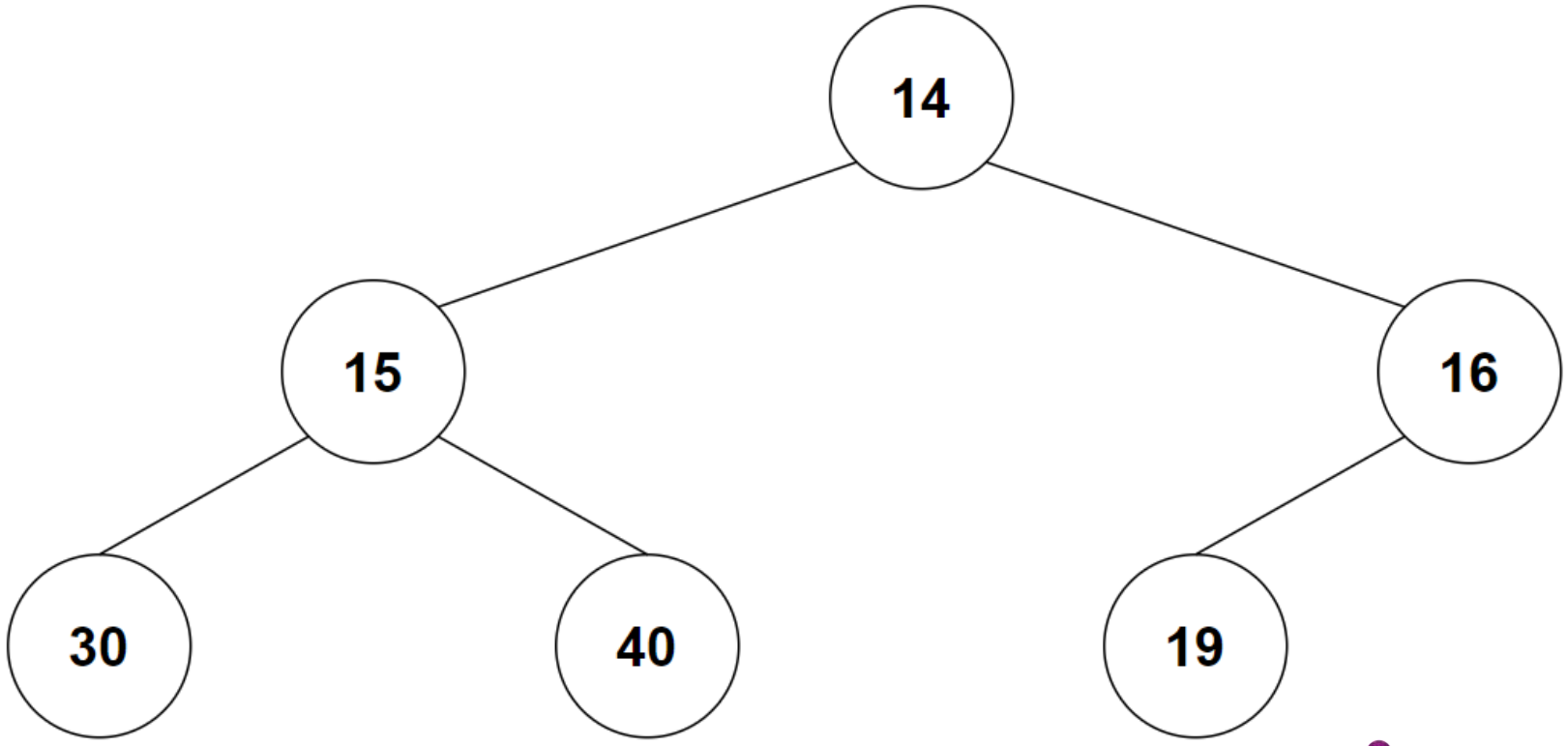
0	1	2	3	4	5	6	7	8	9
6	10	14	15	40	19	16	30	2	1

Sorting Using a Min Heap



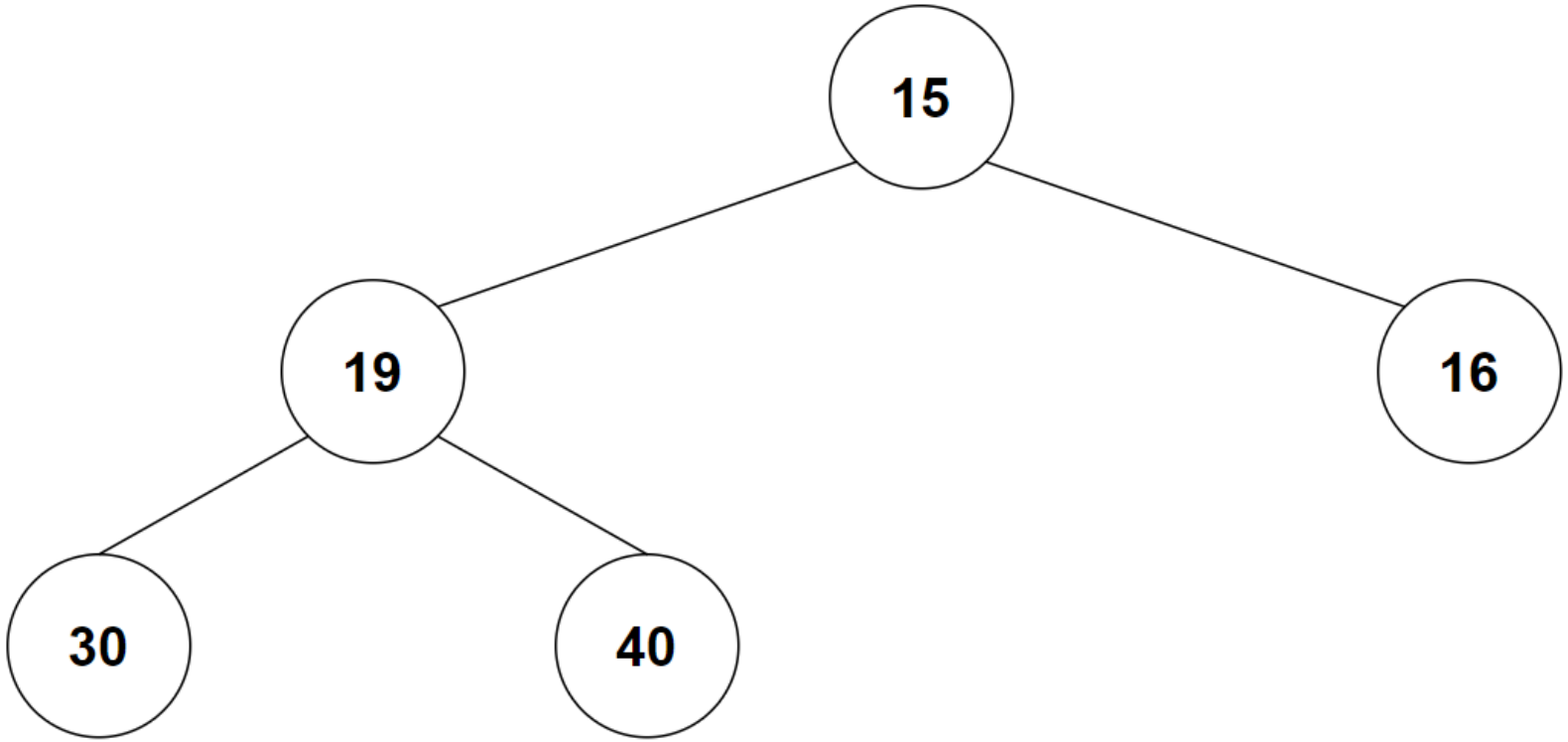
0	1	2	3	4	5	6	7	8	9
10	15	14	30	40	19	16	6	2	1

Sorting Using a Min Heap



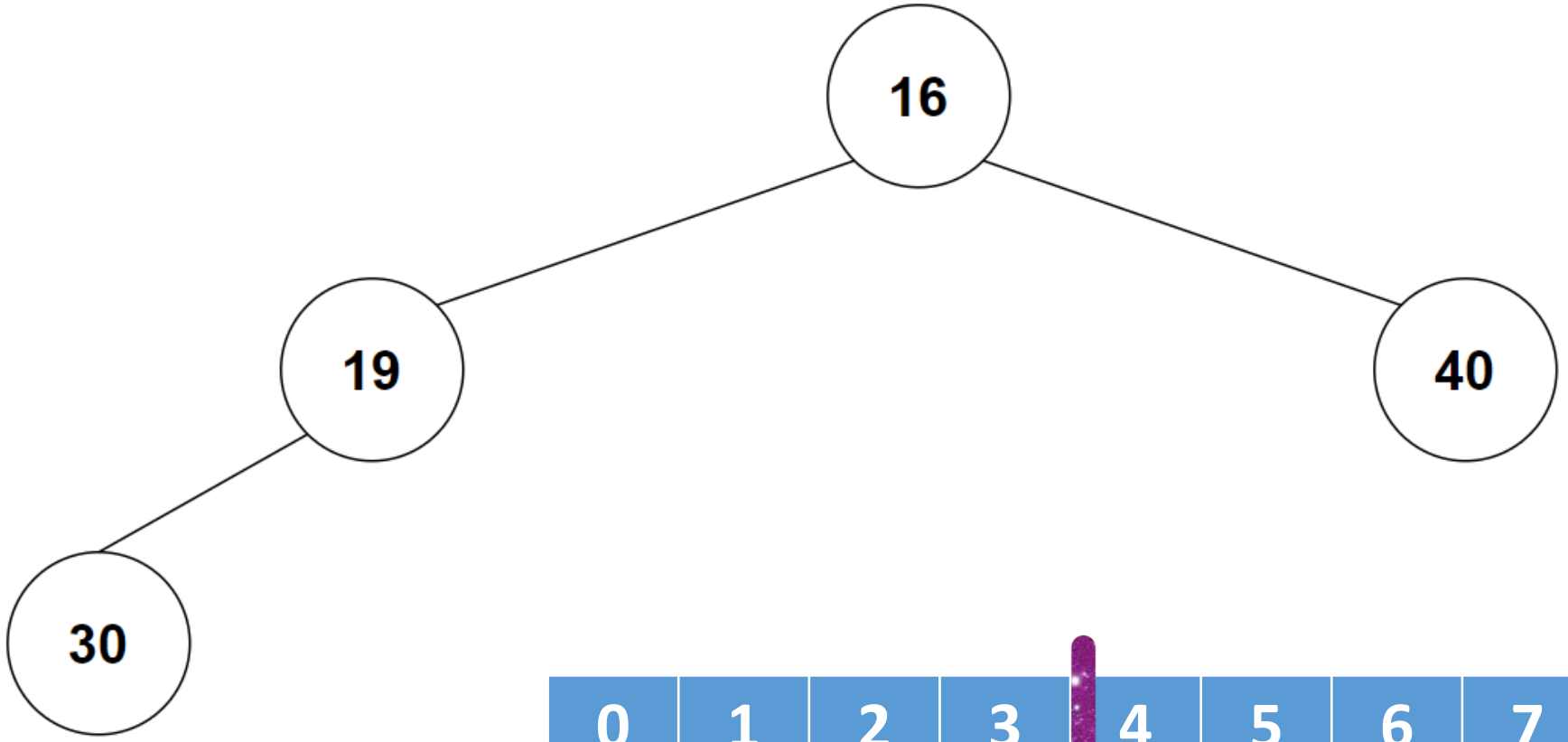
0	1	2	3	4	5	6	7	8	9
14	15	16	30	40	19	10	6	2	1

Sorting Using a Min Heap



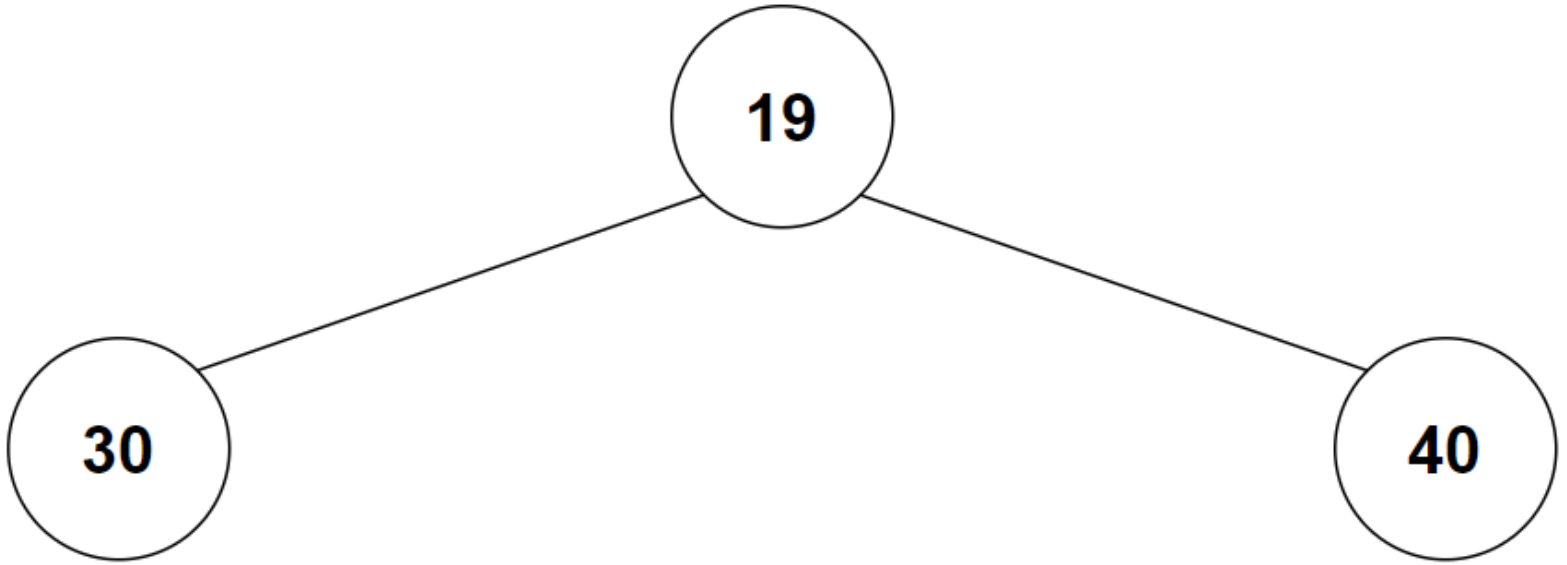
0	1	2	3	4	5	6	7	8	9
15	19	16	30	40	14	10	6	2	1

Sorting Using a Min Heap



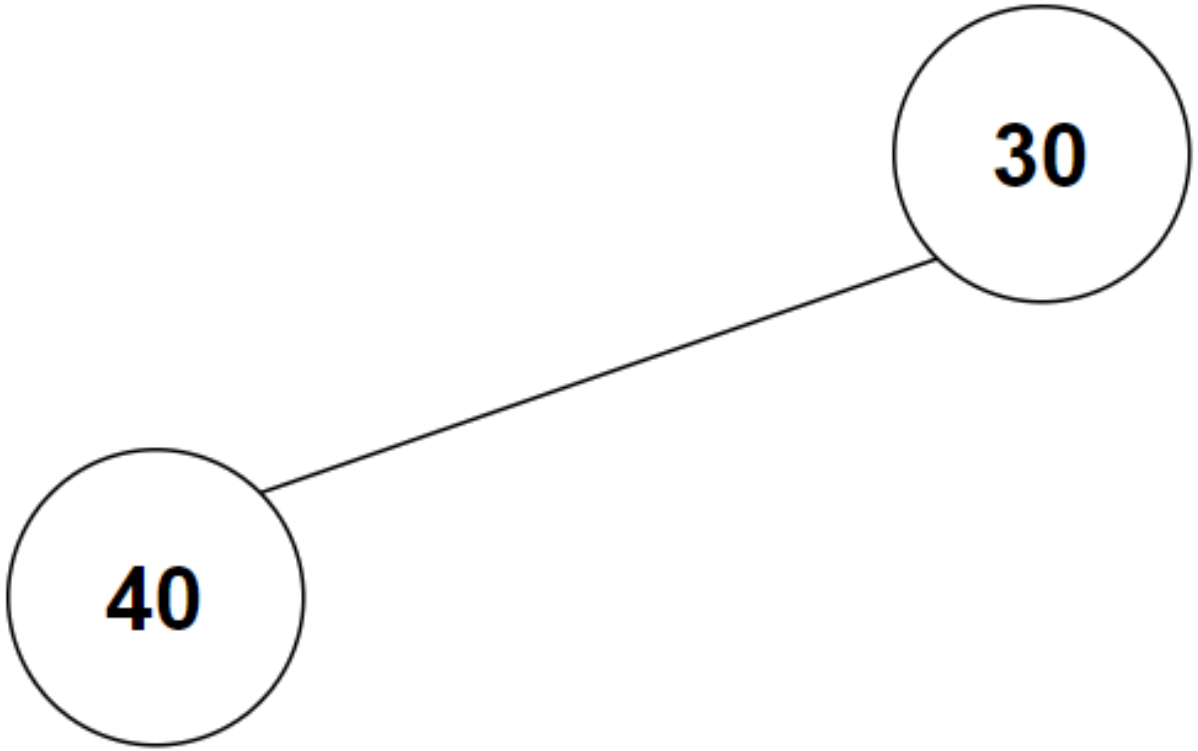
0	1	2	3	4	5	6	7	8	9
16	19	40	30	15	14	10	6	2	1

Sorting Using a Min Heap



0	1	2	3	4	5	6	7	8	9
19	30	40	16	15	14	10	6	2	1

Sorting Using a Min Heap



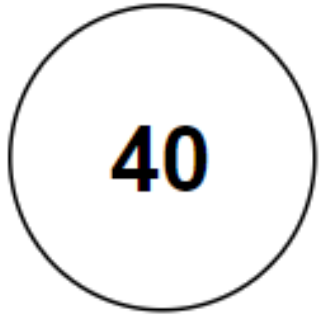
0	1	2	3	4	5	6	7	8	9
30	40	19	16	15	14	10	6	2	1

Sorting Using a Min Heap

Summary

Binary Max Heap sorted the list ascending order.

Binary Min Heap sorted the list in descending order.

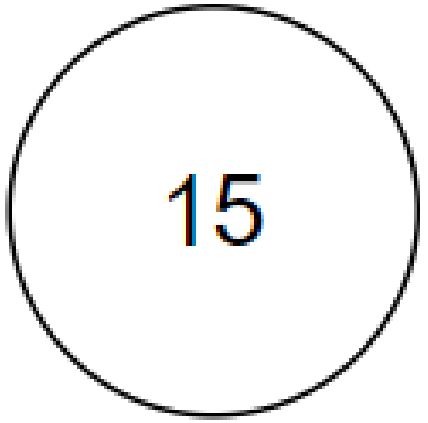


0	1	2	3	4	5	6	7	8	9
40	30	19	16	15	14	10	6	2	1

Creating a Max Heap

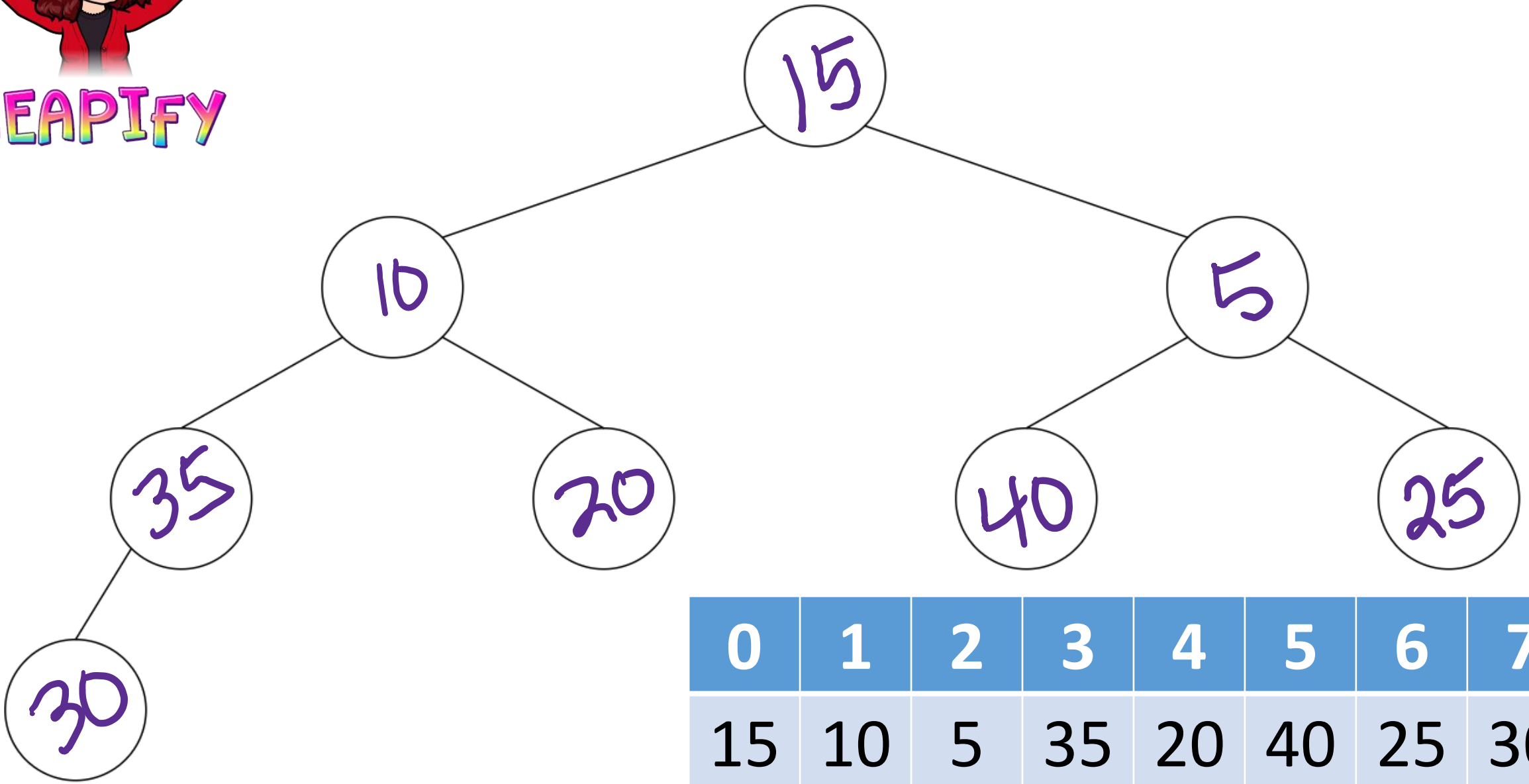
Given this array, create a max heap

0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30



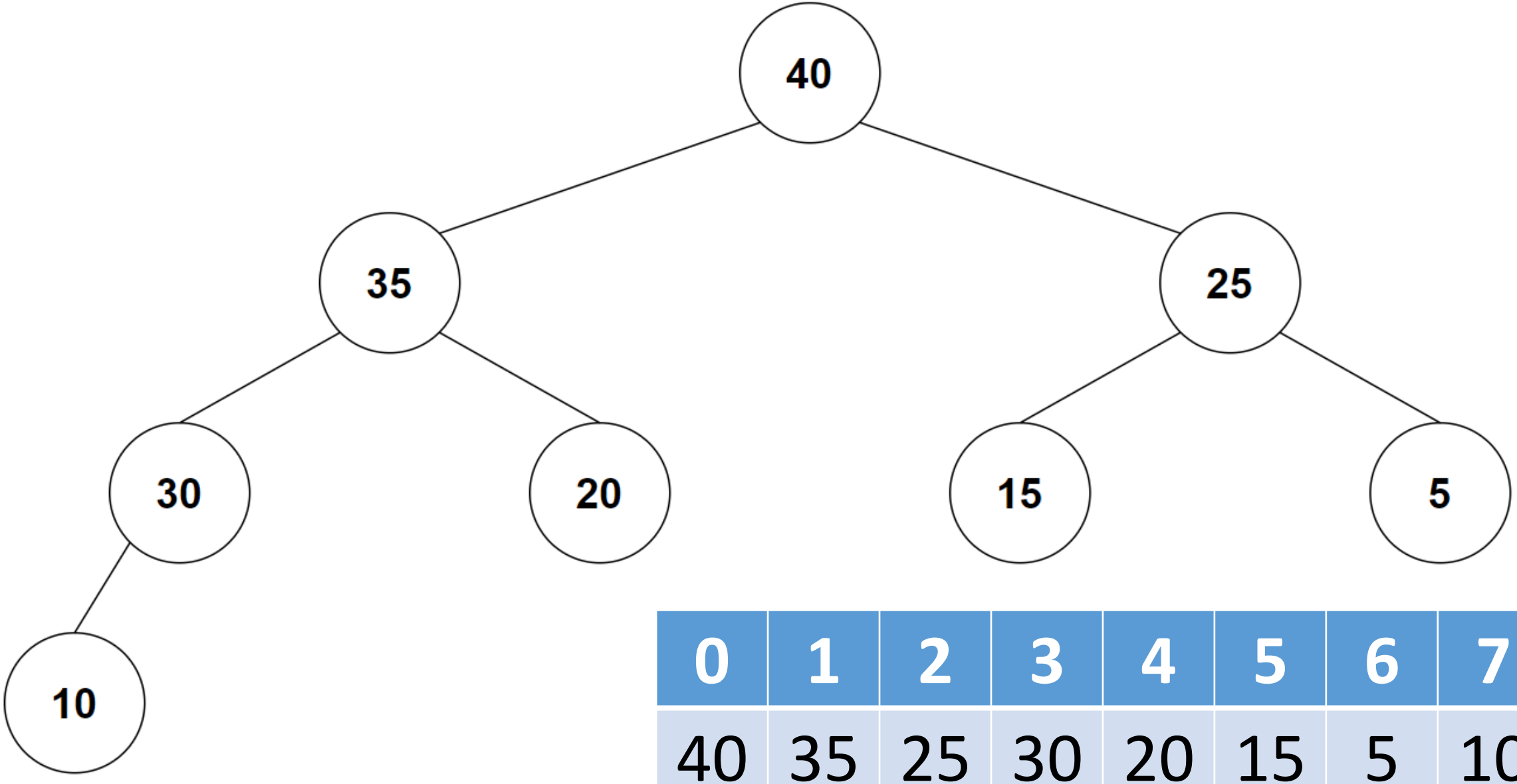
Take the 0^{th} element and make it the root.

Creating a Max Heap



0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30

Creating a Max Heap

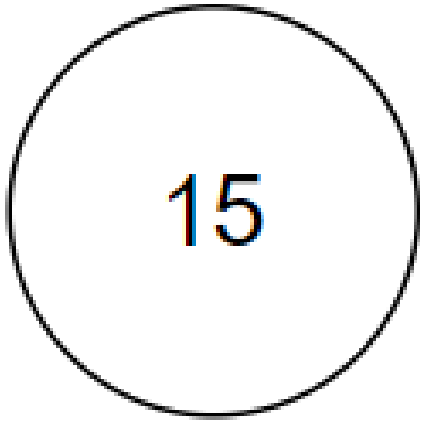


0	1	2	3	4	5	6	7
40	35	25	30	20	15	5	10

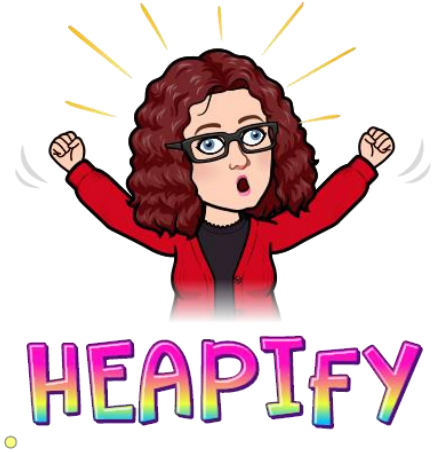
Creating a Min Heap

Given this array, create a min heap

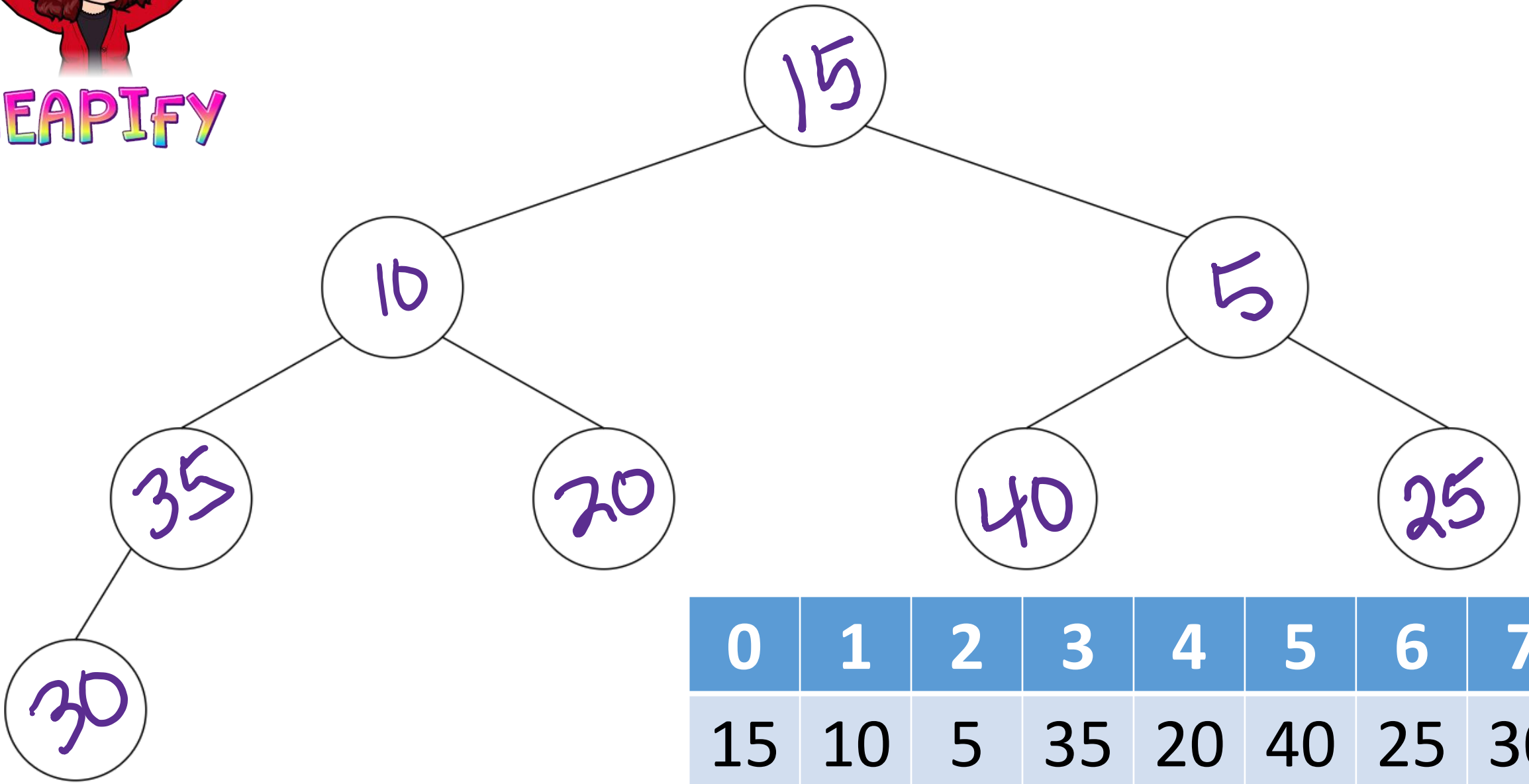
0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30



Take the 0^{th} element and make it the root.

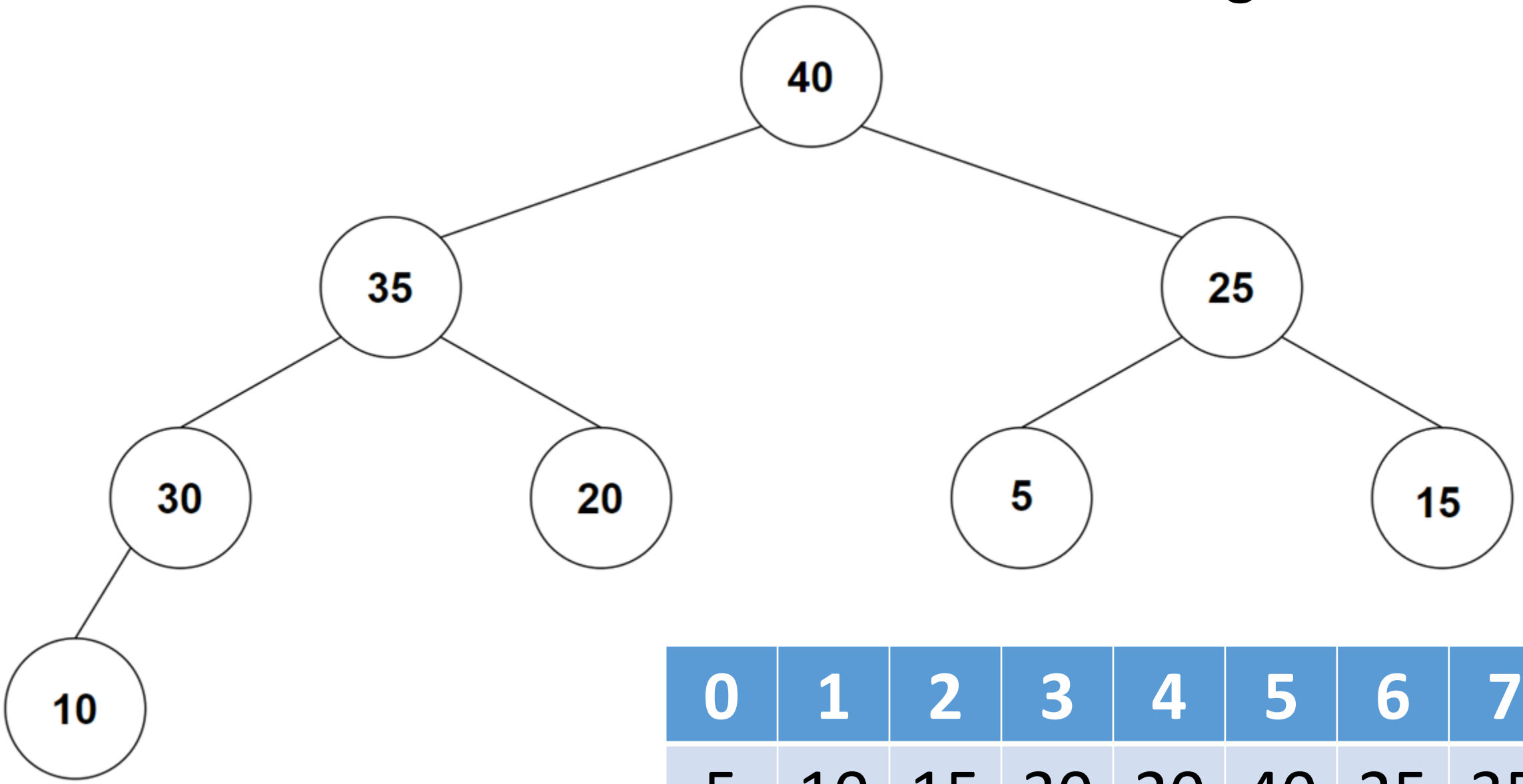


Creating a Min Heap



0	1	2	3	4	5	6	7
15	10	5	35	20	40	25	30

Creating a Min Heap



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Time Complexity

What is the time complexity of building a binary heap?

How many elements did we insert?

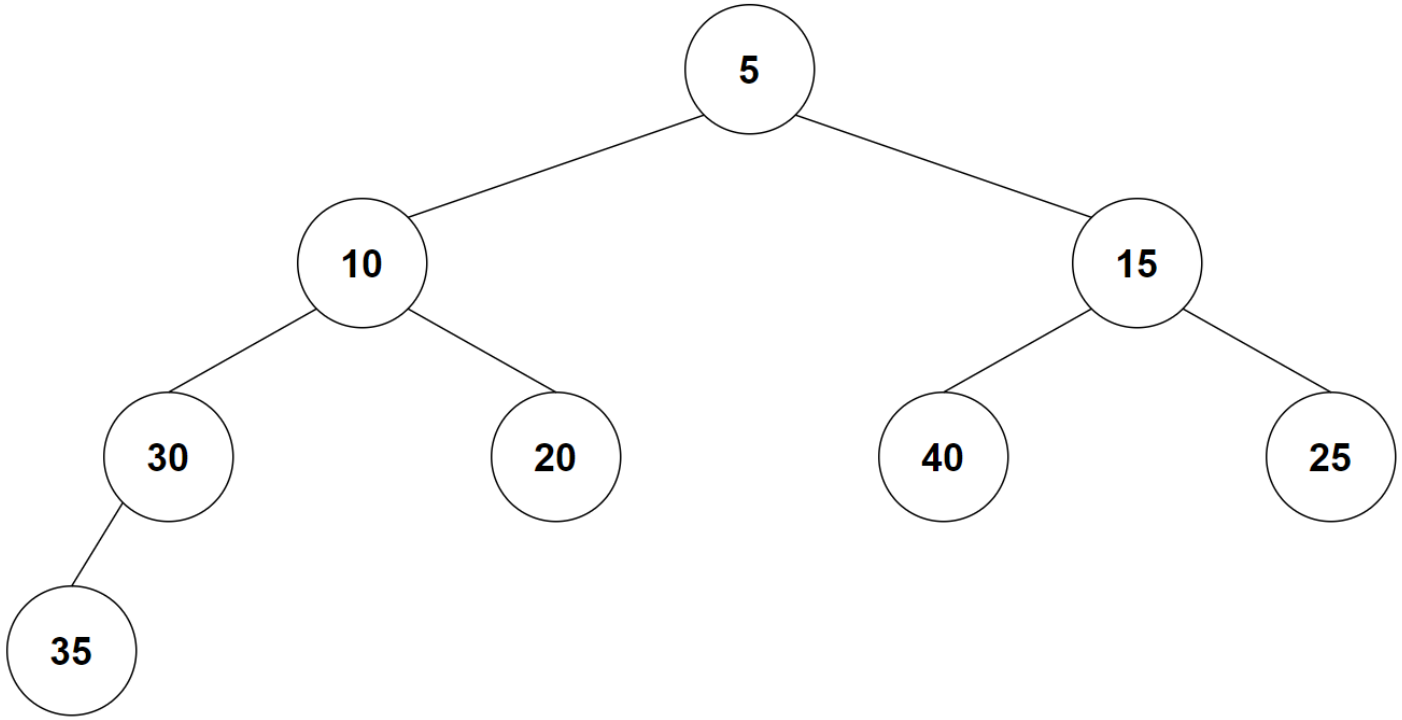
n

The amount of time needed to insert an element into a heap depends on the height of the complete binary tree.

$\log_2 n$

So for worst case, we assume that every element is moved up/down the full height of the tree.

$n\log_2 n$



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Time Complexity

What is the time complexity of deleting a binary heap?

How many elements did we delete?

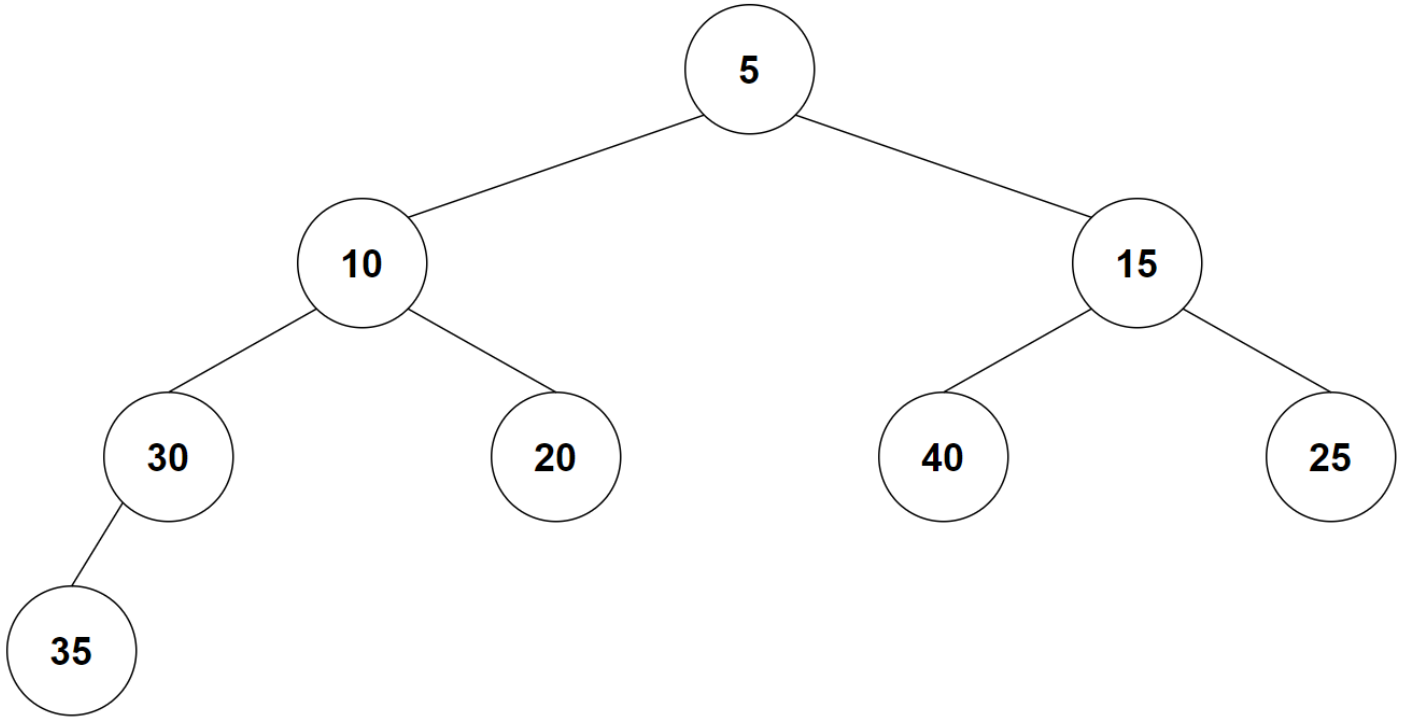
n

The amount of time needed to delete an element into a heap depends on the height of the complete binary tree.

$\log_2 n$

So for worst case, we assume that every element is moved up/down the full height of the tree.

$n\log_2 n$



0	1	2	3	4	5	6	7
5	10	15	30	20	40	25	35

Time Complexity

What is the time complexity of sorting using a binary heap?

We sorted by building a binary heap with the data and then deleting it to get the sort.

Building	+	Deleting	=	Sorting
$n\log_2 n$	+	$n\log_2 n$	=	$2*n\log_2 n$

$2n\log_2 n = O(n\log_2 n)$ for Heap Sort

Priority Queue

Making a priority queue

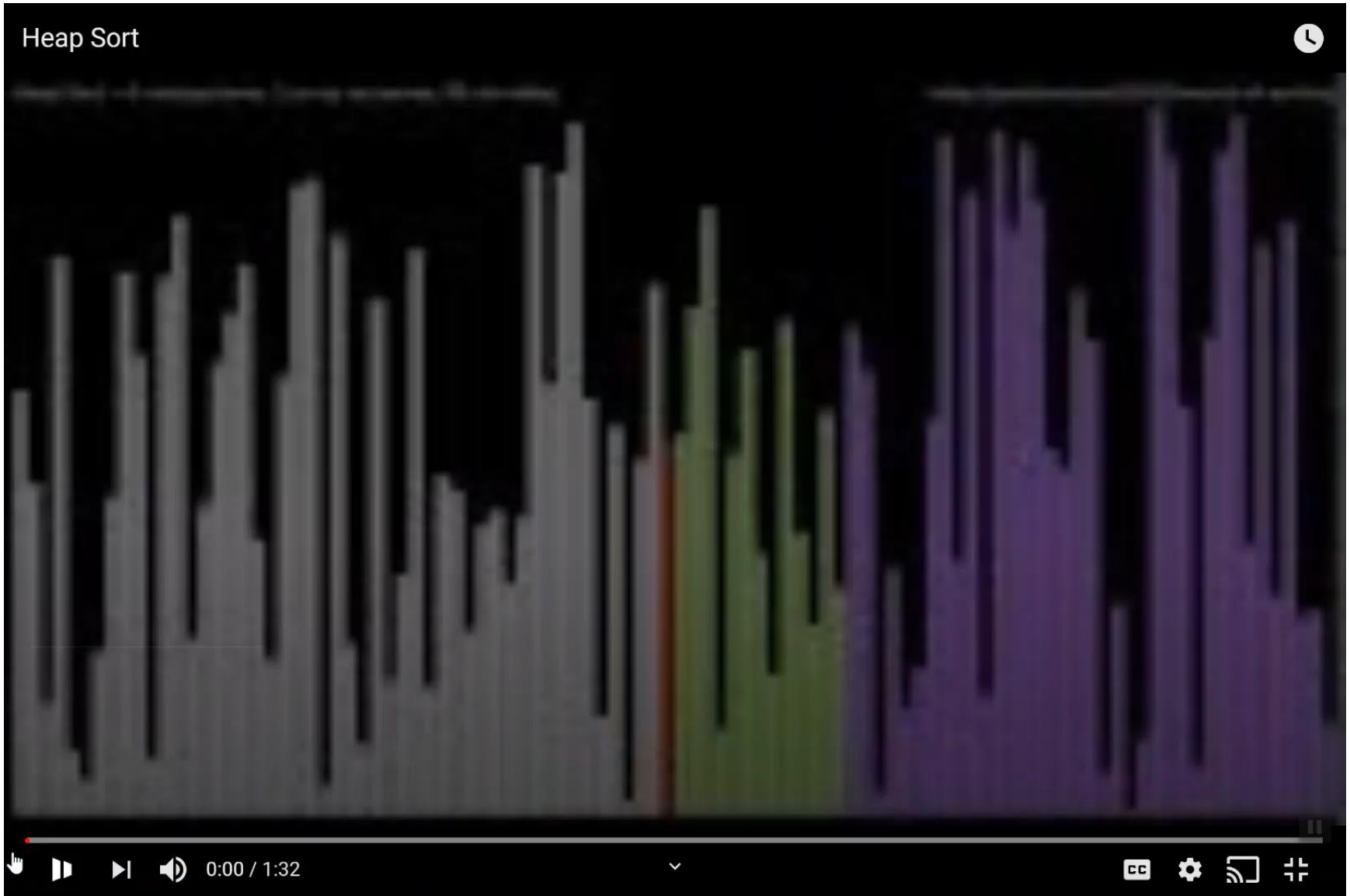
Use a heap structure

Insert new elements into the heap

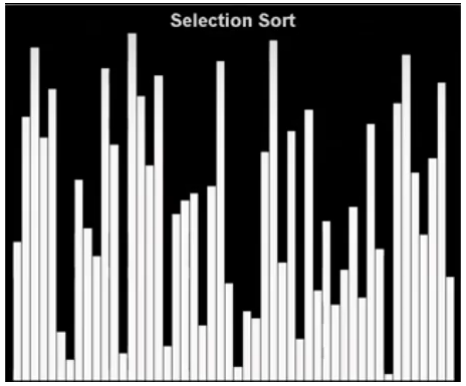
Remove the top element to get the highest priority element

Change priorities by removing the element and then re-inserting it

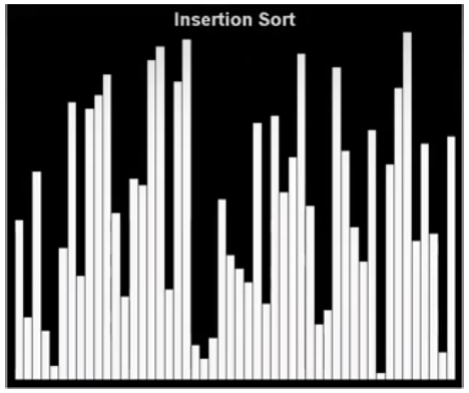
Visual Heap Sort



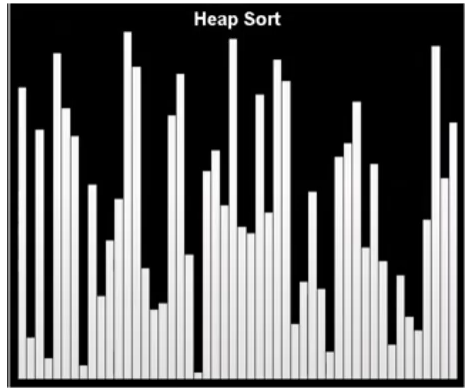
Sorts



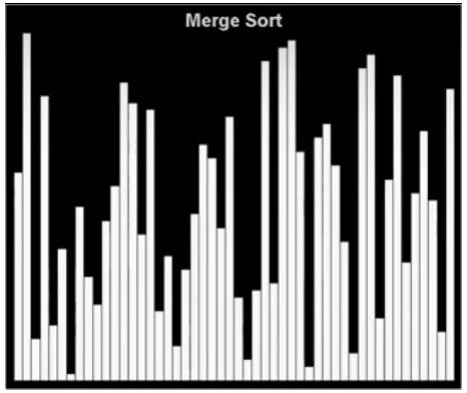
$O(n^2)$



$\Omega(n)$ $O(n^2)$

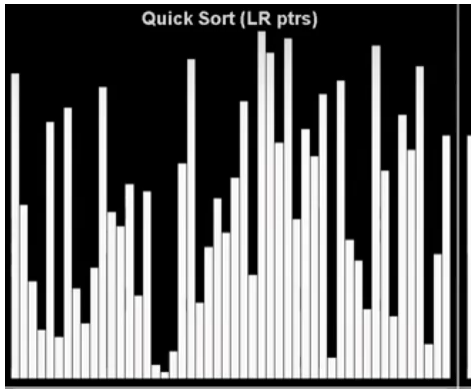


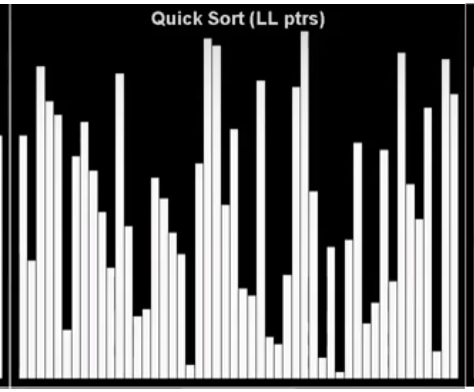
$O(n\log_2 n)$

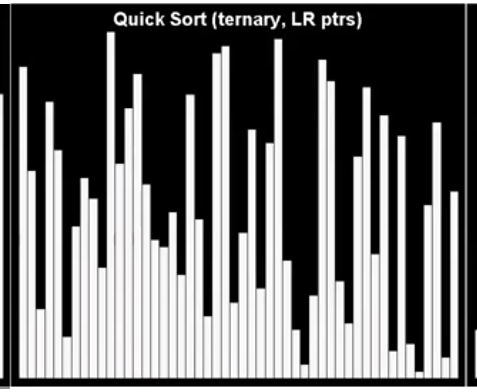


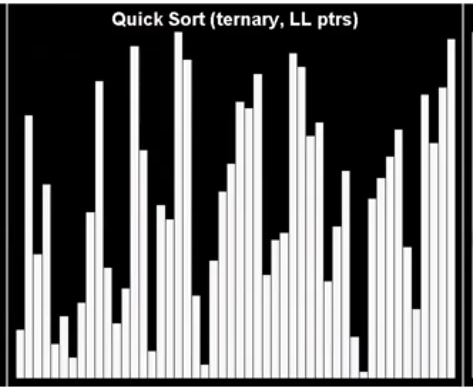
$O(n\log_2 n)$

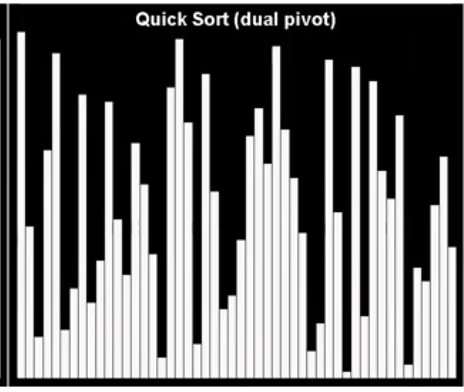
$\Omega(n\log_2 n)$
$\Theta(n\log_2 n)$
$O(n^2)$

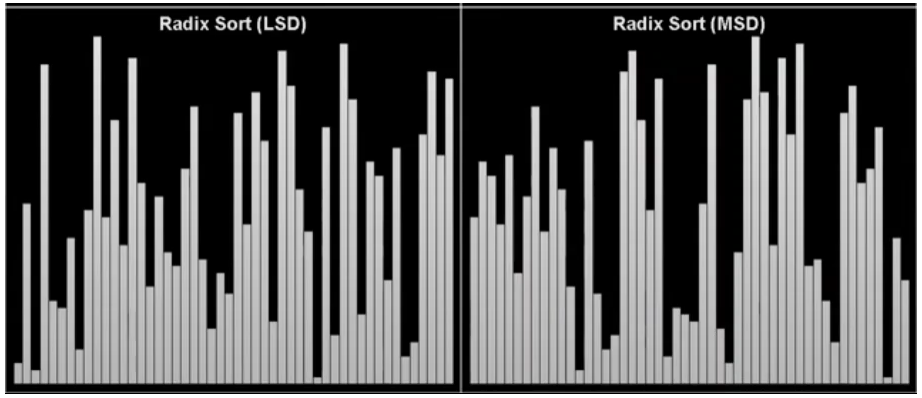












Radix Sort

Let's start with a list of numbers to sort

645, 86, 182, 35, 601, 8

Step 1

All numbers to be sorted need to be the same length.

Radix Sort

645, 86, 182, 35, 601, 8

How do we make 8 and 35 and 86 the same length as 648, 182 and 608?

645, 086, 182, 035, 601, 008

We add leading zeros/front pad our numbers with zeroes.

No difference in value between 001 and 1.

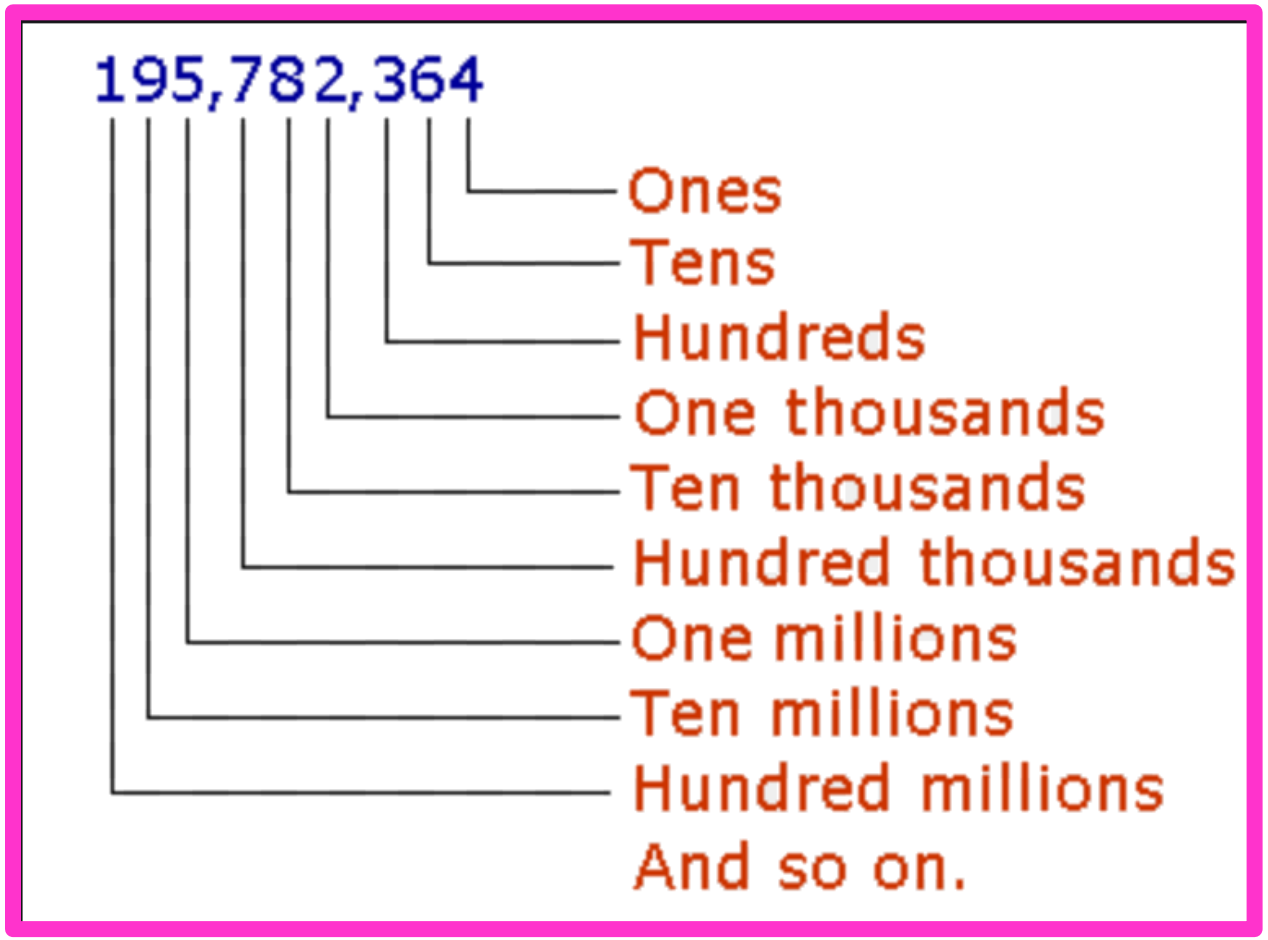
Big difference between 1 and 100 – add leading zeros to not change value.

Radix Sort

645, 086, 182, 035, 601, 008

Now we arrange our list by place value starting with ones.

601, 182, 645, 035, 086, 008

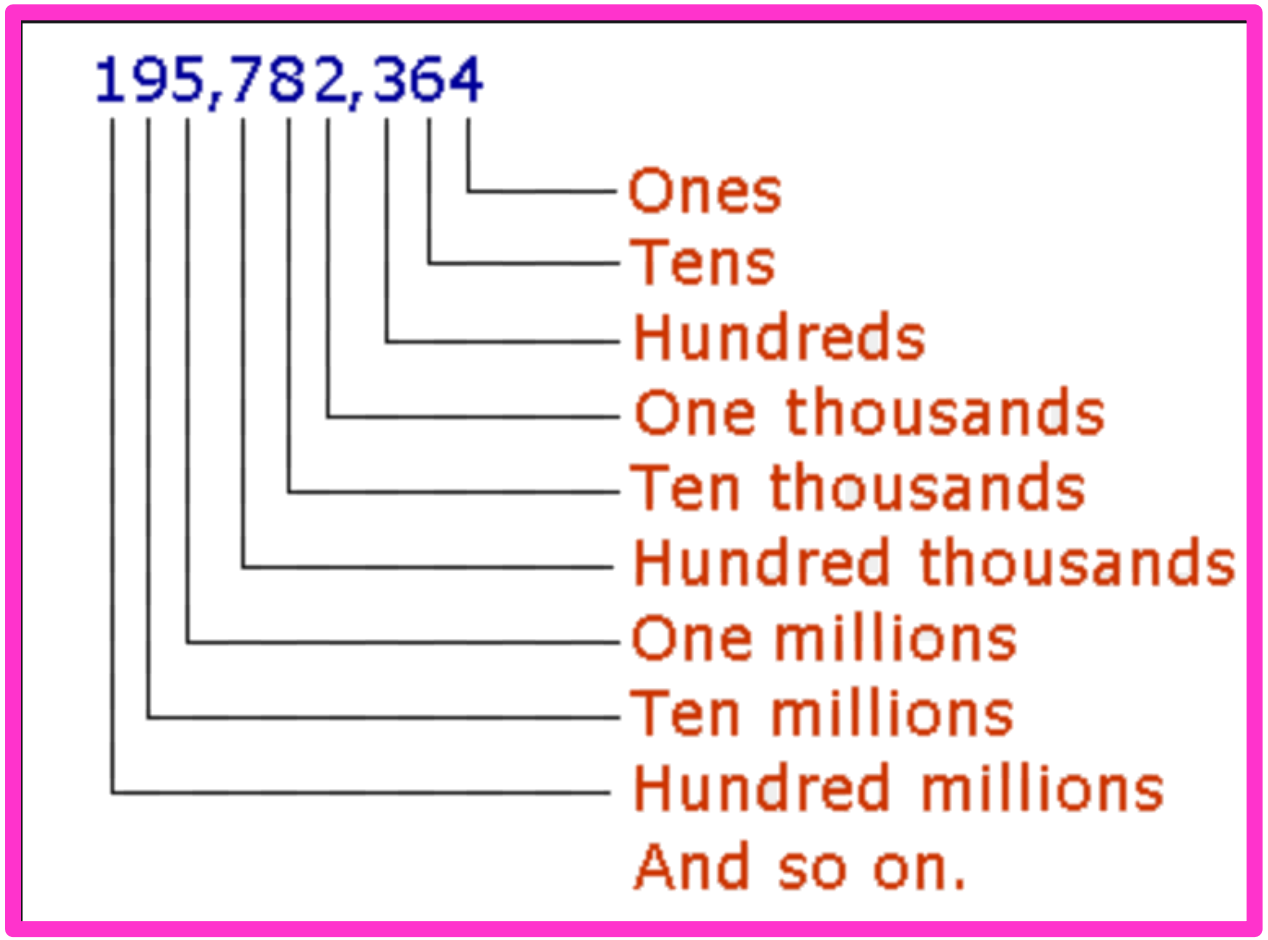


Radix Sort

601, 182, 645, 035, 086, 008

Now we arrange our list by place value tens.

601, 008, 035, 645, 182, 086

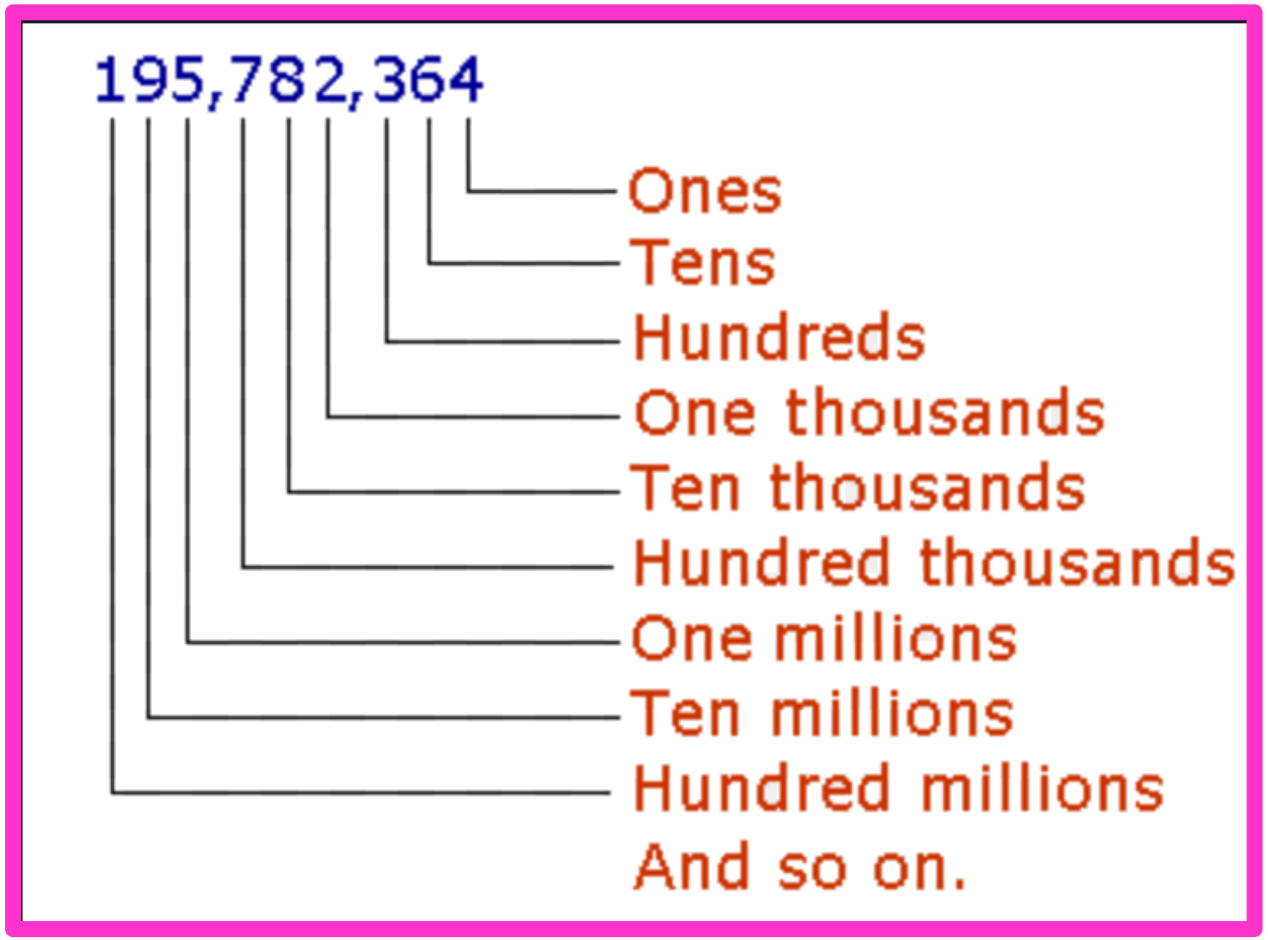


Radix Sort

601, 008, 035, 645, 182, 086

Now we arrange our list by place value hundreds.

008, 035, 086, 182, 601, 645



Radix Sort

645, 086, 182, 035, 601, 008

601, 182, 645, 035, 086, 008

601, 008, 035, 645, 182, 086

008, 035, 086, 182, 601, 645

Radix Sort

Radix Definition

The **base** of a system of numeration

So "Radix Sort" means a sort that uses the **base** of a system of numeration.

So can we use Radix Sort on numbers in bases other than 10?

Radix Sort

Let's start with a list of binary numbers

110, 11, 101, 1, 100, 111 , 10

Add leading zeros

110, 011, 101, 001, 100, 111 , 010

110, 011, 101, 001, 100, 111 , 010

Using ones, tens and hundreds place value is more applicable to the decimal system.

So let's talk about Most Significant Bit (MSB) and Least Significant Bit (LSB).

Radix Sort

MSB

the bit in a binary number which is of the greatest numerical value
the leftmost/first bit of a number

LSB

the bit in a binary number which is of the lowest numerical value.
the rightmost/last bit of a number

Radix Sort

110, 011, 101, 001, 100, 111 , 010

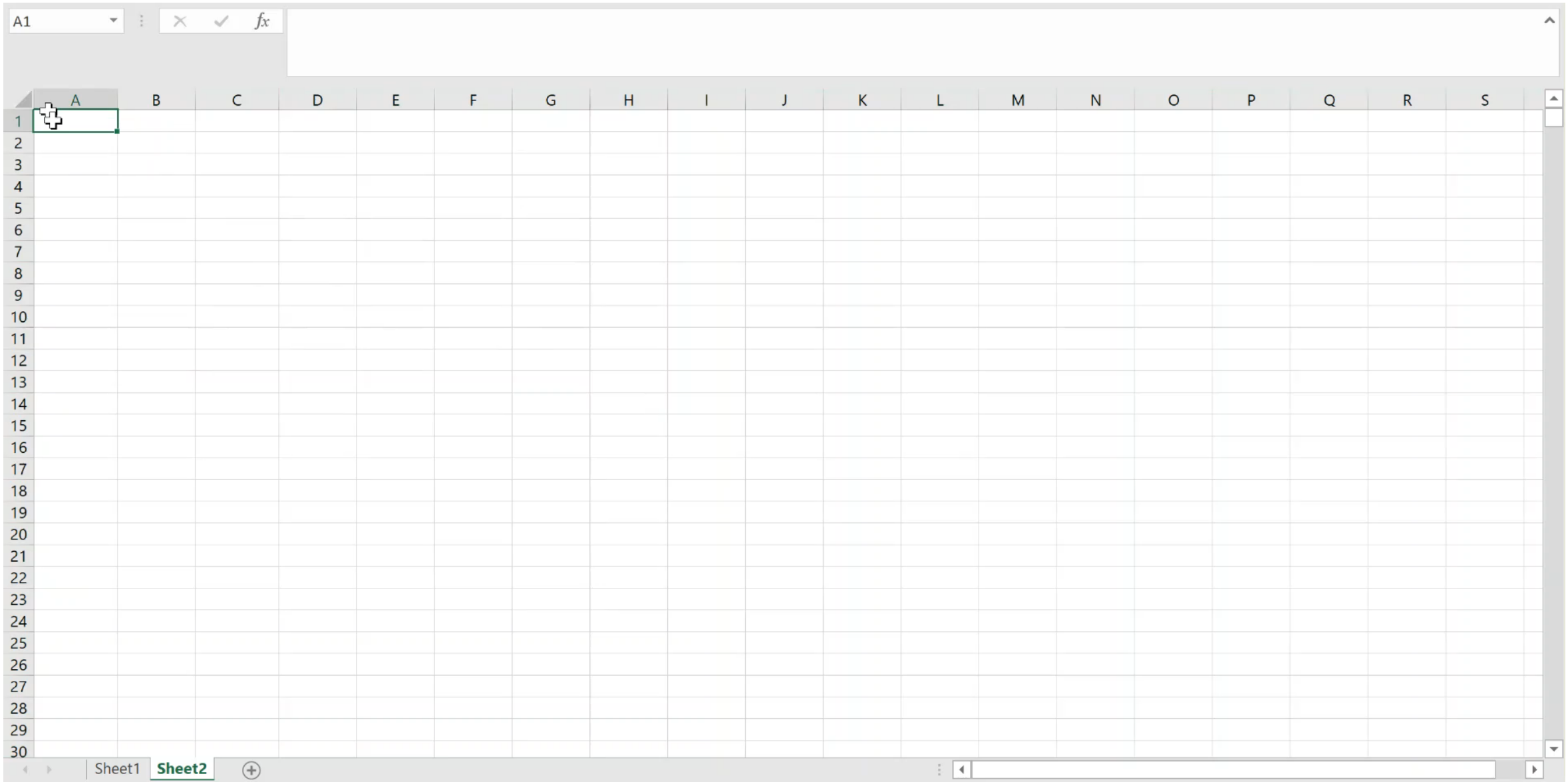
Let's arrange them starting with LSB and work towards the MSB.

110, 100, 010, 011, 101, 001, 111

100, 101, 001, 110, 010, 011, 111

001, 010, 011, 100, 101, 110, 111

Radix Sort


A1
fx
A B C D E F G H I J K L M N O P Q R S
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
Sheet1
Sheet2

Radix Sort Time Complexity

So what is the time complexity of Radix Sort?

Let's establish a few terms

d is the number of digits in the numbers in the list – use the number of digits in the biggest number in the list.

n is the number of numbers in the list

b is the base/radix we are using

Radix Sort Time Complexity

For every number in our list, *n*, we checked each digit.

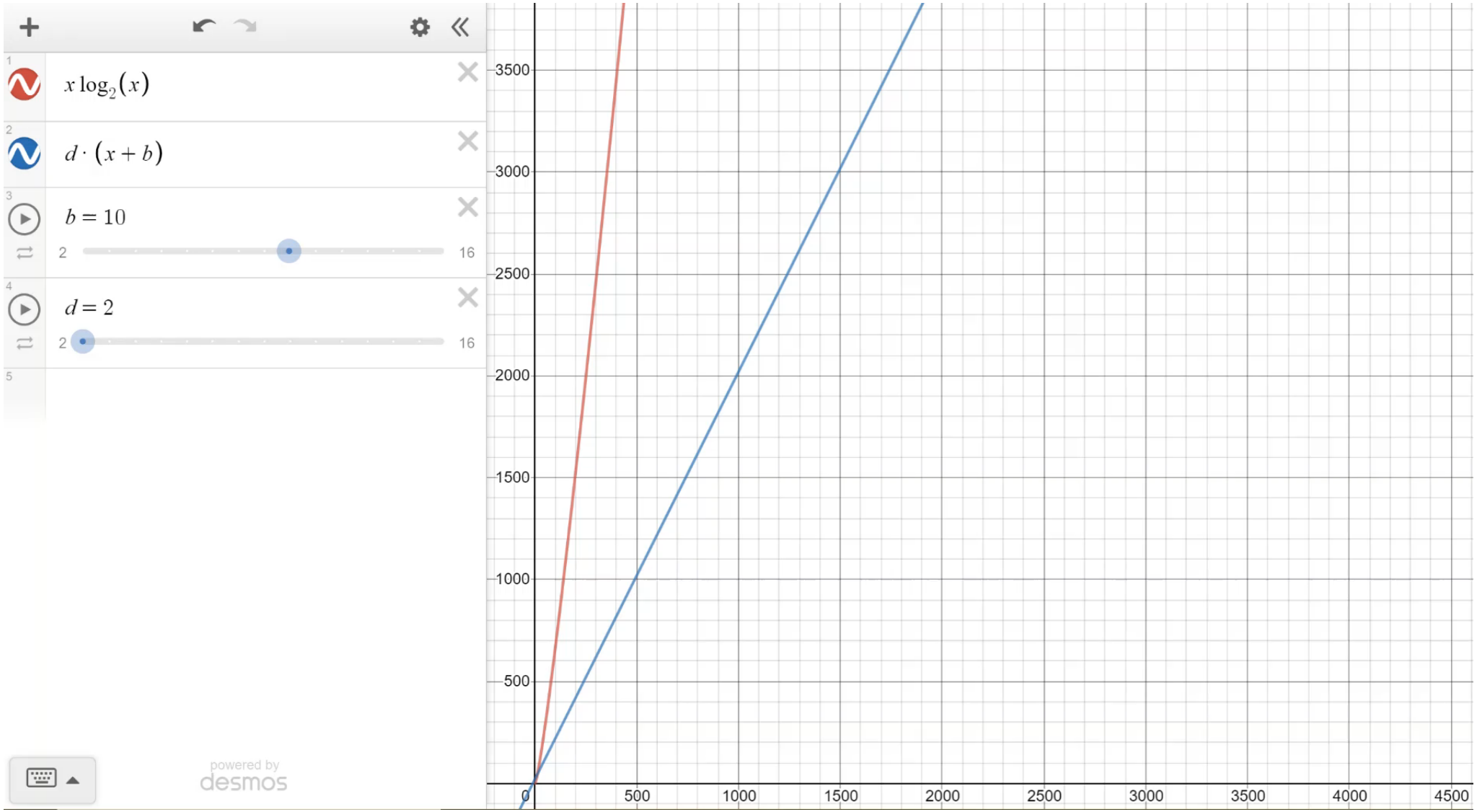
$d*n$

When we had a base 10 number, we had to check 10 digits -> 0-9.
When we had a base 2 number, we had to check 2 digits -> 0 and 1.

So the base influenced how much time we spent on each number.

$O(d * (n+b))$

Radix Sort Time Complexity



Radix Sort Time Complexity

$O(d * (n+b))$

As we saw in the graph, changing the base did not influence the run time very much.

You will also see the time complexity of radix sort written as

$O(nk)$

where k is the number of digits

Radix Sort

Interesting Thoughts on Radix Sort

Radix Sort does not use any comparisons or swaps.

The time complexity of Radix Sort maintains its coefficient (number of digits) because of the significant impact of that value on the time complexity.

$O(nk)$

Even asymptotic notation does not wave away the impact of the number of digits.

Radix Sort

So if Radix Sorts has a better run time than most other sorts, why isn't it used more?

The biggest limitation of Radix Sort is that it is limited to data that only consists of digits or letters.

Radix Sort also does not actually move any elements around in the storage container (like an array) – it makes a new version of the storage container; therefore, has a higher space complexity than some other "slower" sorts. O(n+k).