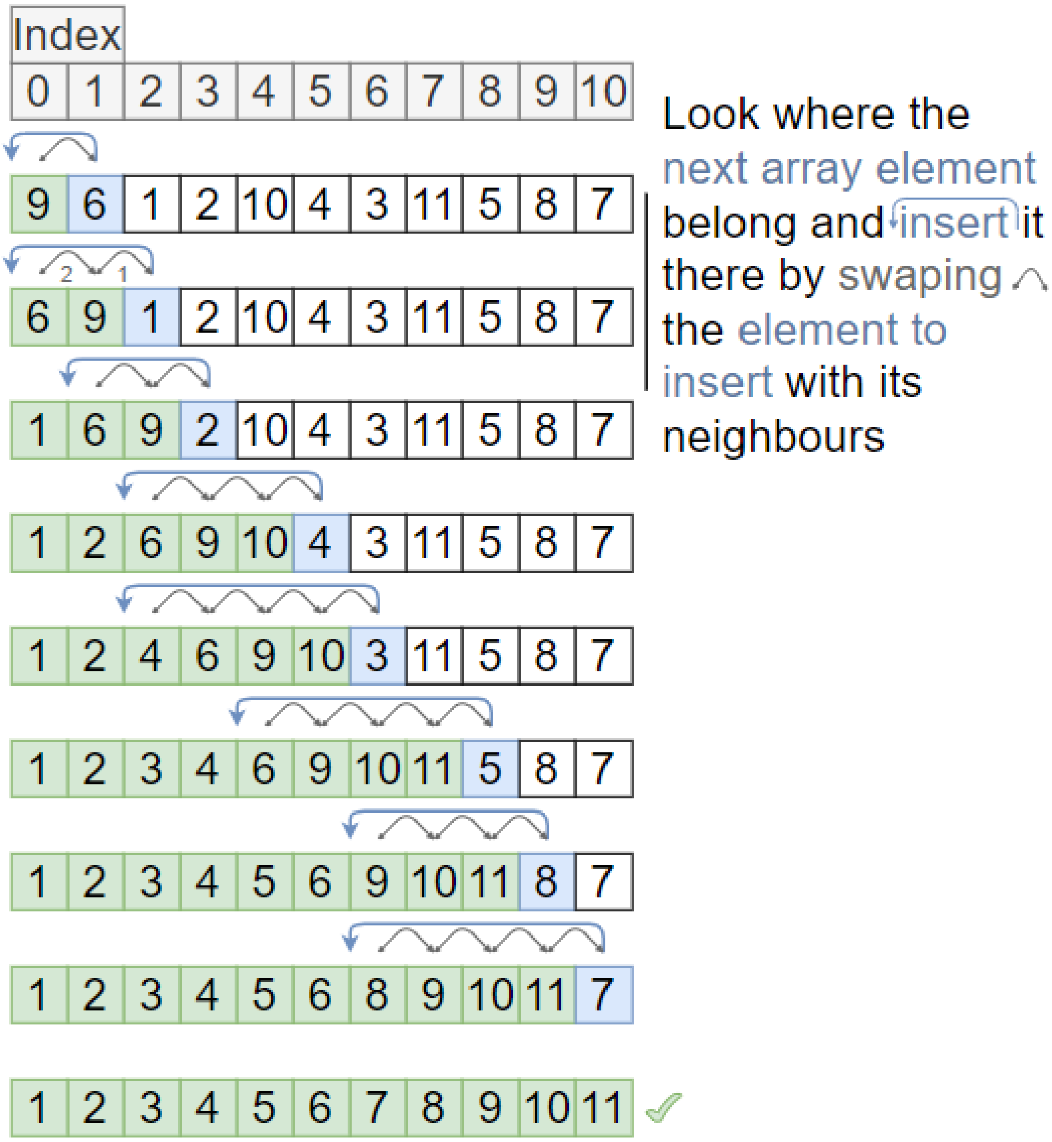
Dustin Scharf

Sorting Algorithms *28.07.2021*

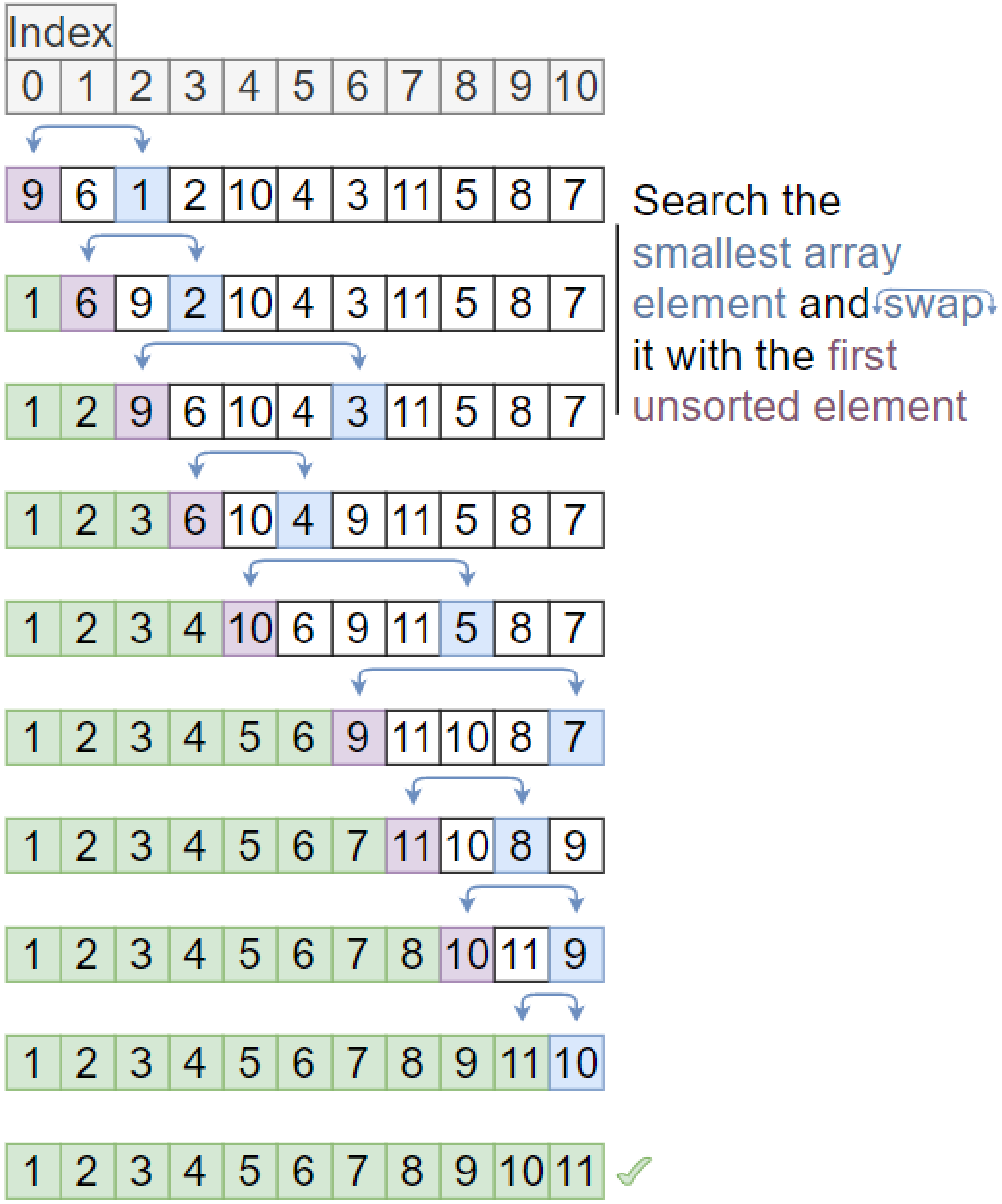
Darstellung einiger Sortierverfahren anhand eines Beispiels mit Erklärung *(engl.)*.

Insertion Sort



Dustin Scharf

Selection Sort



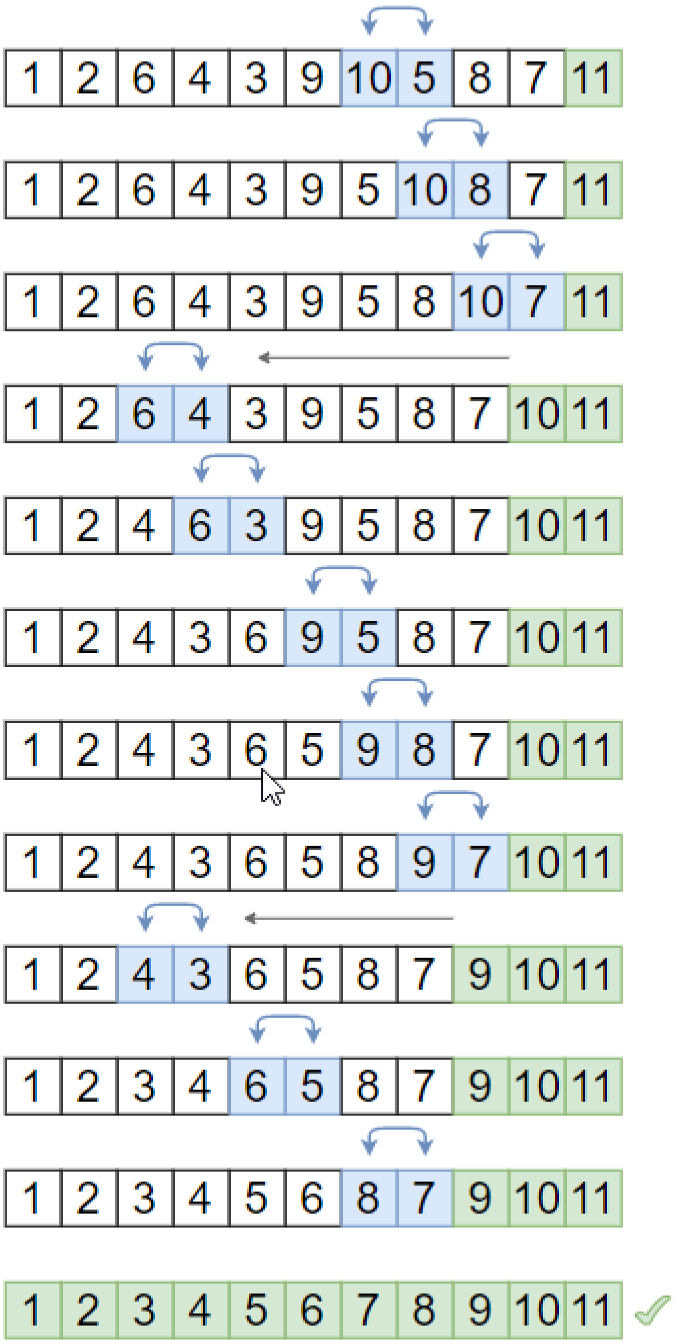
Dustin Scharf

Bubble Sort

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

Walk from left to right and swap neighbours if they are out of order, go to start of array until everything is sorted

Dustin Scharf



Dustin Scharf

Bucket Sort

Index										
0	1	2	3	4	5	6	7	8	9	10

9	6	1	2	10	4	3	11	5	8	7

f(x)=0.09x

.81	.54	.09	.18	.9	.36	.27	.99	.45	.72	.63

Buckets										
0	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	
.09	.18	.27	.36	.45	.54	.63	.72	.81	.9	
									11	
									.99	

1	2	3	4	5	6	7	8	9	10	11

Find a function f that maps a value from 0 (exclusive) to 1 (inclusive) to each array element and wouldn't change the sorted order of the elements (here f is *f(x)=0.09x*)

now create n buckets and put the elements into the bucket number ***FLOOR[f(element)*n]***

finally sort the buckets itself and put all the bucket content into the final array, starting from the bucket with the smallest elements

Dustin Scharf

Counting Sort

Index

0	1	2	3	4	5	6	7	8	9	10

9	6	1	2	10	4	3	11	5	8	7

Counter

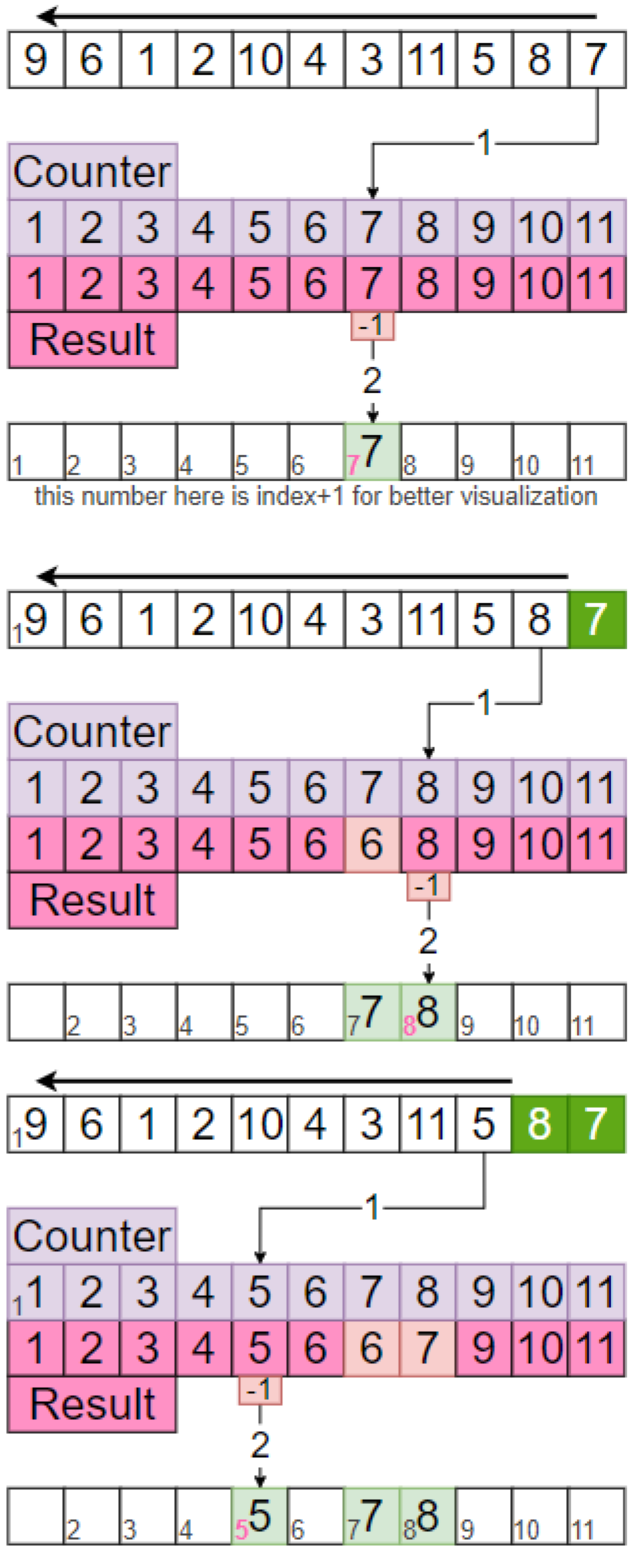
1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	1	1	1	1	1

1+	1	1	1	1	1	1	1	1	1	1
1	**2**+	1	1	1	1	1	1	1	1	1
1	2	**3**	1	1	1	1	1	1	1	1
1	2	3	**4**	1	1	1	1	1	1	1
1	2	3	4	**5**	1	1	1	1	1	1
1	2	3	4	5	**6**	1	1	1	1	1
1	2	3	4	5	6	**7**	1	1	1	1
1	2	3	4	5	6	7	**8**	1	1	1
1	2	3	4	5	6	7	8	**9**	1	1
1	2	3	4	5	6	7	8	9	**10**	1
1	2	3	4	5	6	7	8	9	10	**11**

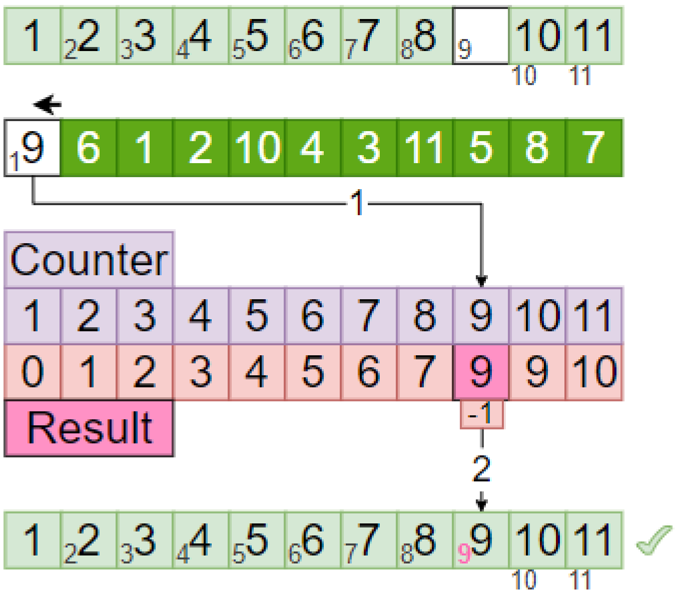
Create a counter field, and count how ofter an element appears

now start from the left of the counter and add this left element to element in the next place, then do this for the 2. and 3. element and so on, until the end of the counter, to get the result

yet start from the end of the unsorted input array and look what number **n** is there, then look how often **n** was counted in the result (m times) and insert **n** in a new array at index m-1
finally subtract 1 from m
do the steps above for the whole unsorted input array


9 6 1 2 10 4 3 11 5 8 7
1
Counter
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 7 8 9 10 11
Result
-1
2
this number here is index+1 for better visualization
Counter
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 6 8 9 10 11
Result
Counter
1 2 3 4 5 6 7 8 9 10 11
1 2 3 4 5 6 6 7 9 10 11
Result

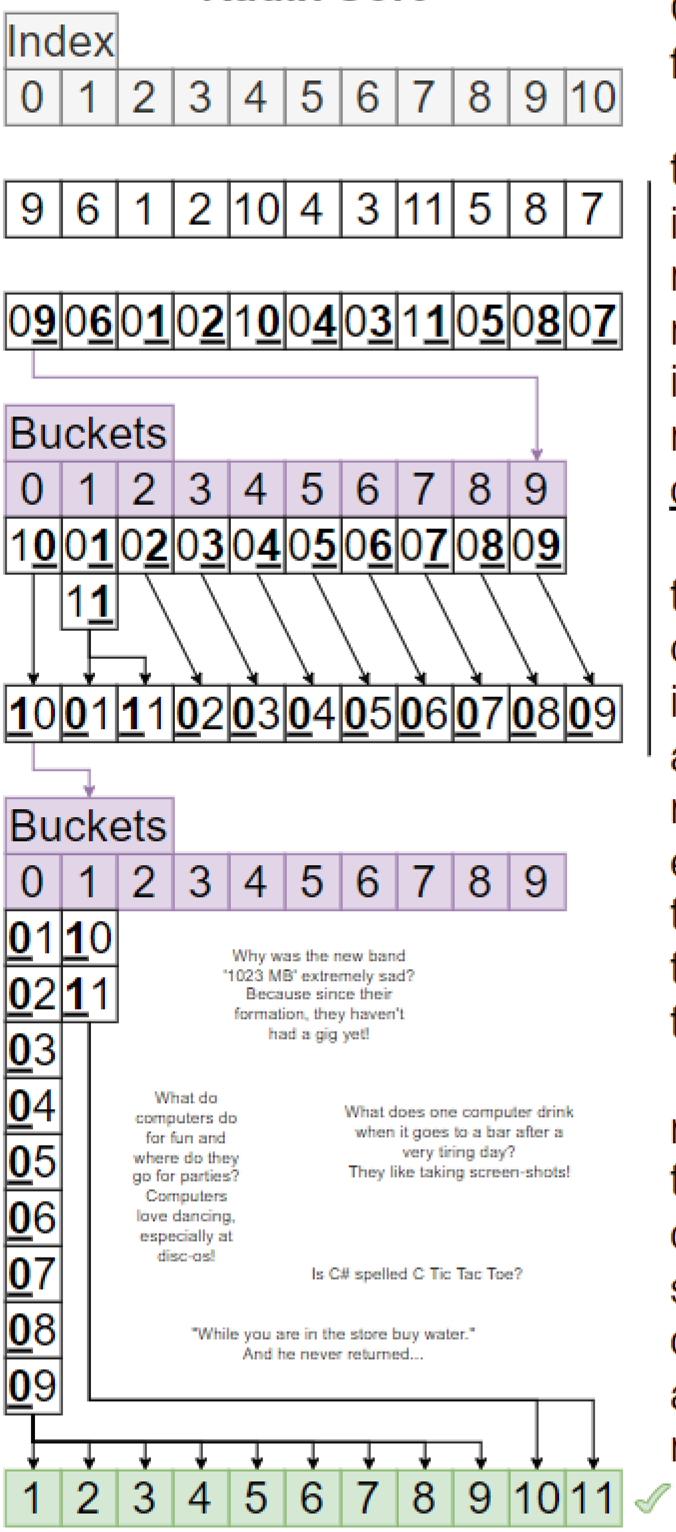
(. . .)



1 2 3 4 5 6 7 8 9 10 11
9 6 1 2 10 4 3 11 5 8 7
1
Counter
1 2 3 4 5 6 7 8 9 10 11
0 1 2 3 4 5 6 7 9 9 10
Result
-1
2
1 2 3 4 5 6 7 8 9 10 11

Dustin Scharf

Radix Sort



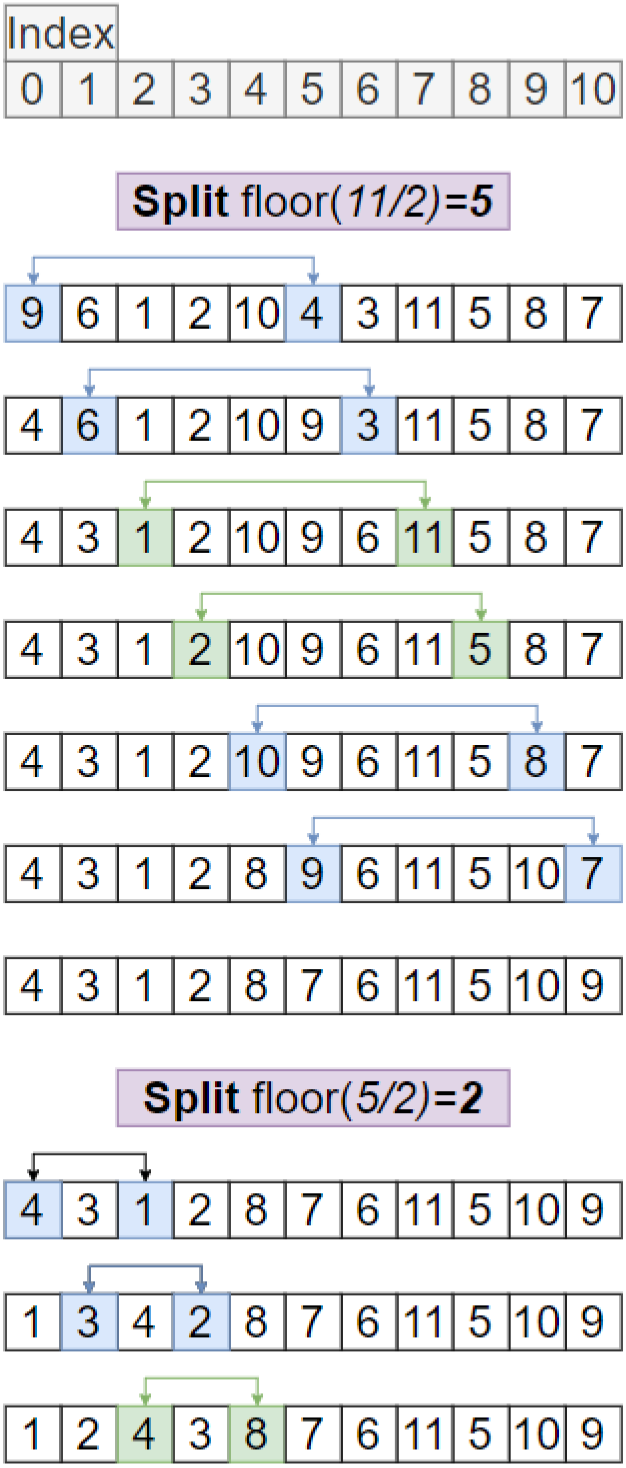
Create 10 buckets for the numbers 0-9

then go through the input array left to right and insert each number of the array in the bucket with the number of its last digit

then put the bucket content left to right into the input array (if a bucket contains more than one element, put them in the same order into the input array, as they were before)

now do the same for the seccond last digit, then third, and so on, until this was done so many times as the longest number has digtits

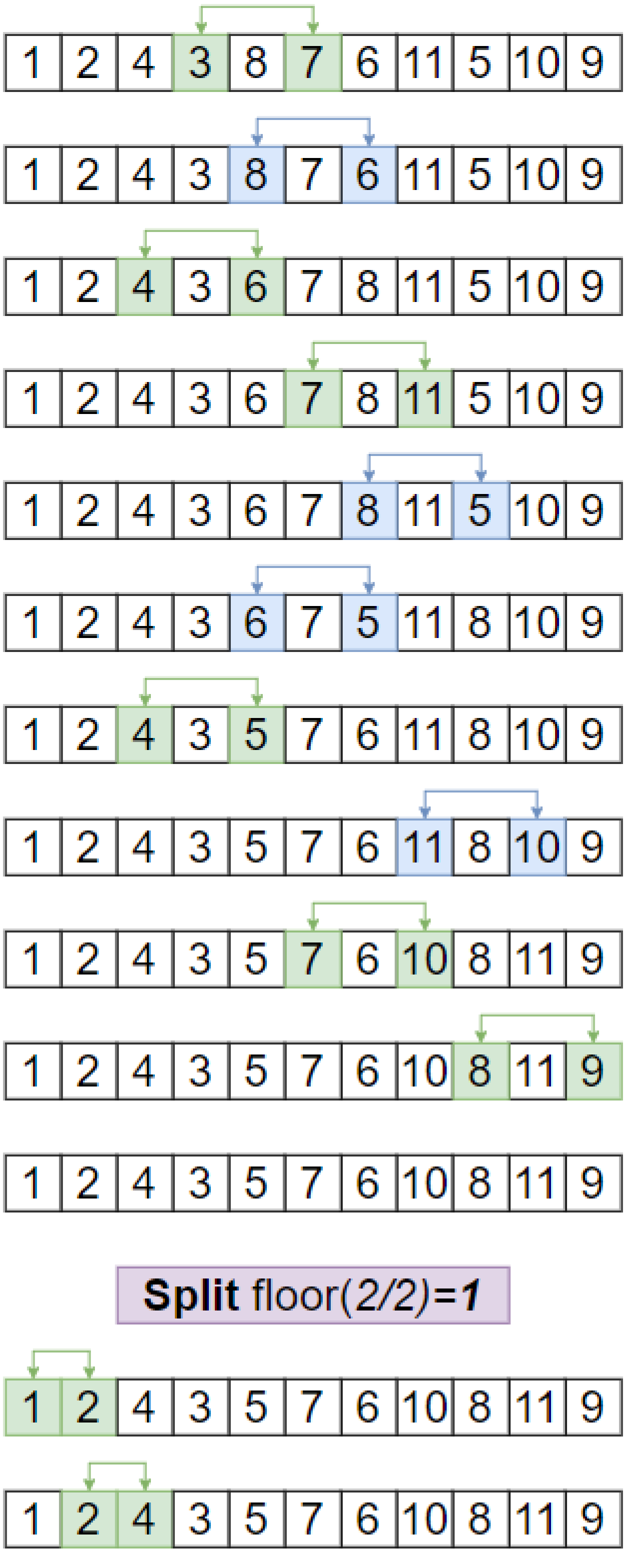
Shell Sort

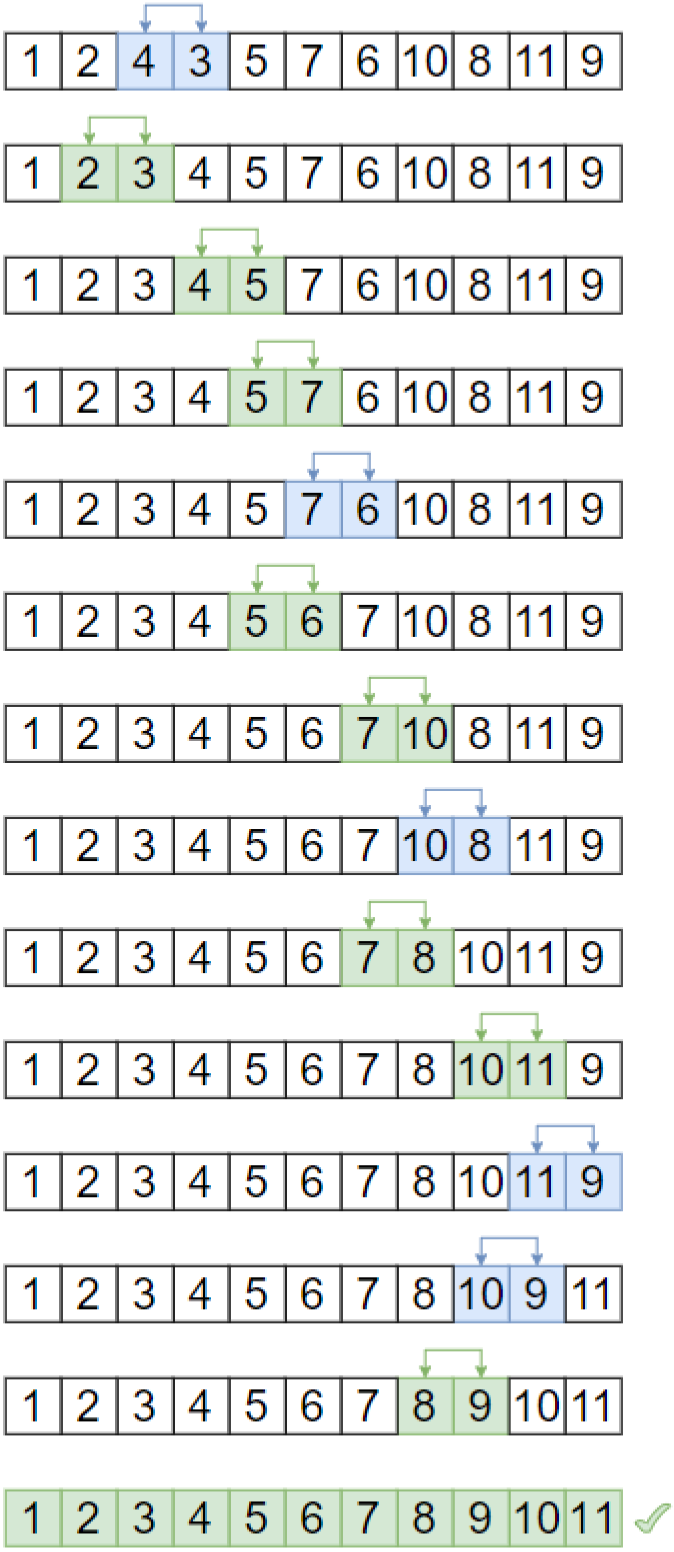


Divide the length of the input array by a split number (here 2) and floor the **result**

start with element at index 0 and sort all array elements with distance **result** from this element at index 0
now do this for index 1, then 2, and so on, until the end of the array is reached

then divide the **result** by the split number again and do the procedure described above again until the **result** is 1 and the array is finally sorted


1 2 4 3 8 7 6 11 5 10 9
1 2 4 3 8 7 6 11 5 10 9
1 2 4 3 6 7 8 11 5 10 9
1 2 4 3 6 7 8 11 5 10 9
1 2 4 3 6 7 8 11 5 10 9
1 2 4 3 6 7 5 11 8 10 9
1 2 4 3 5 7 6 11 8 10 9
1 2 4 3 5 7 6 11 8 10 9
1 2 4 3 5 7 6 10 8 11 9
1 2 4 3 5 7 6 10 8 11 9
1 2 4 3 5 7 6 10 8 11 9
Split floor(2/2)=1
1 2 4 3 5 7 6 10 8 11 9
1 2 4 3 5 7 6 10 8 11 9

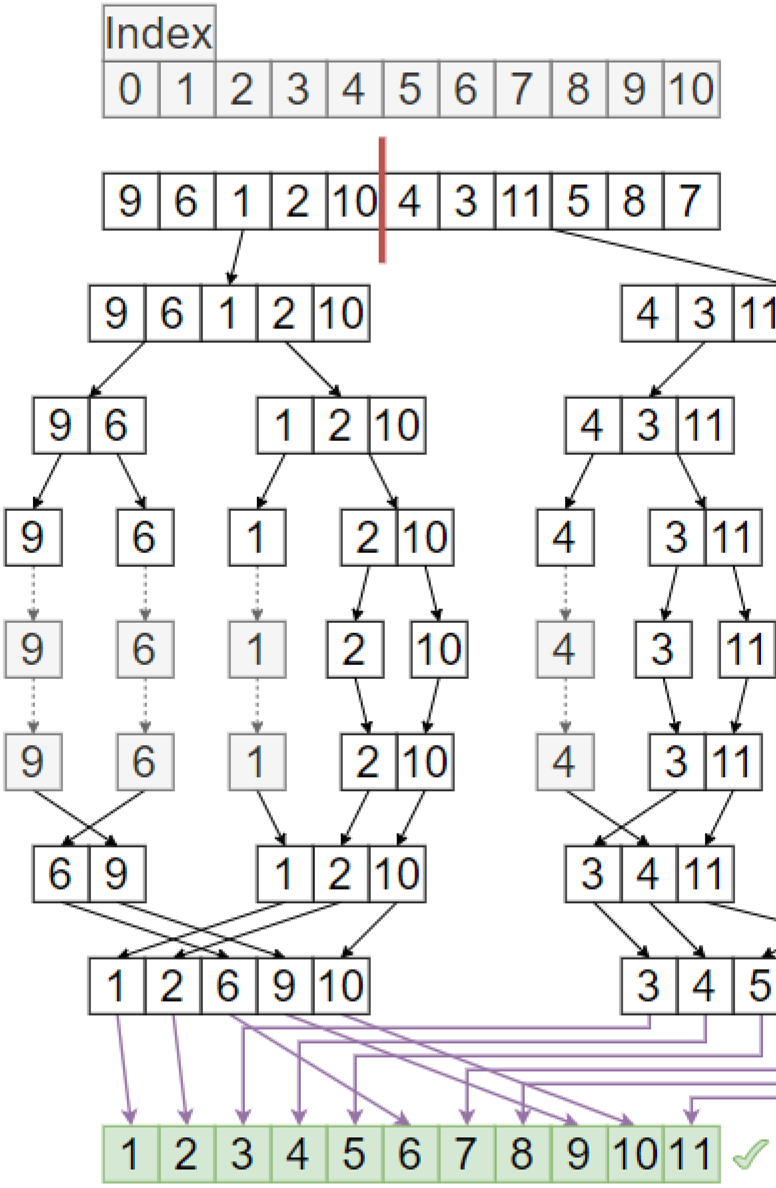

1 2 4 3 5 7 6 10 8 11 9
1 2 3 4 5 7 6 10 8 11 9
1 2 3 4 5 7 6 10 8 11 9
1 2 3 4 5 7 6 10 8 11 9
1 2 3 4 5 7 6 10 8 11 9
1 2 3 4 5 6 7 10 8 11 9
1 2 3 4 5 6 7 10 8 11 9
1 2 3 4 5 6 7 10 8 11 9
1 2 3 4 5 6 7 8 10 11 9
1 2 3 4 5 6 7 8 10 11 9
1 2 3 4 5 6 7 8 10 11 9
1 2 3 4 5 6 7 8 10 9 11
1 2 3 4 5 6 7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

Merge Sort

Split the input array into a left and right half (if element amount is odd, left half has one element more than the right half)

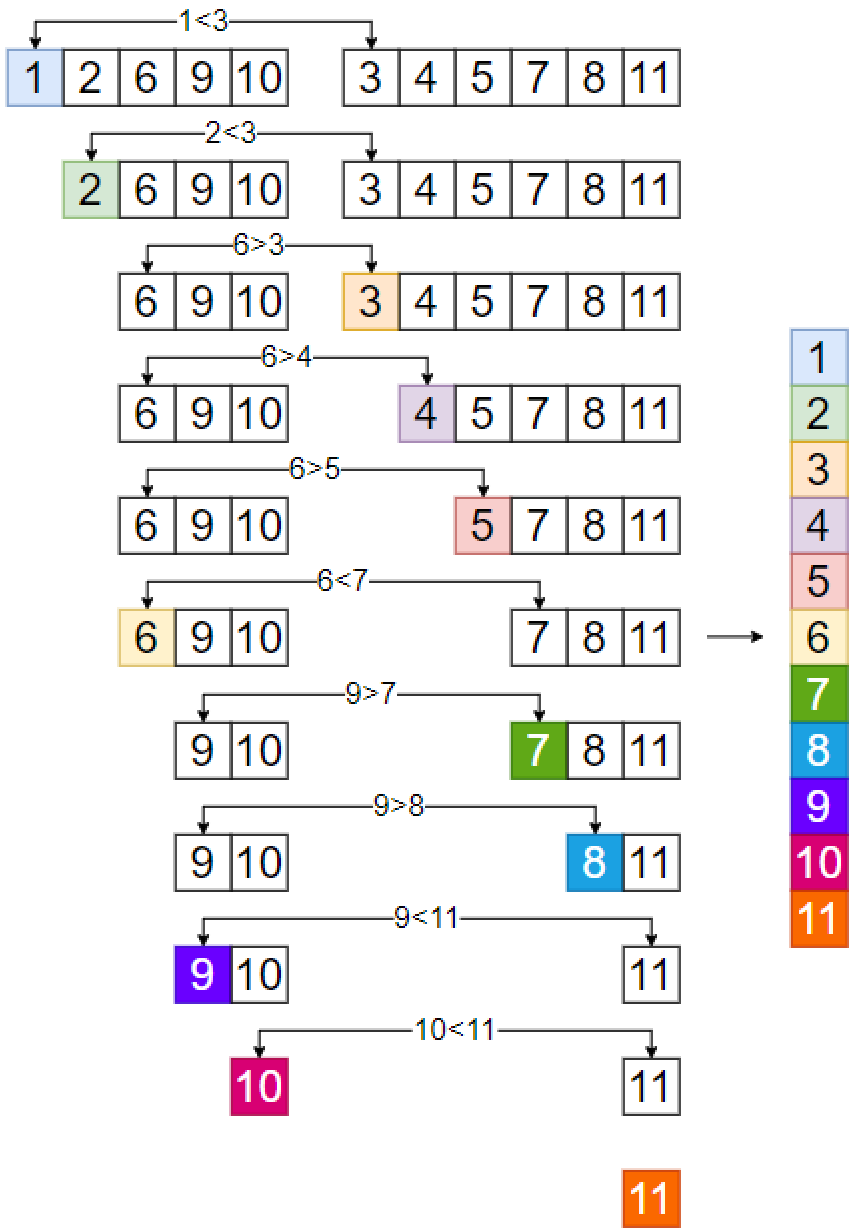
split until each "half" contains only one element

then merge all the "halfs" together in the same order as they were splitted
(merge is done by checking what first element of the 2 "halfs" is less, move this element in the final array and look again what first element is smaller, and repeat this until everything is in the final array)

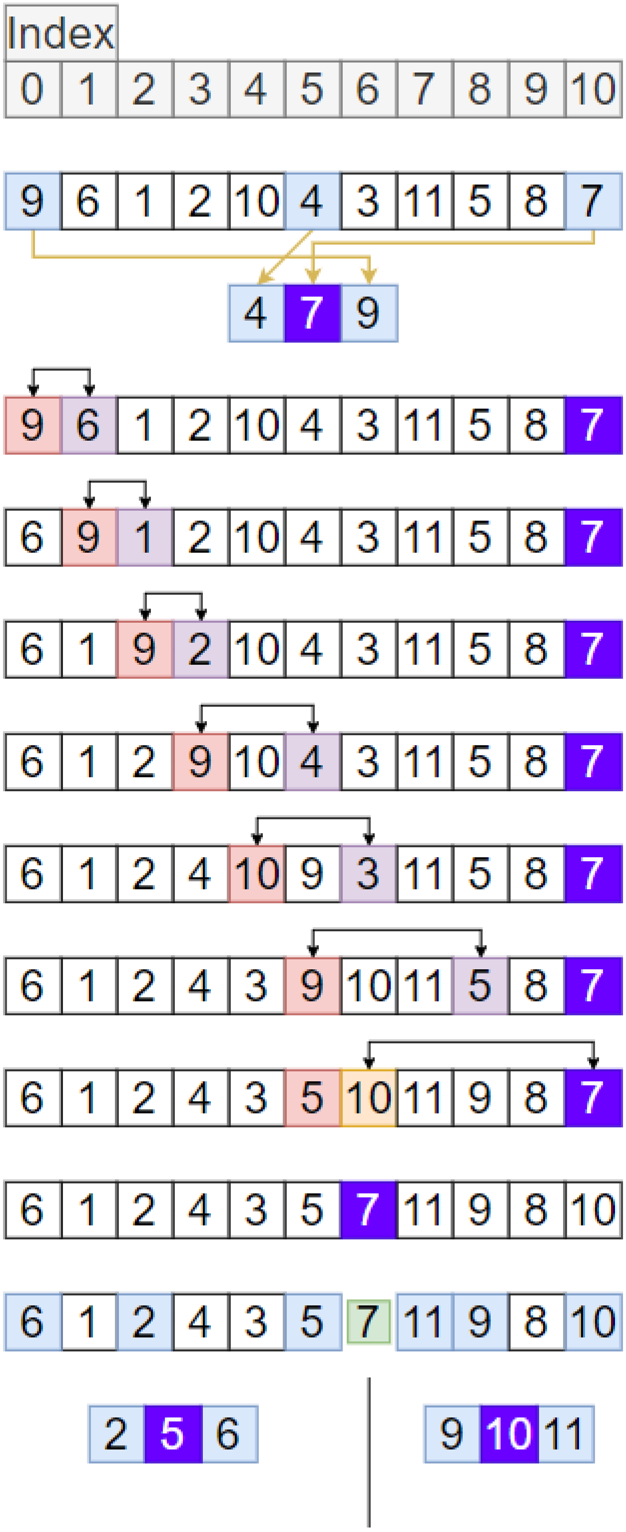

Index
0 1 2 3 4 5 6 7 8 9 10
9 6 1 2 10 4 3 11 5 8 7
9 6 1 2 10
4 3 11 5 8 7
1 2 6 9 10
3 4 5 7 8 11
1 2 3 4 5 6 7 8 9 10 11

Dustin Scharf

Array Merge Example



Quick Sort



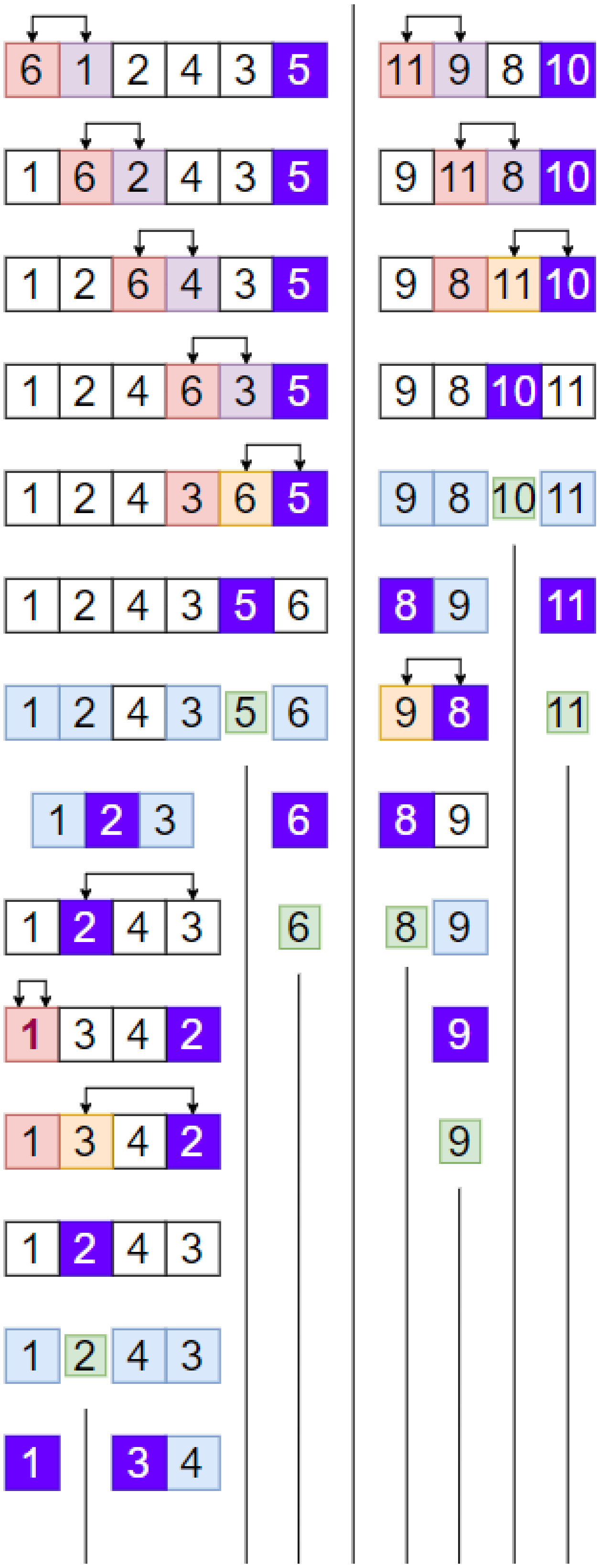
Choose a **pivot element** (here we take the first, middle and last element of the array, sort them, and take the middle element as **pivot**)

swap **pivot** with the most right array element

then create 2 pointers and let them point right before the first array element

start moving the first pointer from the left and check if the current element is less then the **pivot**, if so, let the second pointer walk one step, finally swap the pointed elements

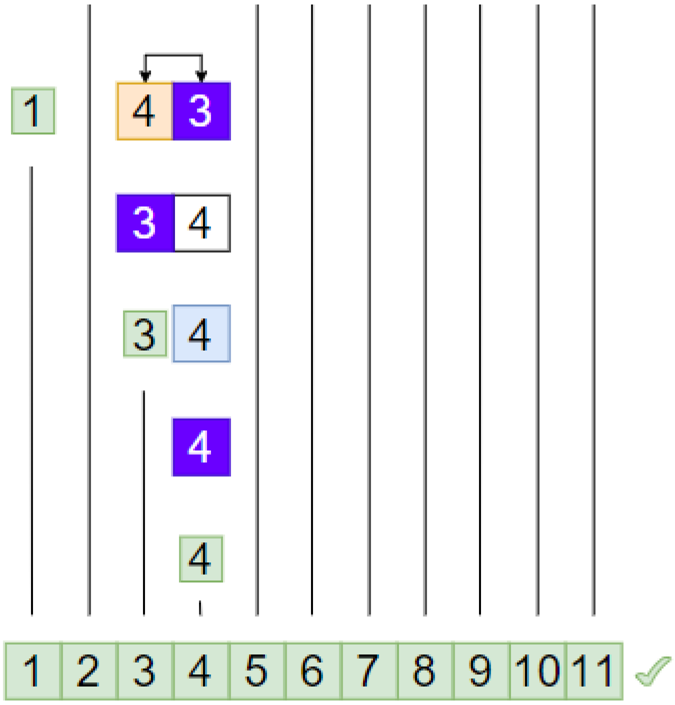
Dustin Scharf



as soon as the first pointer points to the **pivot**, swap the **pivot** with the element next to the second pointer

now all elements left to the **pivot** are less than the **pivot** and all elements right to the **pivot** are greater than the **pivot**
that means we can use the procedure described above for the array parts left and right of the **pivot** recursively unitl everything is sorted

Dustin Scharf



Dustin Scharf

Heap Sort

Heap knowledge is necessary here

Consider the input array as a heap and heapify all non leafs in reversed insertion order, so the result is a max heap

yet swap the root (max element) with the last element in the heap (this is also the last element in the array)

the max element is in the right place of the array now, so we don't mind it anymore

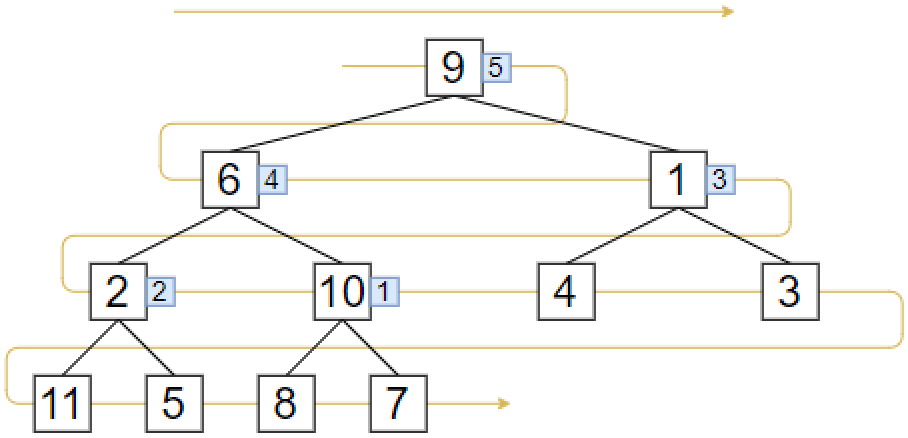
now heapify the new root

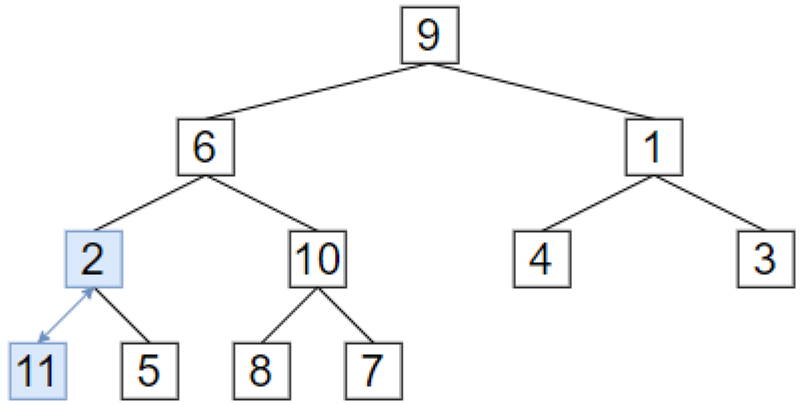
finally repeat the process until the whole array is sorted (make sure to not heapify the already sorted elements)

Dustin Scharf

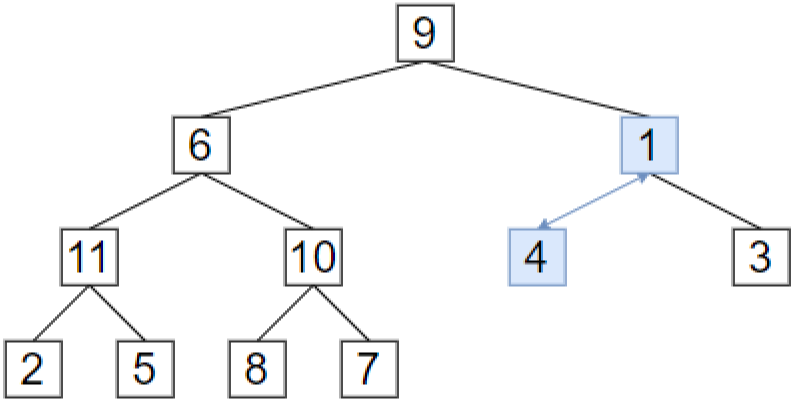
Index										
0	1	2	3	4	5	6	7	8	9	10

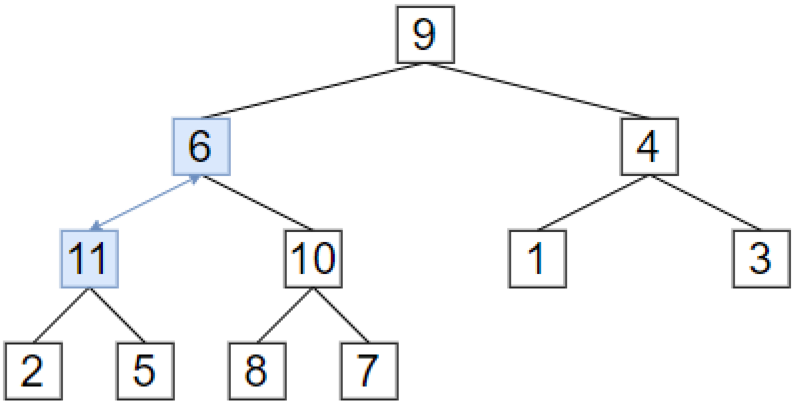
9	6	1	2	10	4	3	11	5	8	7

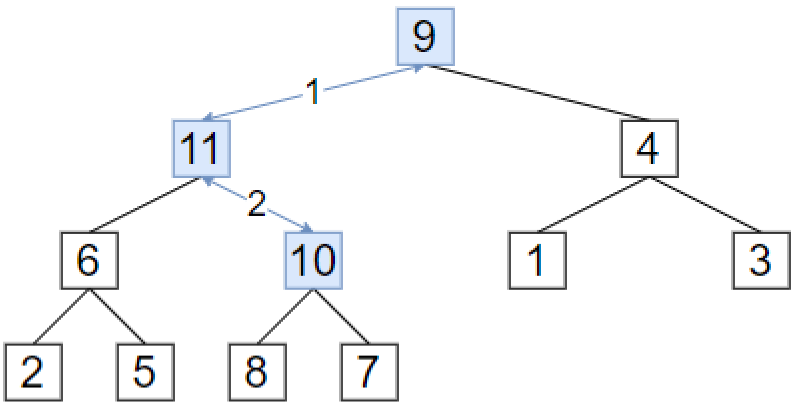


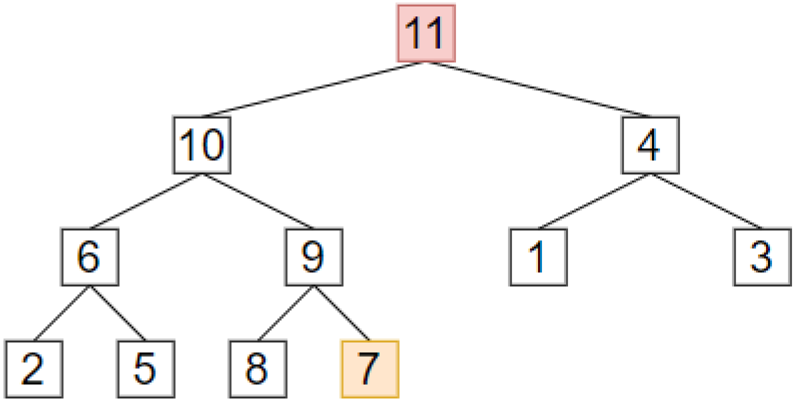


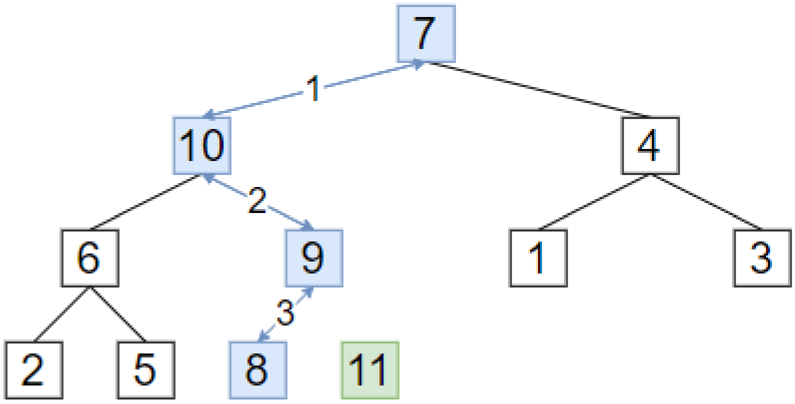
Dustin Scharf

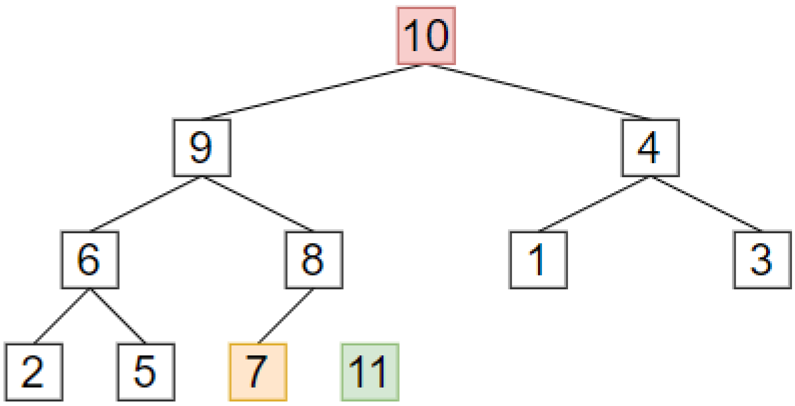




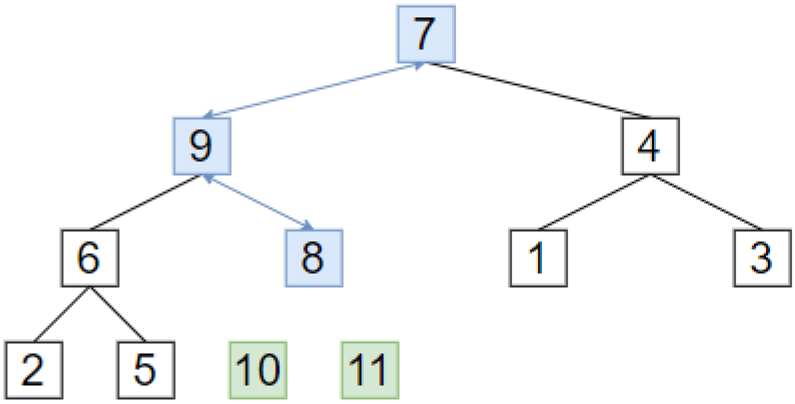


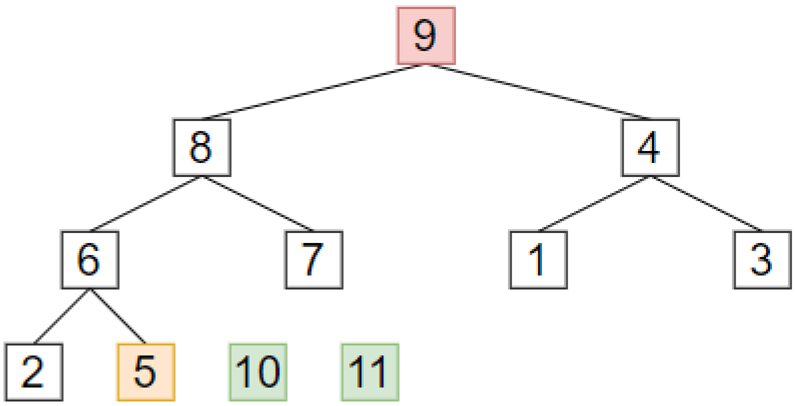

11
10
4
6
9
1
3
2
5
8
7

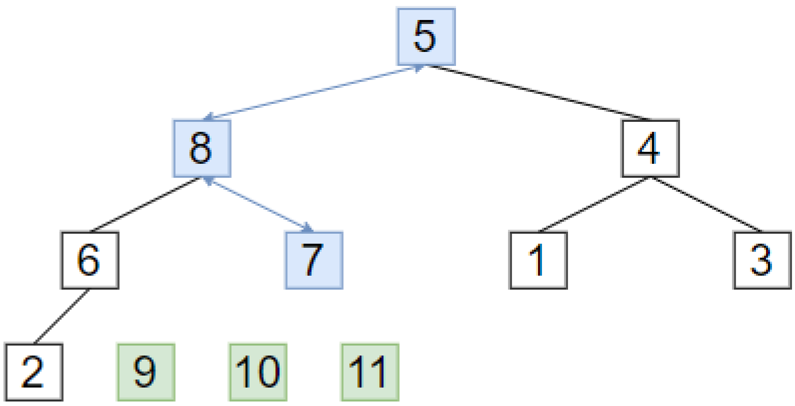

7
1
10
4
2
6
9
1
3
3
2
5
8
11

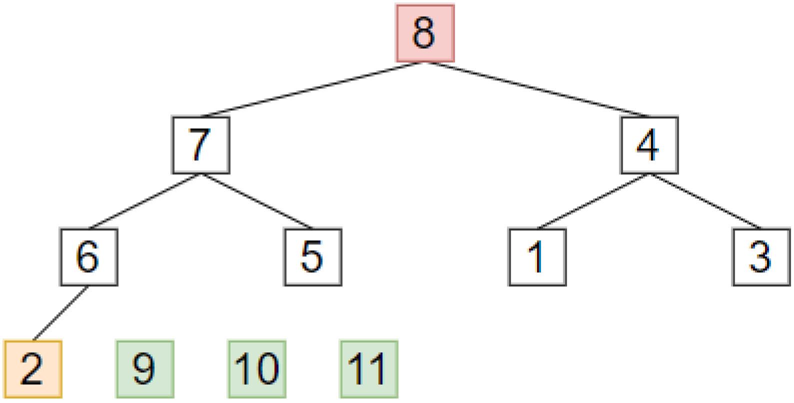

10
9
4
6
8
1
3
2
5
7
11

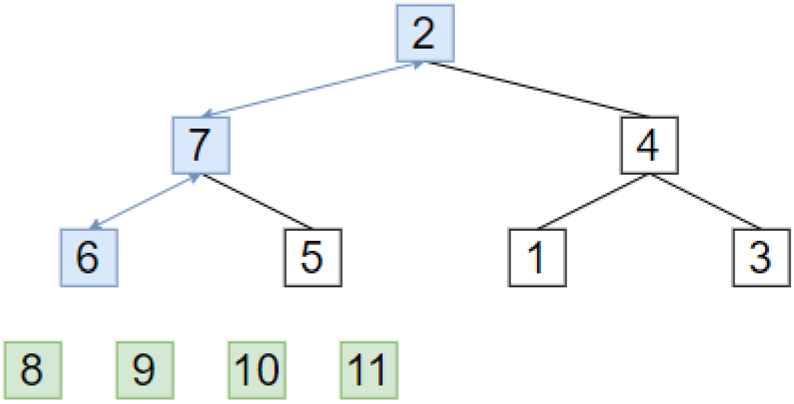
Dustin Scharf

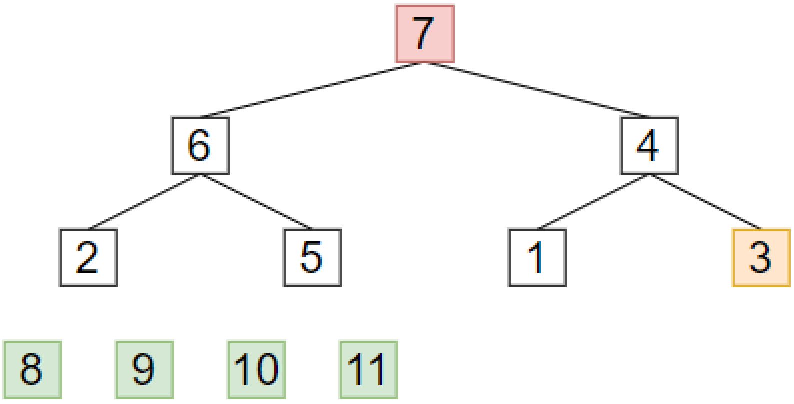




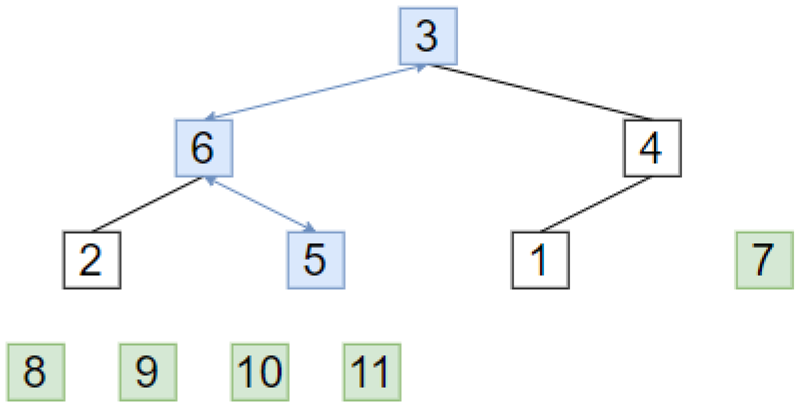


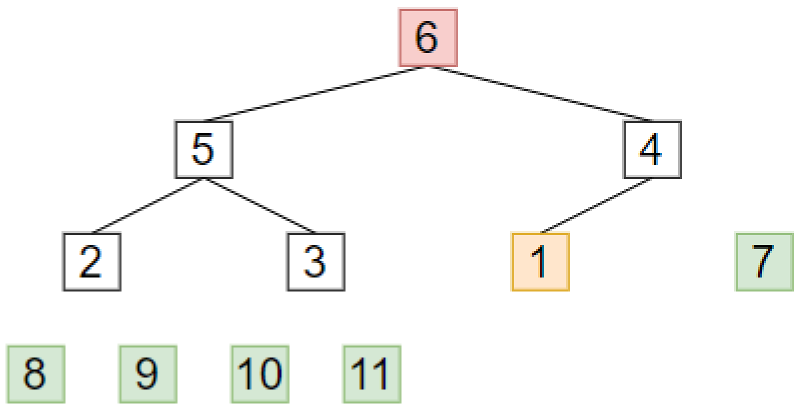

8
7
4
6
5
1
3
2
9
10
11

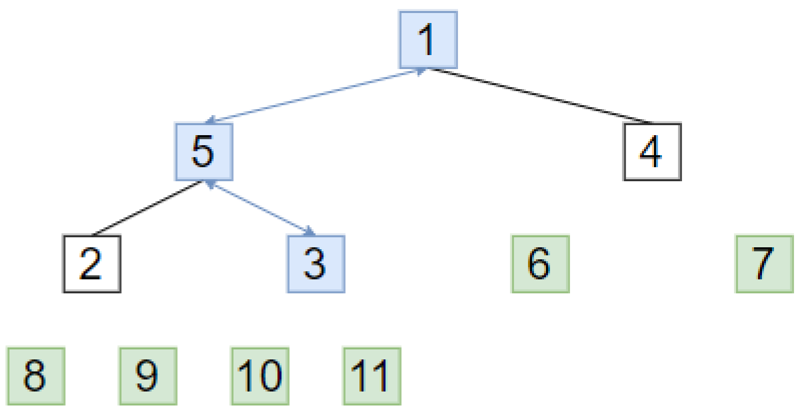

2
7
4
6
5
1
3
8
9
10
11


7
6
4
2
5
1
3
8
9
10
11

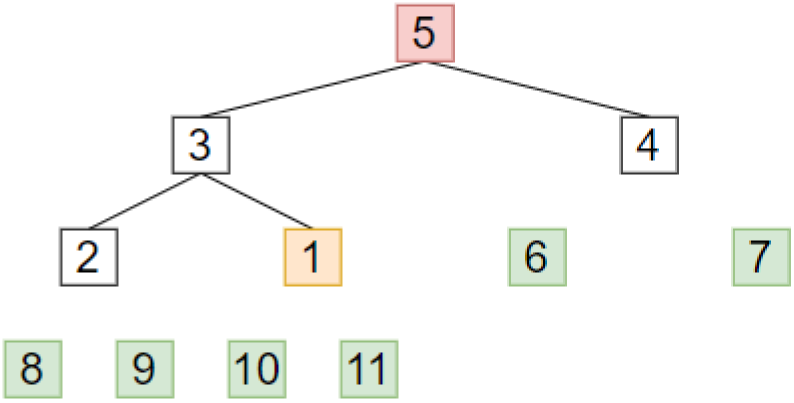
Dustin Scharf

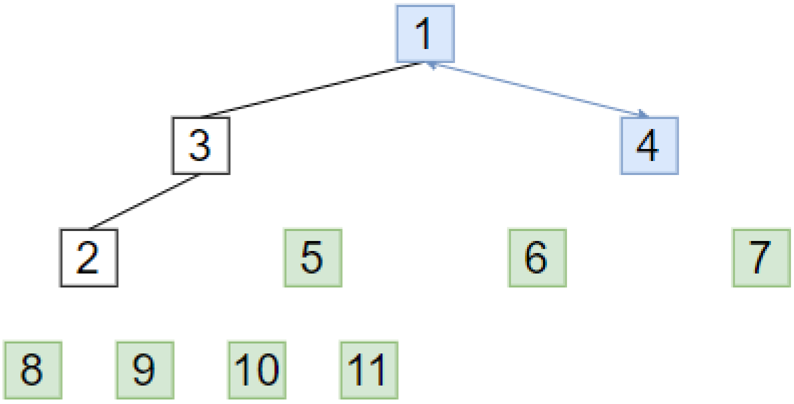


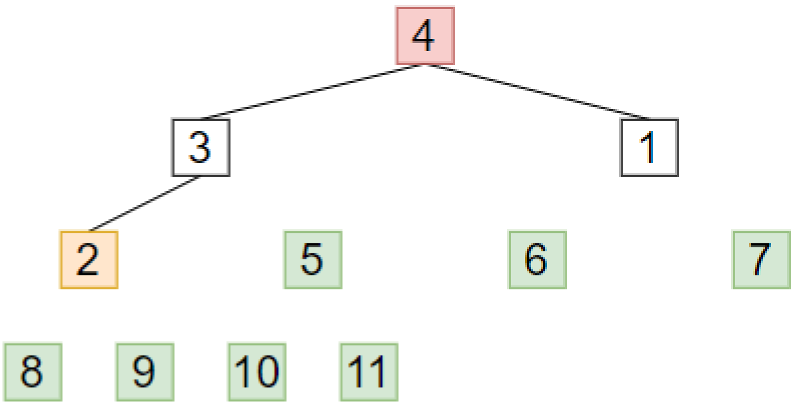




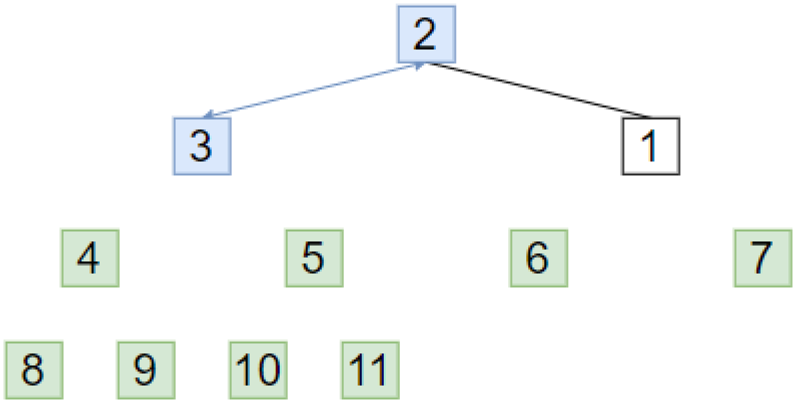
Dustin Scharf

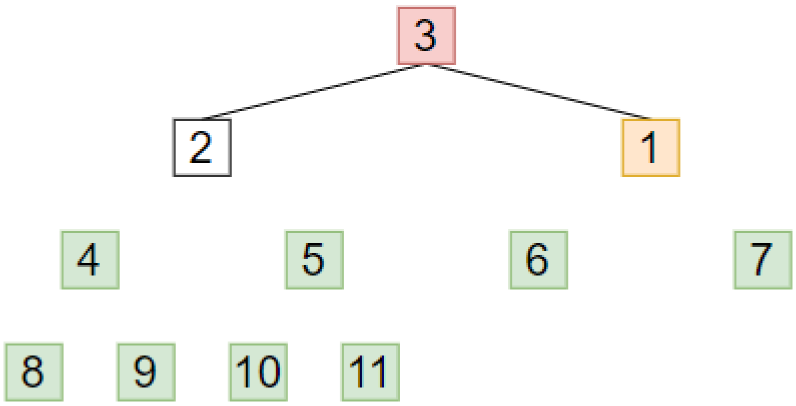


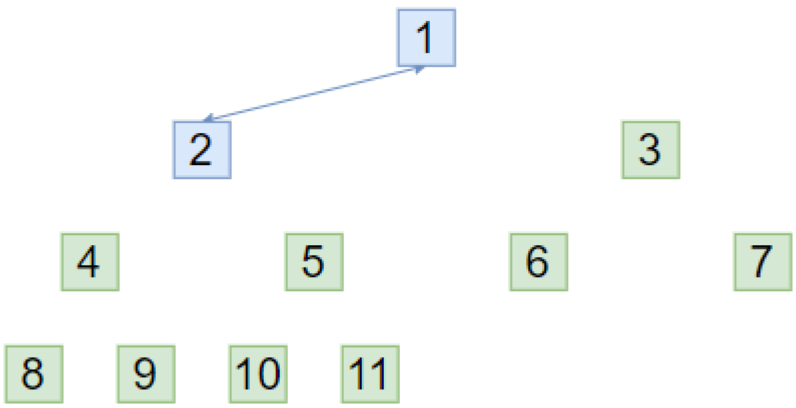




Dustin Scharf







Dustin Scharf