CSCI 132:
Basic Data Structures and Algorithms

Sorting (Part 4)

Reese Pearsall

Spring 2023

Announcements

Program 5 due Sunday May 7th

Fill out the course evaluation

Next Wednesday will be a help session for program 5 (no lecture)

Running Time of Sorting Algorithms

Bubble Sort	Iterate through array and swap pairs of numbers. Large numbers ("bubbles") will rise to the top naturally	$O(n^2)$
Selection Sort	Iterate through the array and find the minimum value **n** times, and place minimum in correct position	$O(n^2)$
Merge Sort	Use recursion to split array in sub-arrays of size. Sort the sub-arrays while merging until you solve the original problem	$O(n \log n)$
Quick Sort	Partition array around a pivot value. Use recursion and place pivot in correct spot and repeat until array is sorted	$O(n^2)$** **Put usually performs much better (O(n log n())

You will not be tested about today's sorting algorithms.

Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

38	27	43	3	9	82	10	14

Insertion Sort

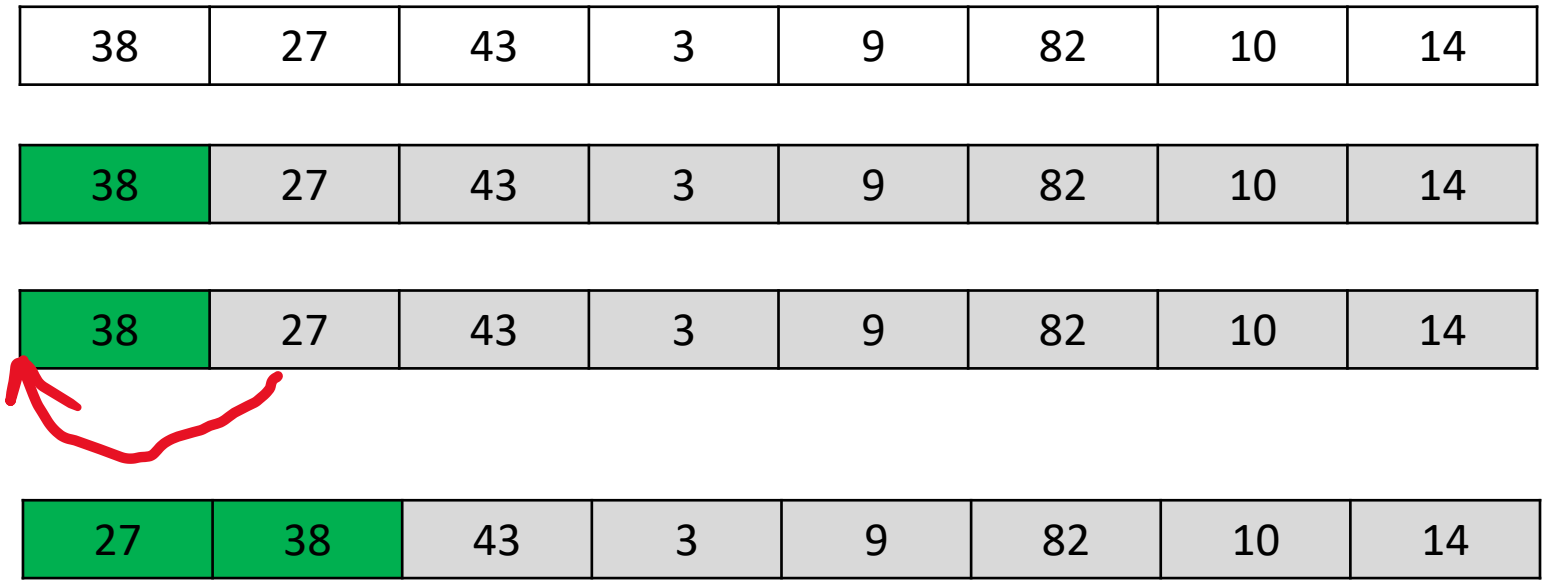
We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

38	27	43	3	9	82	10	14

38	27	43	3	9	82	10	14

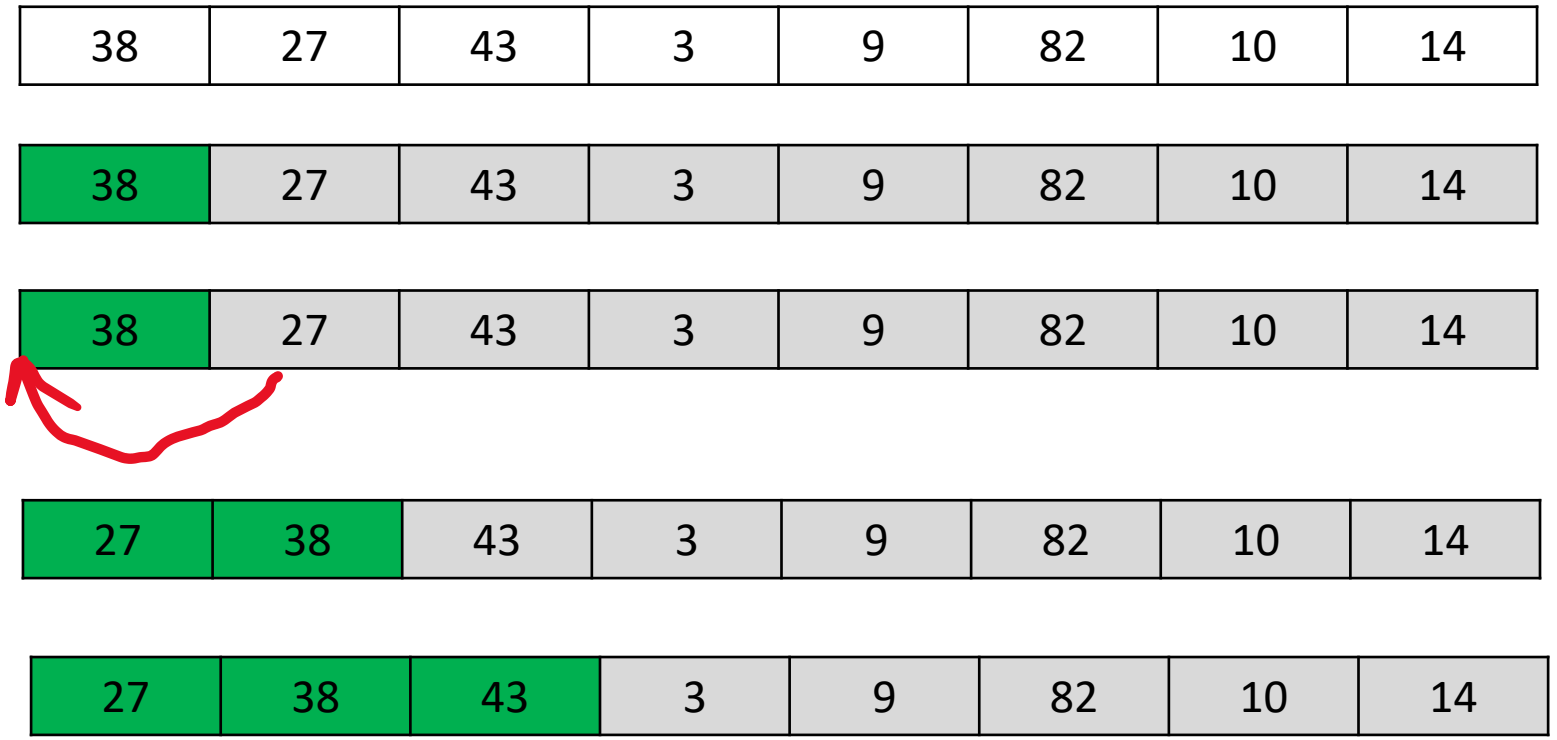
Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



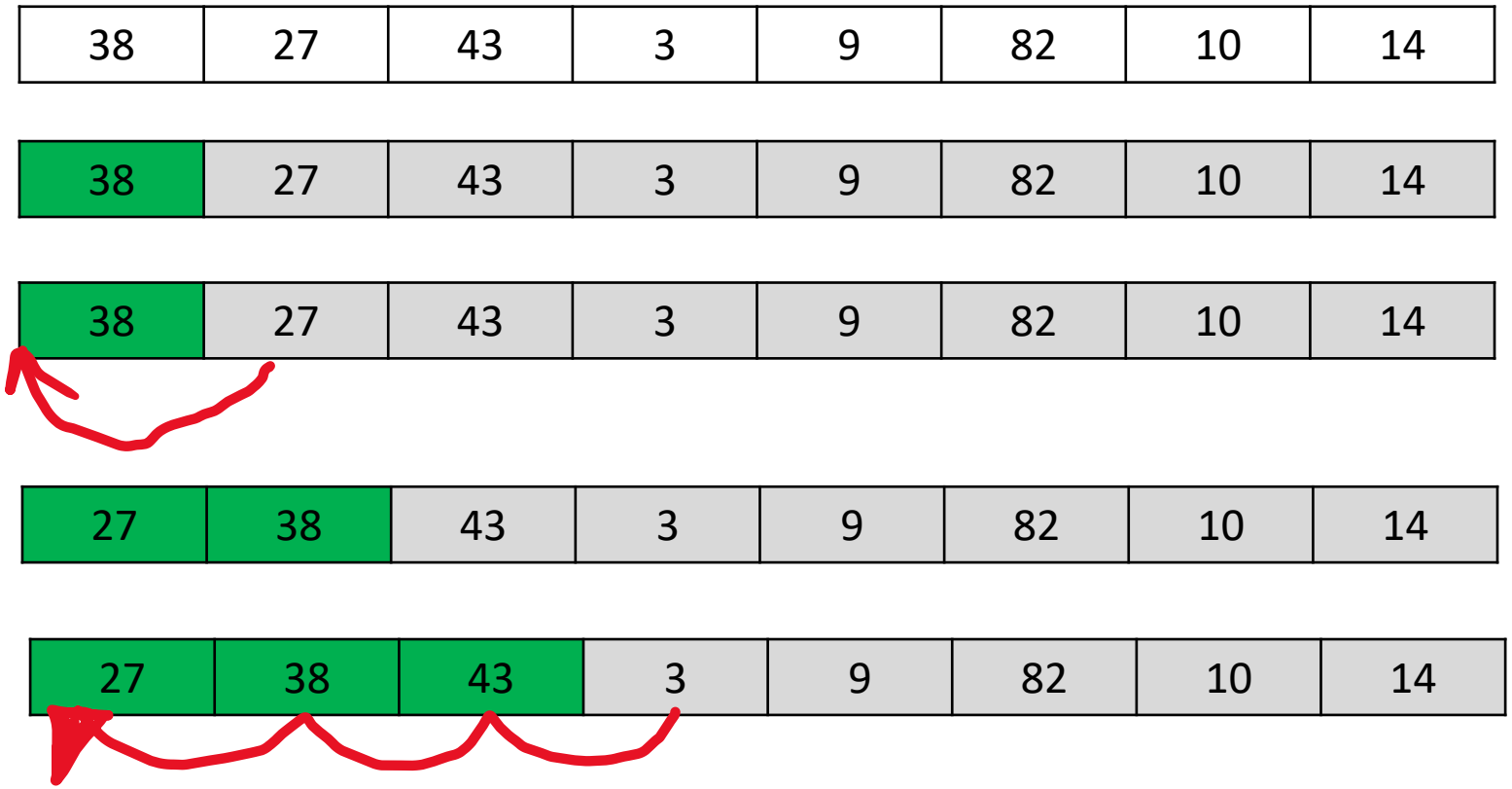
Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



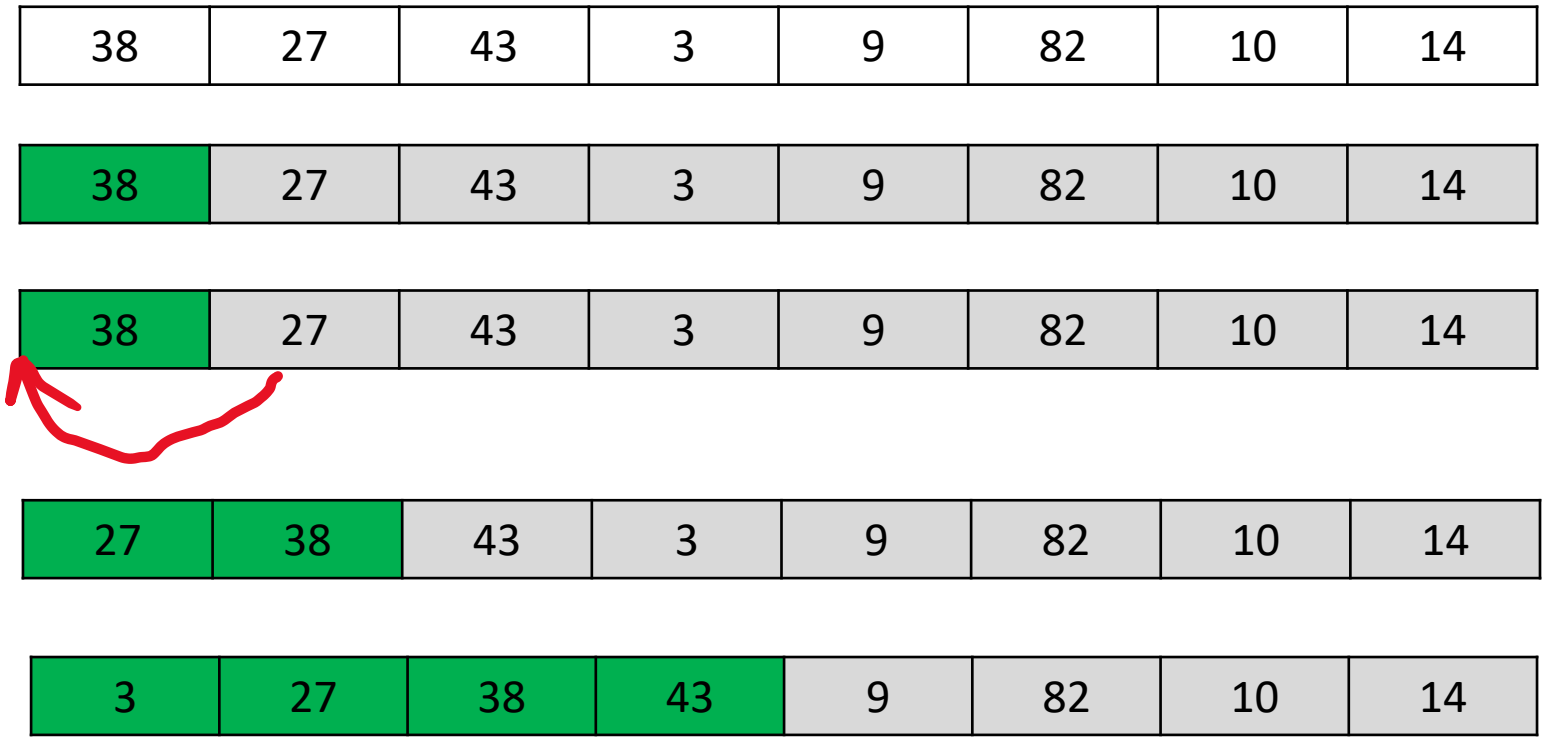
Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



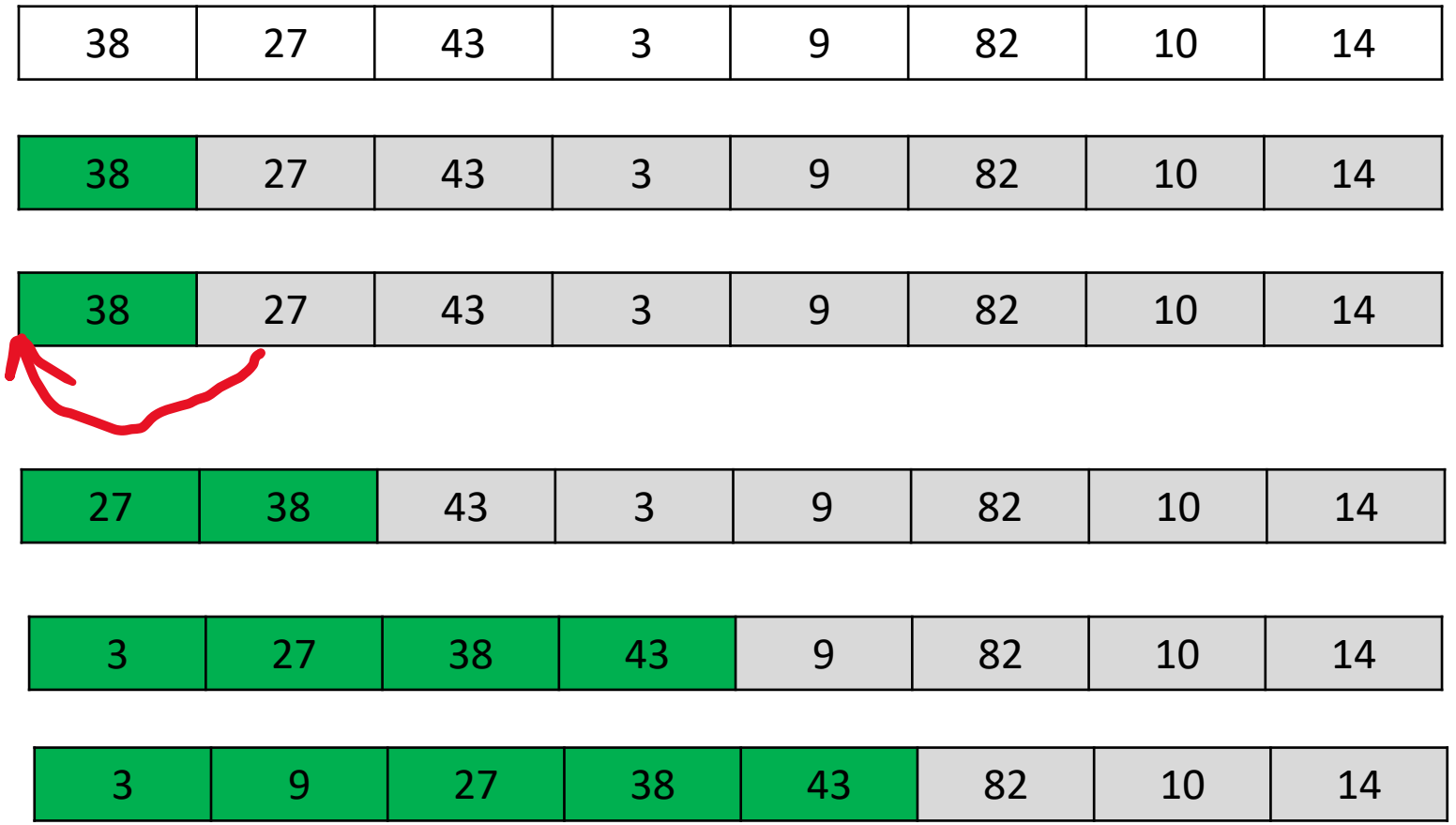
Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



Insertion Sort

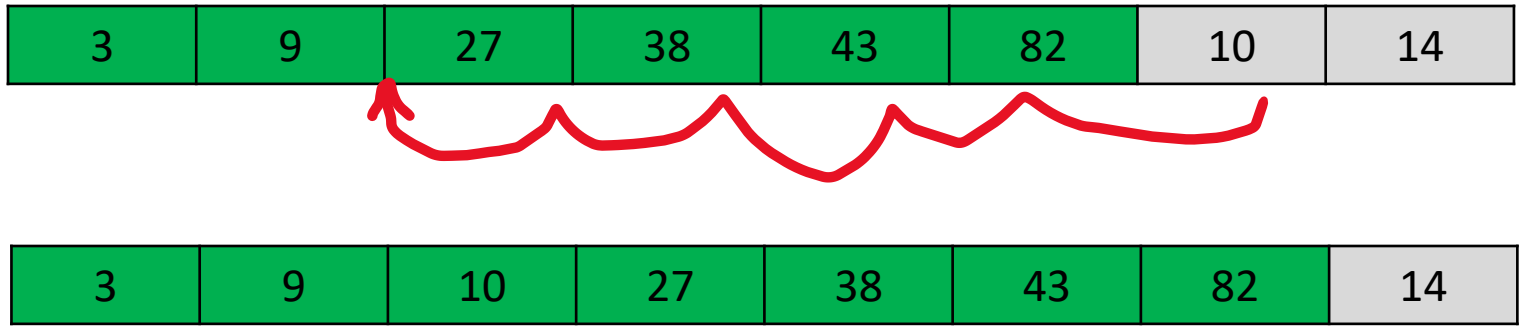
We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

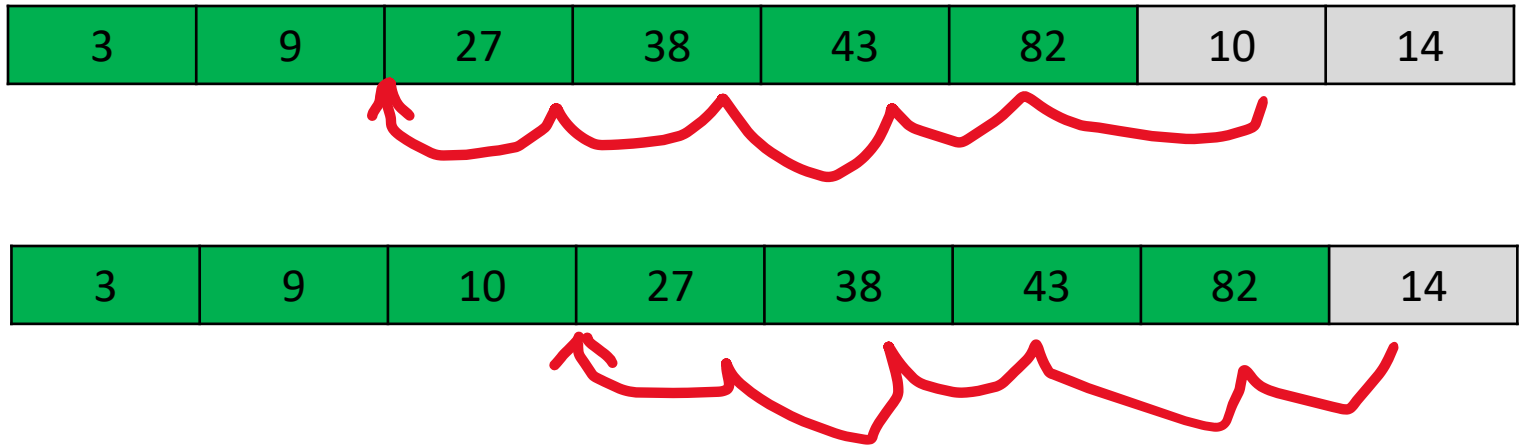
Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section



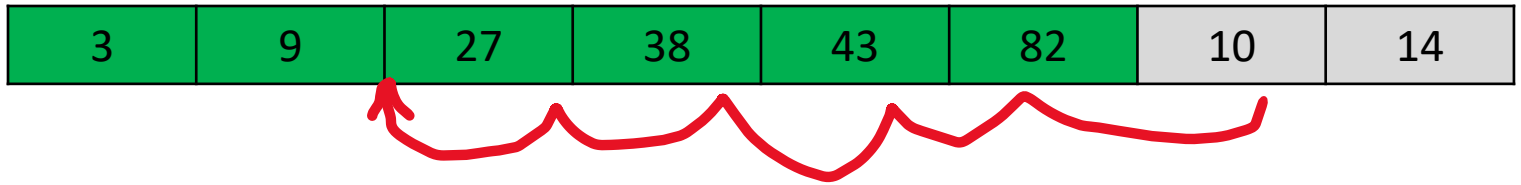
Insertion Sort

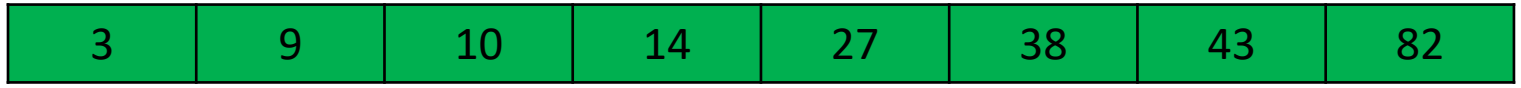
We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

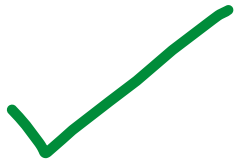


Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section

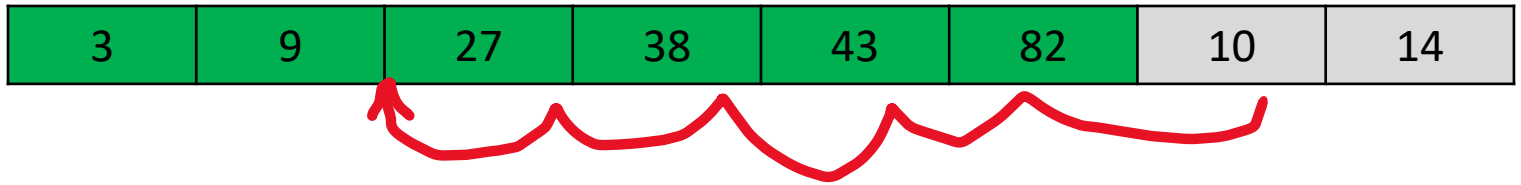


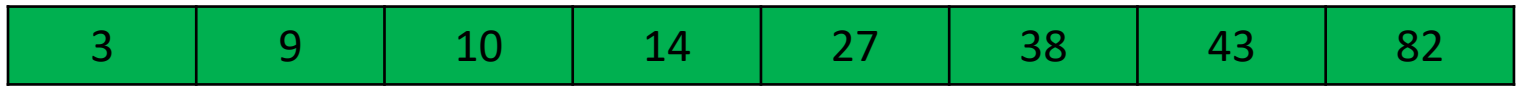


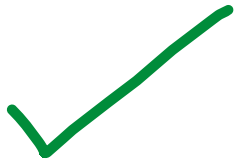


Insertion Sort

We divide our array into two sections. A sorted section, and an unsorted section. We iterate through the array, and for each iteration, we move one element from the unsorted section to the sorted section







Running time: $O(n^2)$

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

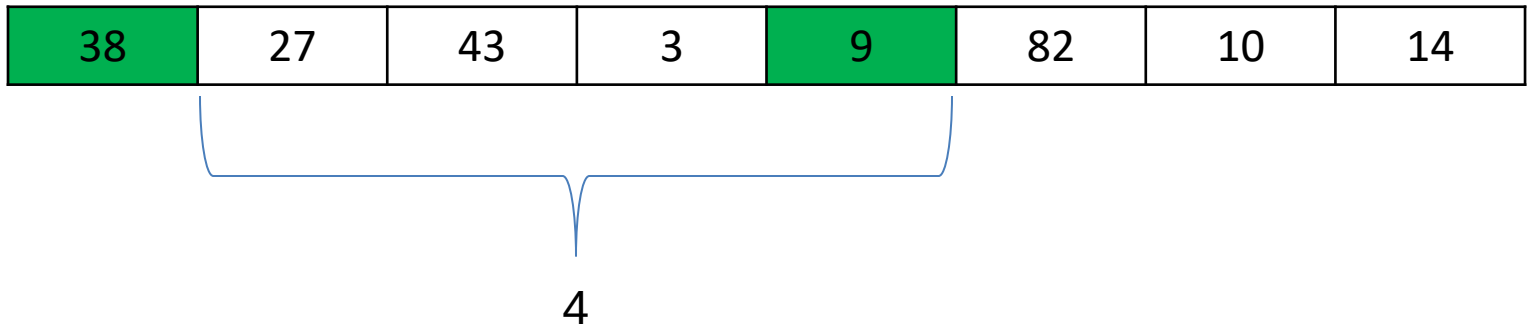
38	27	43	3	9	82	10	14

N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

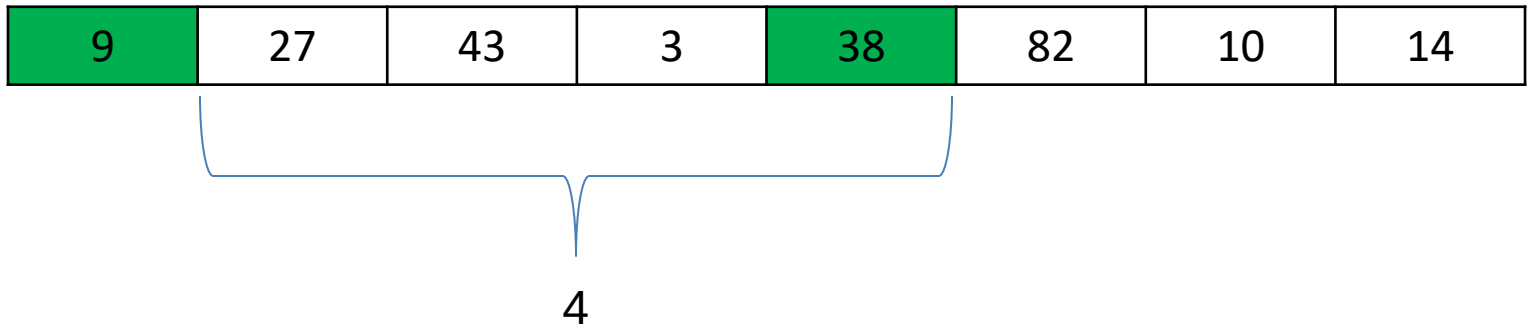


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

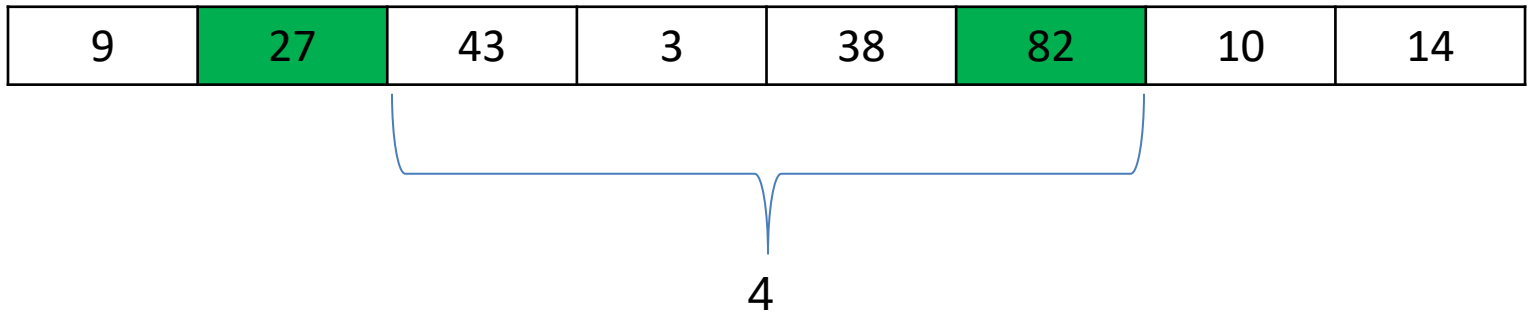


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

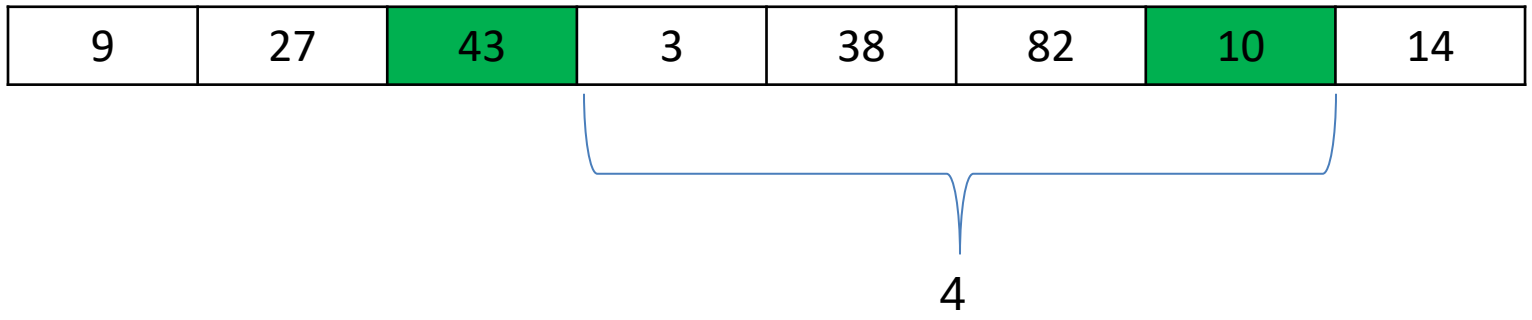


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

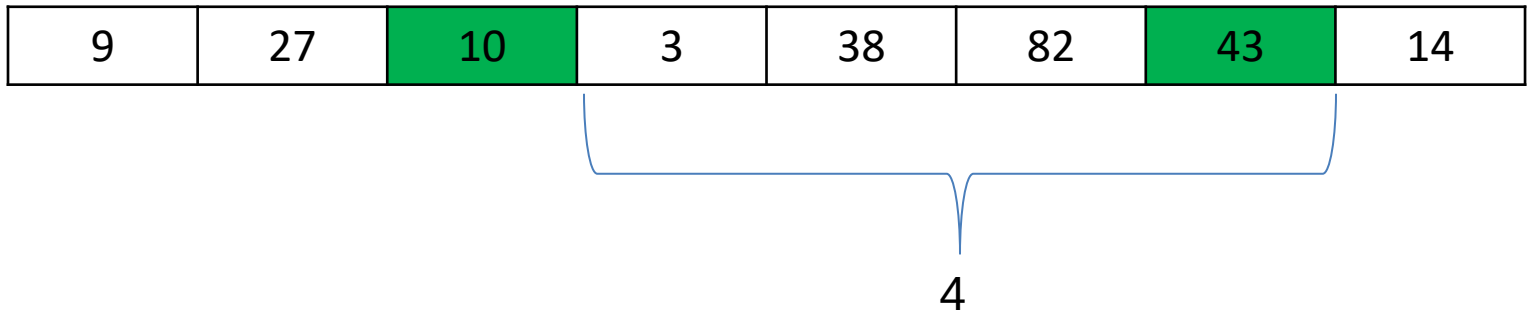


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

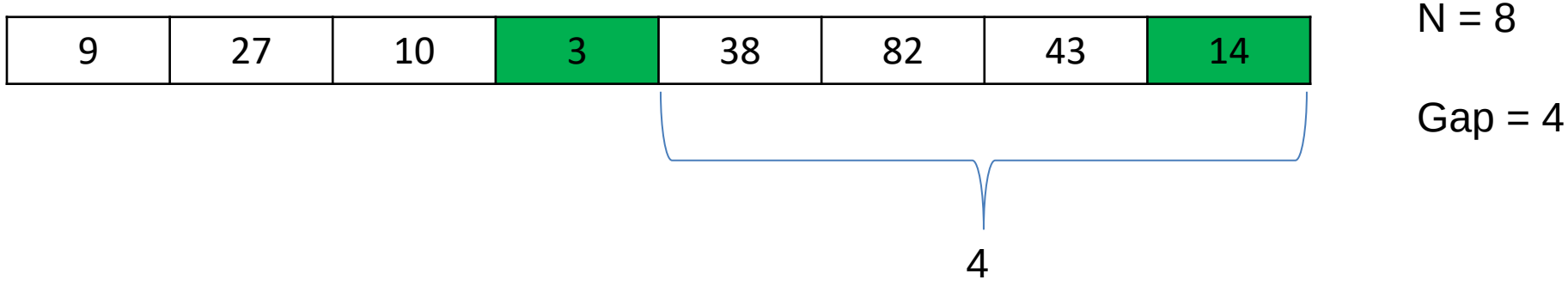


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

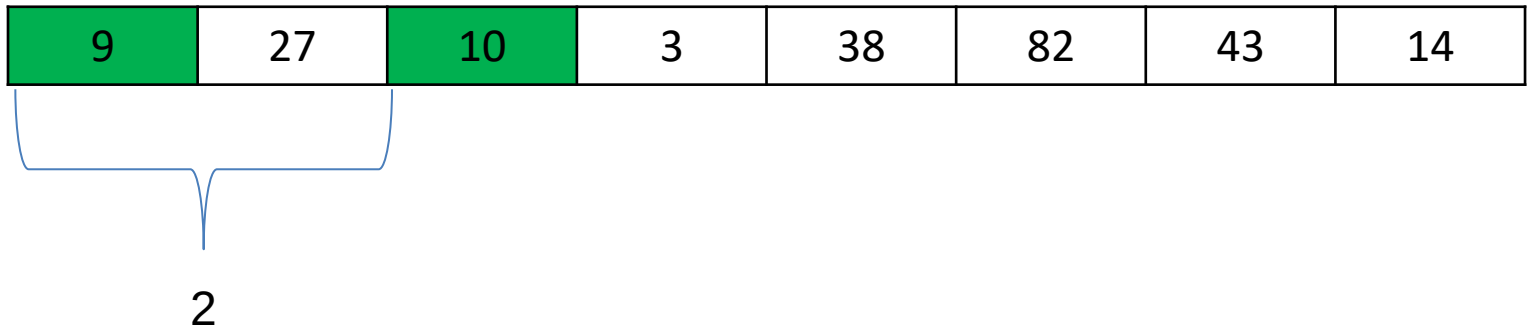


N = 8

Gap = 4

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



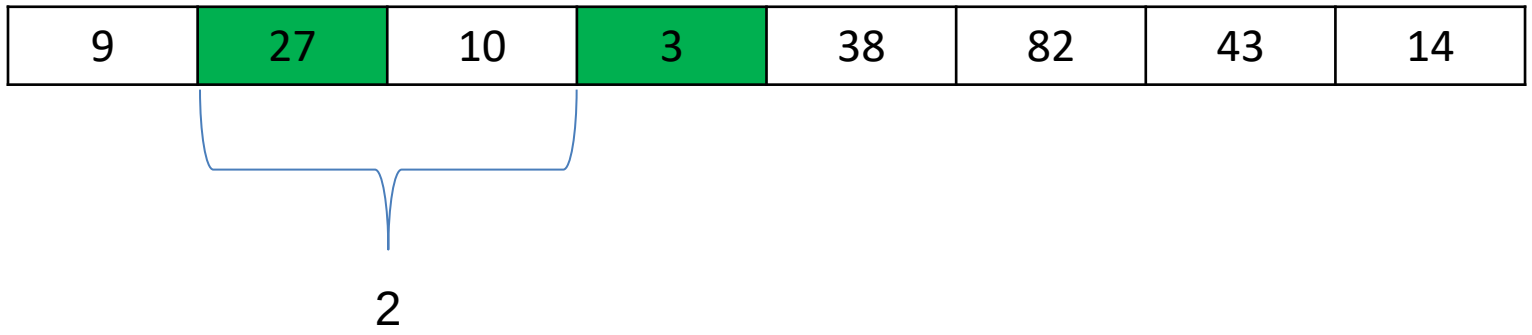
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



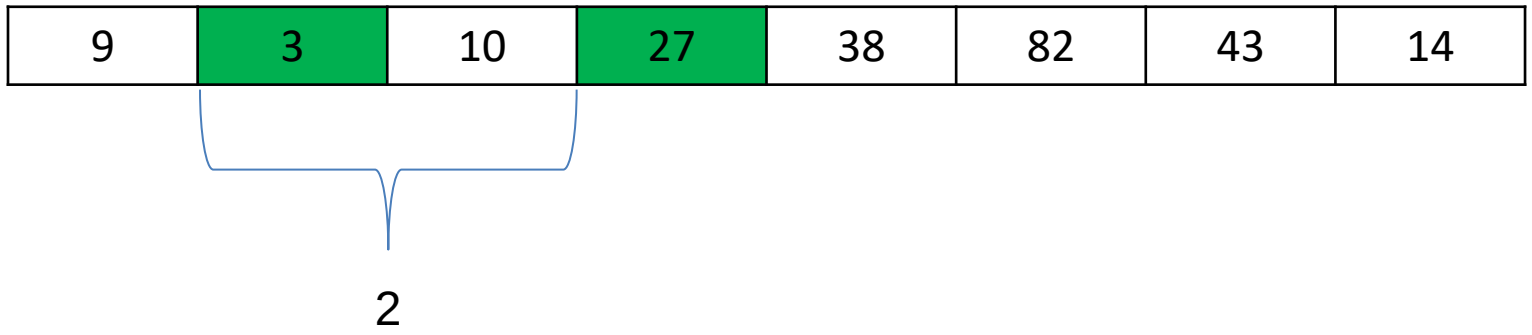
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



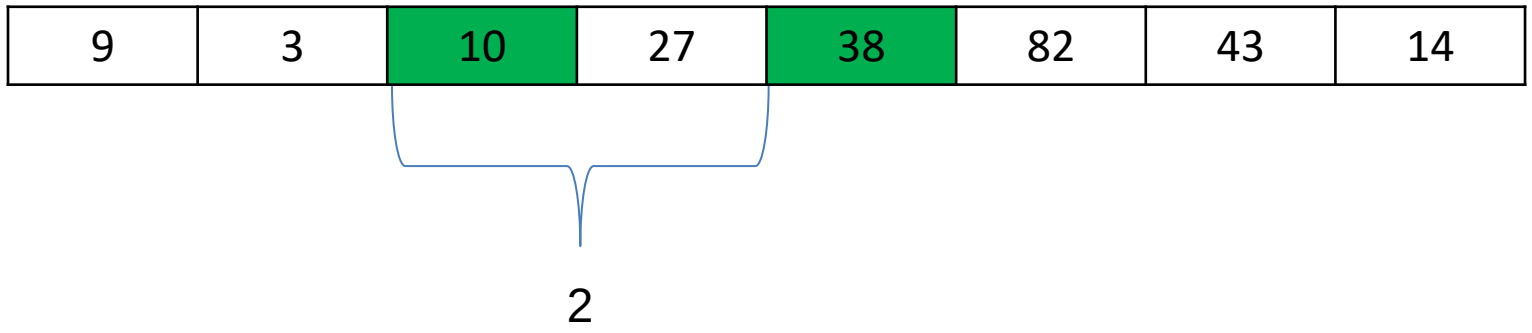
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

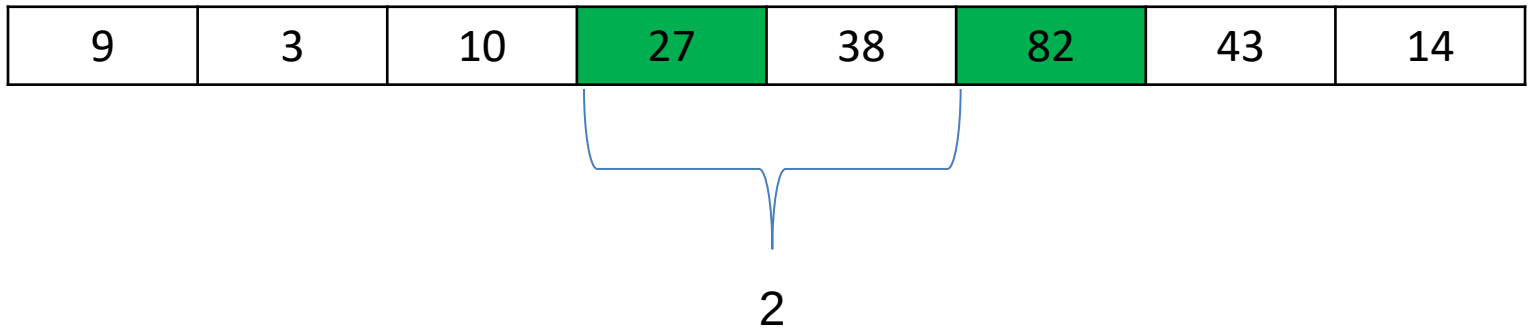


N = 8

~~Gap = 4~~
Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



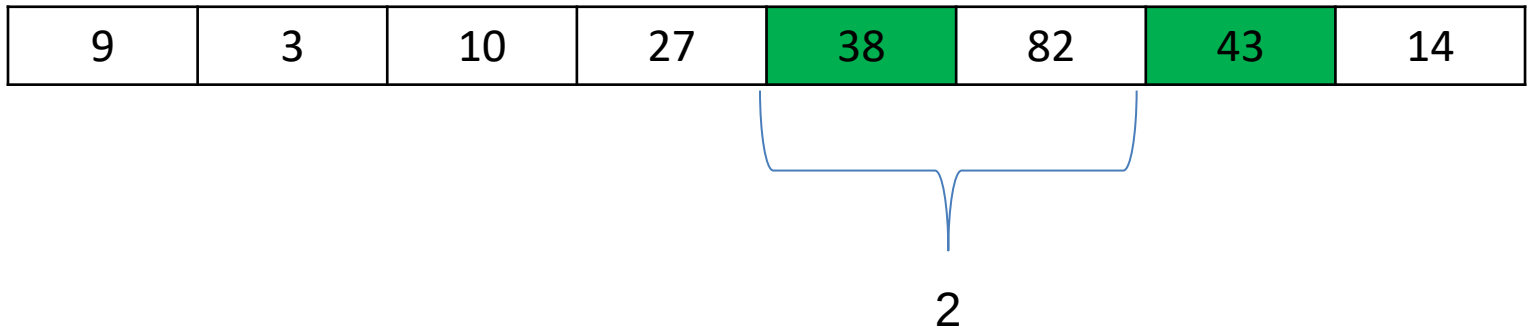
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



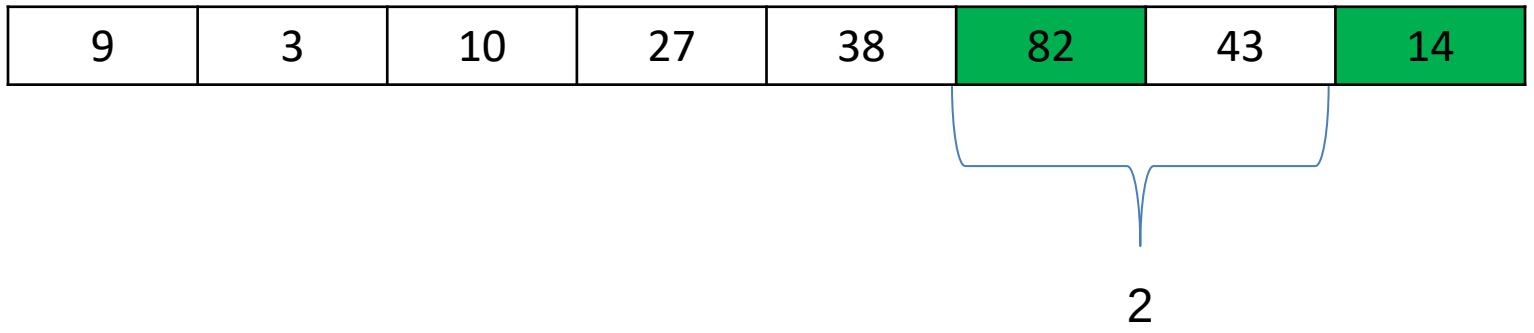
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



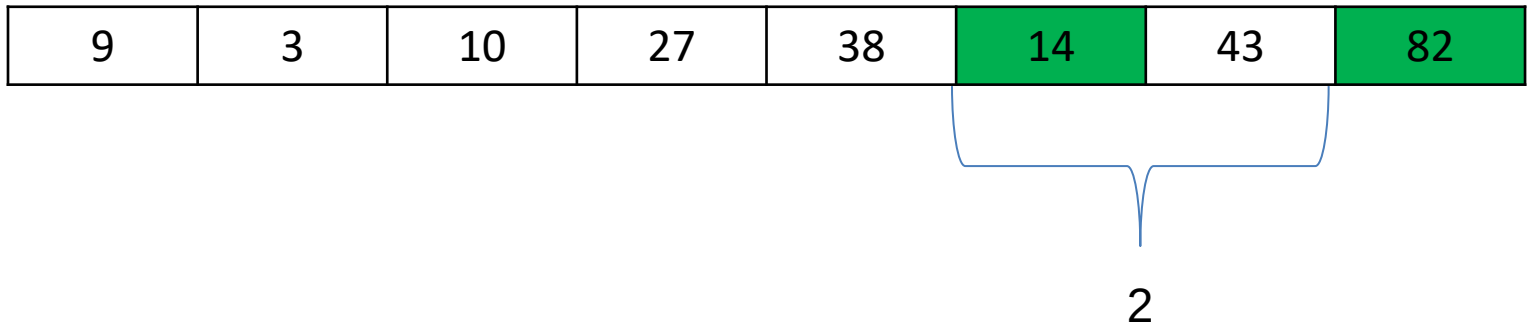
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



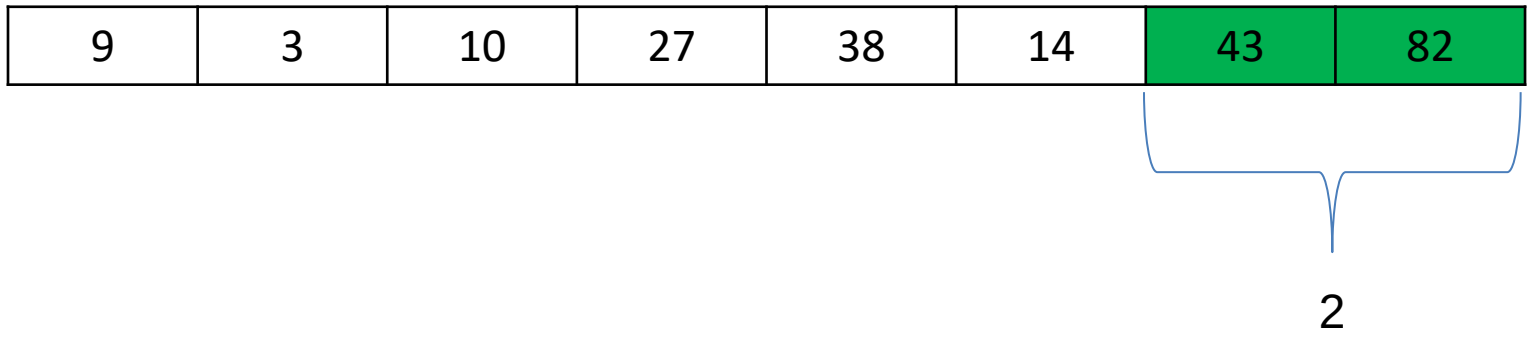
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



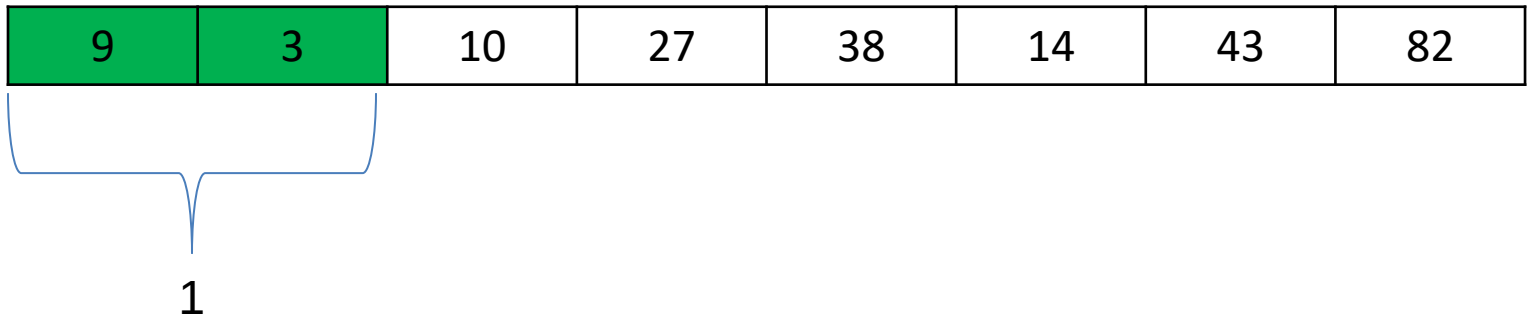
N = 8

~~Gap = 4~~

Gap = 2

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

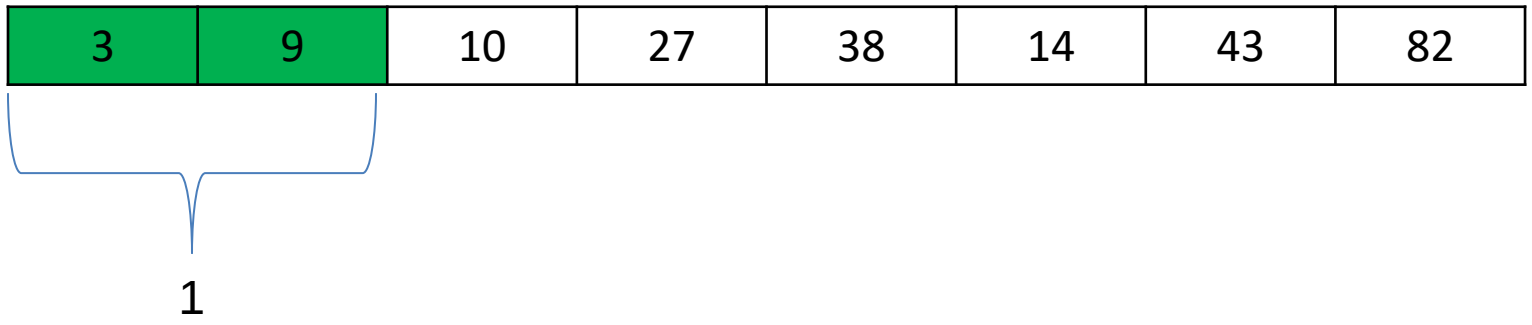


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

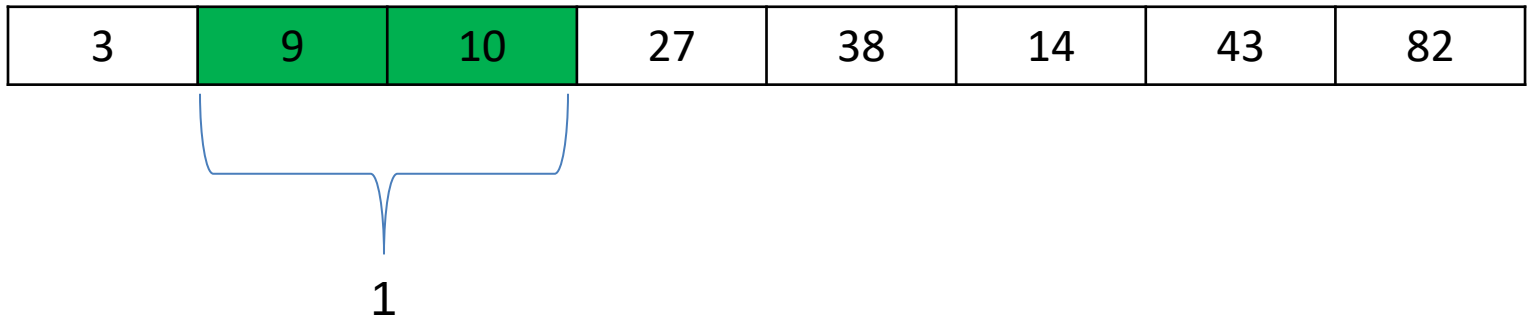


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

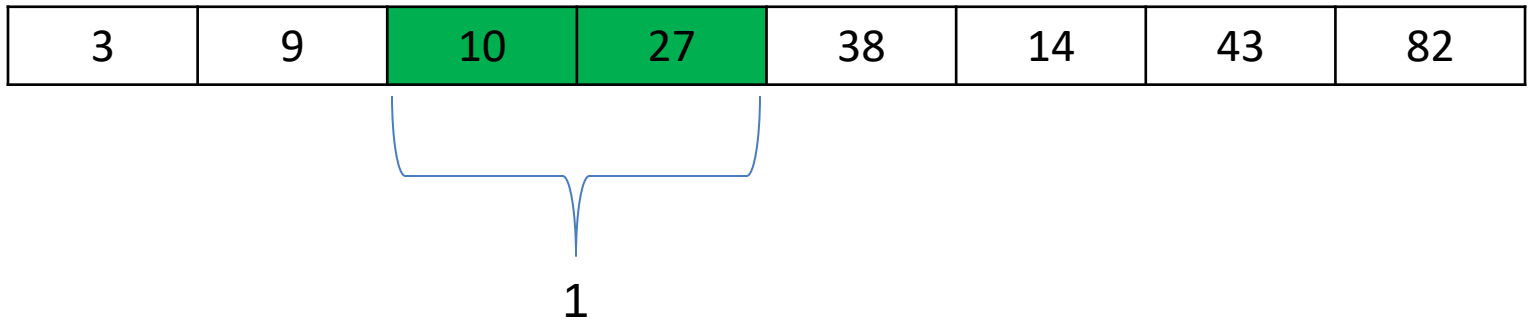


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

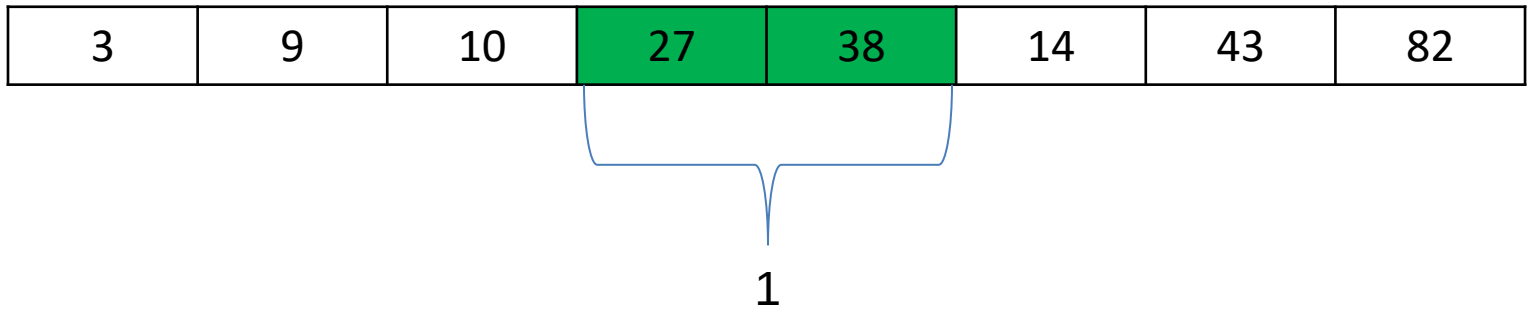


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

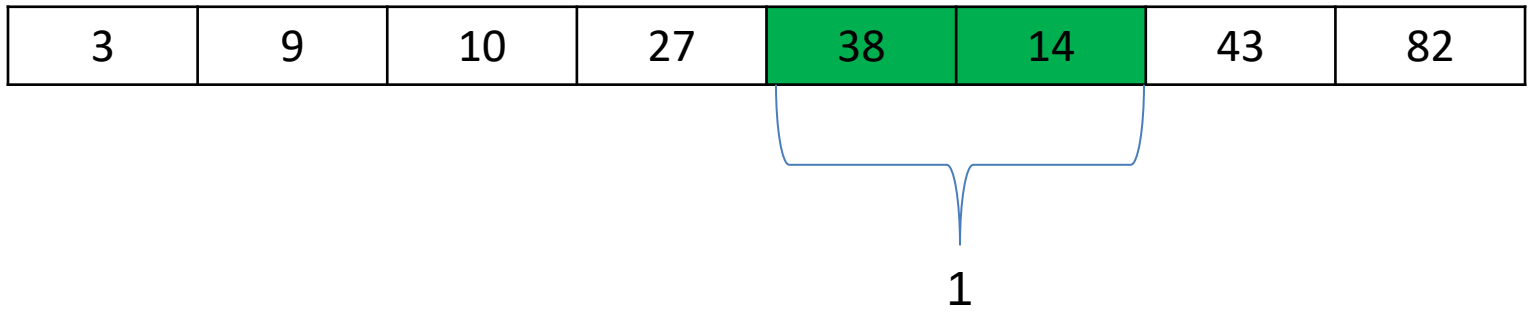


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

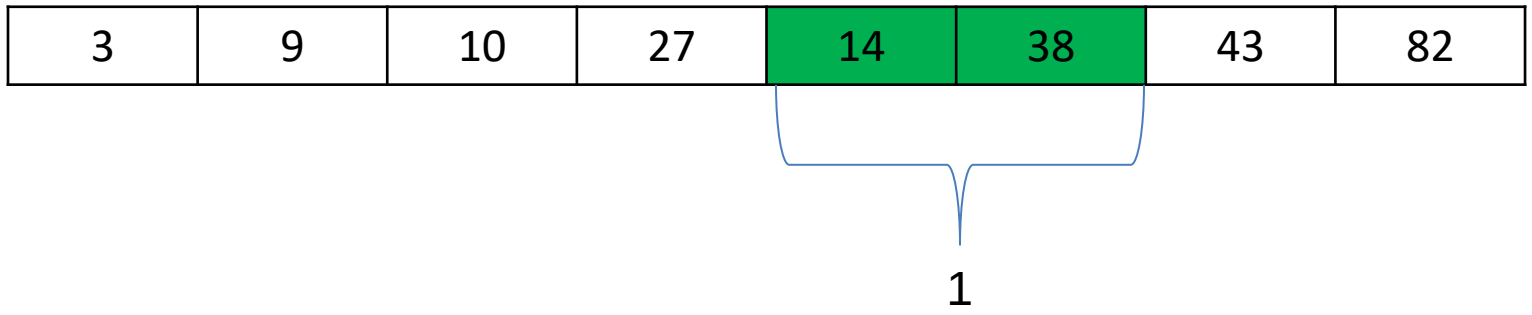


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

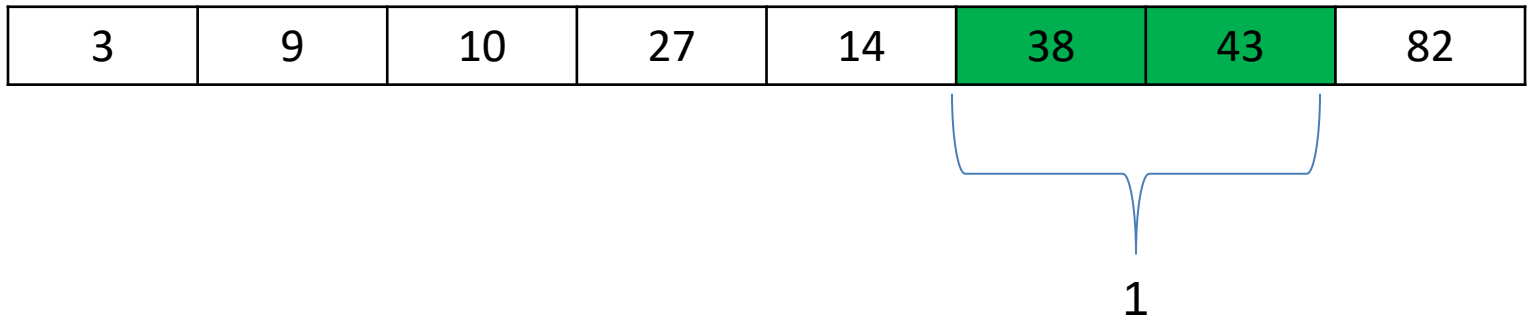


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

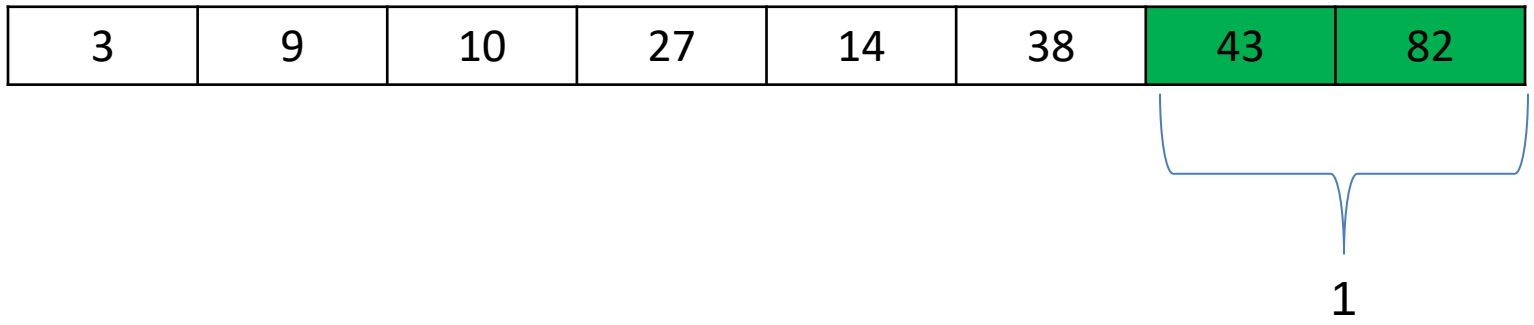


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



N = 8

~~Gap = 4~~

~~Gap = 2~~

Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

3	9	10	27	14	38	43	82

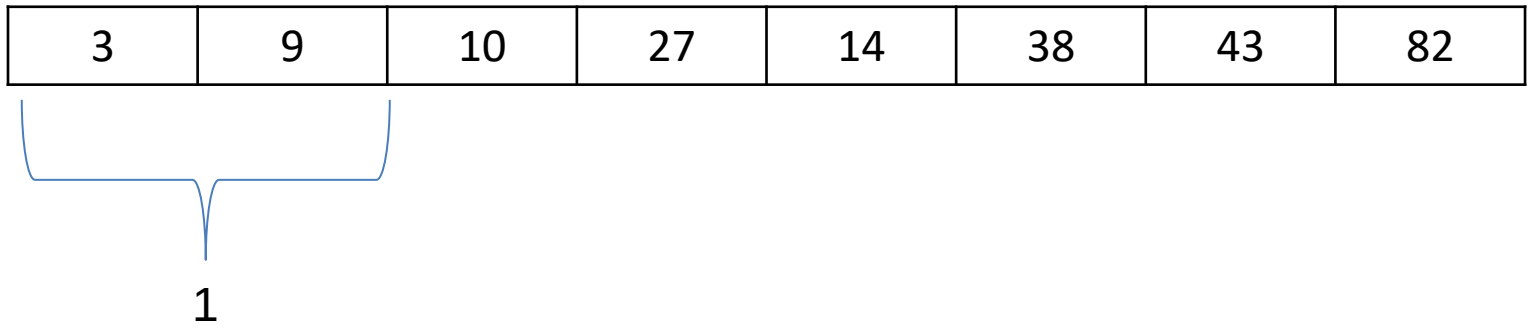
N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

(do it again ??)

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

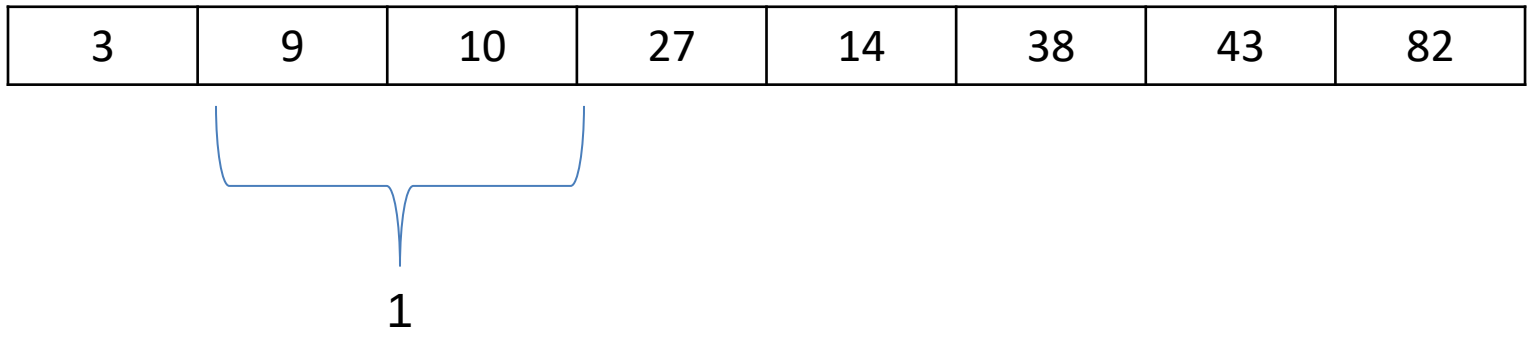


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

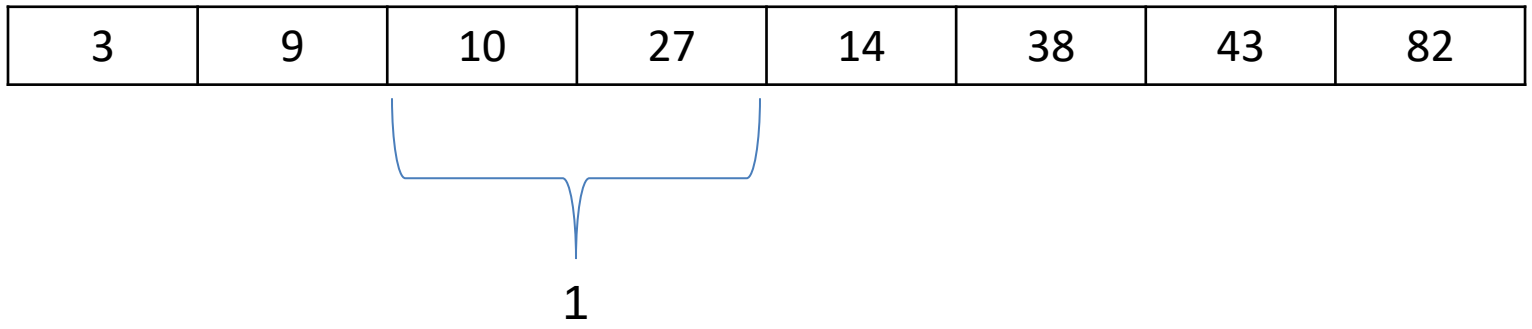


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

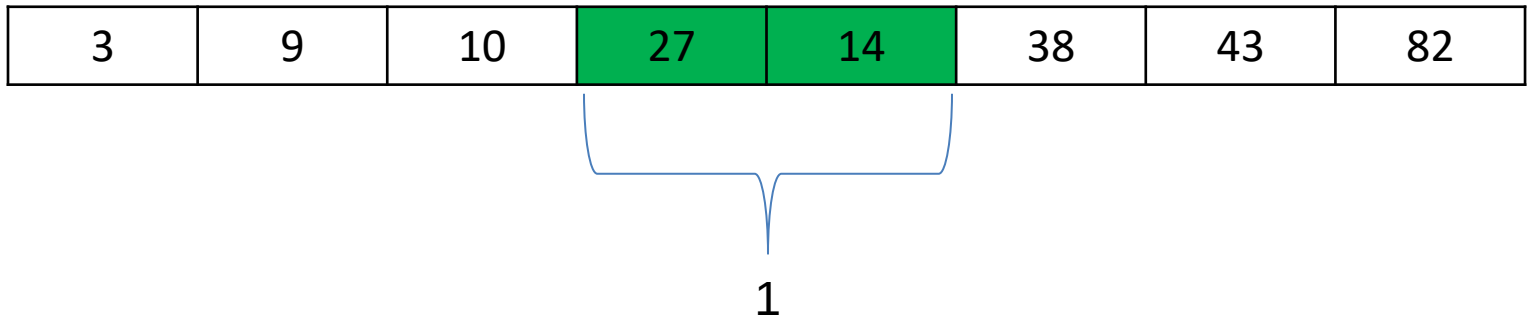


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

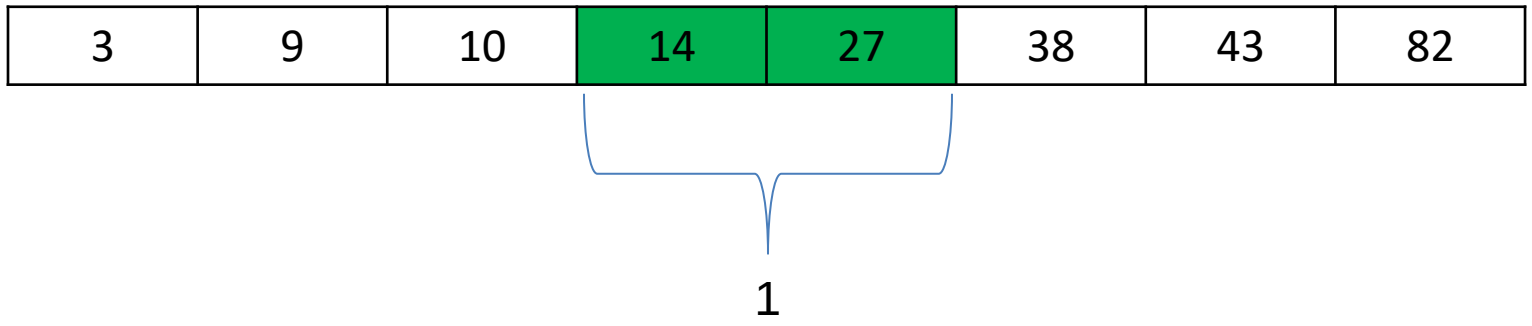


N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.



N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1

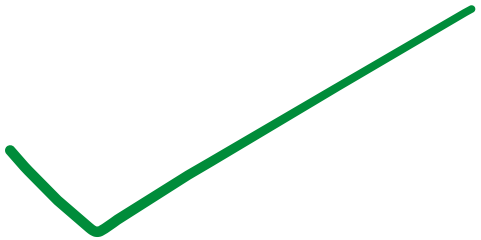
Shell Sort

Compare items that are distant from each other. After each iteration, decrease the gap size.

3	9	10	14	27	38	43	82

N = 8

~~Gap = 4~~
~~Gap = 2~~
Gap = 1



Running time: $O(n^2)$

Cocktail Shaker Sort

Double Sided Bubble Sort

https://en.wikipedia.org/wiki/Cocktail_shaker_sort

Running time: $O(n^2)$

Does anyone have any ideas for a very bad sorting algorithm, but still works?

Does anyone have any ideas for a very bad sorting algorithm, but still works?

If we are really lucky, our algorithm is insanely fast

If we are really unlucky, our algorithm will never finish

Bogo Sort (stupid sort) randomly shuffles the array until its sorted

```
while not sorted(array):
        shuffle(array)
```

Running time: O(pain) if we don't keep track of permutations checked

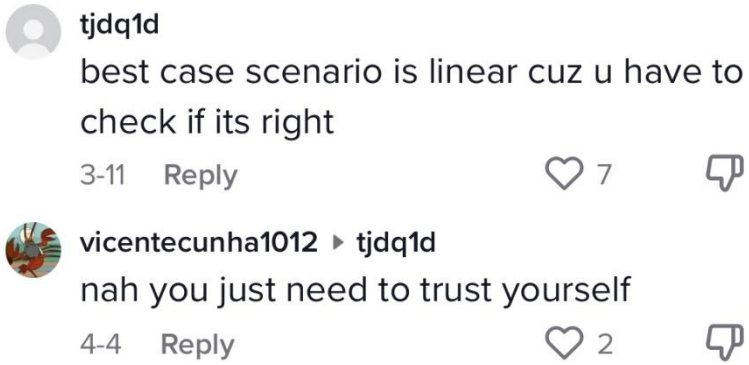
O(n!) if we keep track of permuations

Bogo Sort (stupid sort) randomly shuffles the array until its sorted

```
while not sorted(array):

        shuffle(array)
```

Best case scenario, this is the most efficient sorting algorithm!



Running time: O(pain) if we don't keep track of permutations checked

O(n!) if we keep track of permutations

This sorting algorithm is a joke, please don't take this one seriously…

Sorting Algorithms Visualized

https://youtu.be/kPRA0W1kECg