SORTING

The sort() method sorts the list ascending by default. Arranging a list in ascending or descending order.

INPUT:	OUTPUT:
cars = ['Ford', 'Volvo', 'BMW'] cars.sort() print(cars) cars.sort(reverse = True) print(cars)	['BMW', 'Ford', 'Volvo'] ['Volvo', 'Ford', 'BMW']

0 1 2 7 8 ➔ ascending order
8 7 2 0 1 ➔ descending order.

In most of the case sorted will be in ascending order.

In list a = [2 1 7 8 0]; `a.sort()`, will sort the array in ascending order, by default. And if we use `a.sort(reverse=True)`, the list will be sorted in descending order.

We can take 2 steps for the reverse order by first using `a.sort()` to get the list in ascending order and then using `a[::-1]` to reverse the list which will give us the list in descending order.

`a.sort()` → will sort the list in ascending order
`a.sort(reverse=True)` → will sort the list in descending order

Most of the work can be done with the above two statements, but if you come across a place of finding a sorting issue where you have to know the algorithm of things are sorted!

All the elements in the list should be of same data types.

If we don't have to use sort method, how can we sort the list? **Sorting Algorithms**.

Sorting Algorithms

Sorting Algorithms are methods of reorganizing a large number of items into some specific order such as highest to lowest, or vice-versa, or even in some alphabetical order.

Types:

- Selection Sort
- Bubble Sort
- Quick Sort
- Merge Sort
- Insertion Sort
- Heap Sort
- Radix Sort
- Bucket Sort

Time Complexities of Sorting Algorithms:

Algorithm	Best	Average	Worst
Quick Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)
Merge Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)
Heap Sort	Ω(n log(n))	Θ(n log(n))	O(n log(n))
Radix Sort	Ω(nk)	Θ(nk)	O(nk)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)

ASCII Values:

>>> ord (“a”)
97
>>> ord (“b”)
98
>>> ord (“A”)
65

Every key on our keyboard has an integer value. Ex: [b, c, B, A]

So, for a list of “str”, it will be sorted based on their ASCII value.

ASCII is used as a method to give all computers the same language, allowing them to share documents and files. ASCII is important because the development gave computers a common language.

ASCII is used as a method to give all computers the same language, allowing them to share documents and files. ASCII is important because the development gave computers a common language.

Algorithm behind Sorting:

For A = [5, 3, 2, 6, 4, 1].

Step 1: Find the element with the least (minimum) value in A.

Step 2: Put the minimum element in B

Step 3: Remove the minimum element from A.

We will repeat these 3 steps until we get our sorted array.

This type of sorting is called **Selection Sort**.

In A = [1, 5, 7, 8, 10, 4], min_ele = 1 and min_ele_idx = 0. The min_ele is already at 0^{th} index, so we will not make any changes to it.

Now, for [5, 7, 8, 10, 4], I have to check for the min_ele and finding that 4 is the min_ele which is at n-1 position, I have to swap it with the element in the 1^{st} index place. So,

[5, 7, 8, 10, 4] = [4, 7, 8, 10, 5]

Now for [7 8 10 5], we will check for the min_ele value and we will swap the element in the 2^{nd} index place with the min_ele. This continues till the list is sorted in ascending order.

We can swap the element by, **A[i], A[min_ele_idx] = A[min_ele_idx], A[i]**

[1, 5, 3, 7, 8]

For i = 0, we will swap the element at 0^{th} element. Res = [1, 5, 3, 7, 8]

For i = 1, the value is 5, min_ele_idx = 3, which will be sorted by the element in the 1^{st} index. Res = [1, 3, 5, 7, 8]

For i = 2, value = 5 and for i = 3 & 4 as well, the values are already sorted. So, the finally output would be

[1, 3, 5, 7, 8]

A tea cup and a coffee cup. To swap them, us a third cup (temp). put tea in temp, then pour coffee from its cup to the empty cup which previously has tea and then fill the empty cup which had coffee with tea.

Simple code for this would be:

```
a = 5
b = 10
print("before: ",a, b)

temp = a
a = b
b = temp
print("after: ", a, b)
```

Shorter version for this would be

```
a = 5
b = 10
a, b = b, a
print("before: ",a, b)
```

There is no need for temp in Python.

CODE:

```
def selection_sort(A):
    # [1, 8, 4, 2, 3, 9]

    n = len(A)
    for i in range(n):
        min_ele = A[i]
        min_ele_idx = i
        for idx in range(i, len(A)):
            if A[idx] < min_e:
                min_ele = A[idx]
                min_ele_idx = idx
        A[i], A[min_ele_idx] = A[min_ele_idx], A[i]
    return A

if __name__ == "__main__":
```

```
    list1 = [5, 4, 2, 3, 1]
    list2 = ["A", "a", "b", "B"]
    print(selection_sort(list1))
    print(selection_sort(list2))
```

OUTPUT:

[1, 2, 3, 4, 5]

['A', 'B', 'a', 'b']

Time Complexity:

For i = 0, n

i = 1, n – 1

i = 2, n – 2

i = 3, n – 3

:

:

:

:

i = n, n – (n – 1)

this is like, n + (n - 1) + (n - 2) + (n - 3) + ……3 + 2 + 1 = o (n)

Space Complexity would be O (1)

Bubble Sort:

For an array [5, 1, 8, 3, 4, 7]

Between, the first two elements in the array, compare them and let the least value be at the fist followed by greater value. It pushes the heavy element to the end or towards the right.

Step 1: Compare 5 & 1. Since 5 < 1 = False, they will swap position in array.

So, res = [1, 5, 8, 3, 4, 7]

Step 2: Compare 5 & 8, since 5 < 8 = True, no need to swap. res = [1, 5, 8, 3, 4, 7]

Step 3: Compare 8 & 3, 8 < 3 = False, they will swap position in array. res = [1, 5, 3, 8, 4, 7]

Step 4: Compare 8 & 4, 8 < 4 = False, they will swap position in array. res = [1, 5, 3, 4, 8, 7]

Step 5: Compare 8 & 7, 8 < 7 = False, they will swap position in array. res = [1, 5, 3, 4, 7, 8]

After these 5 steps, the heaviest element at the end of the array, res = [1, 5, 3, 4, 7, 8]

Now, we will repeat the same steps again for [1, 5, 3, 4, 7, 8].

CODE:

```
def bubbleSort(A):
    n = len(A)
    for i in range(n):
        for j in range(0, n-1):
            if A[j] > A[j + 1]:
                A[j], A[j + 1] = A[j + 1], A[j]
    return A

print(bubbleSort([5, 1, 8, 3, 4, 7]))
```

INPUT: [5, 1, 8, 3, 4, 7]

OUPUT: [1, 3, 4, 5, 7, 8]

Time Complexity: O(n)

Space Complexity: O(1)

Selection sort has a smaller number of swaps then Bubble sort.

https://www.geeksforgeeks.org/sorting-algorithms/

https://www.w3schools.com/python/ref_list_sort.asp

https://www.upgrad.com/blog/sorting-in-data-structure-with-examples/