

For duplicated elements they are sorted by their index number. First comes first then other index by numbers.

Out of Place Sort: If I take all of the books and throw them on the floor basically I need equal amount of storage for my data structure

In place sort If I leave data structure intact just shuffle things in my data structure

Stable Sort: Order of duplicates is unchanged.

Unstable sort: Order of duplicates may change*

Complexity of Sorting algorithms:

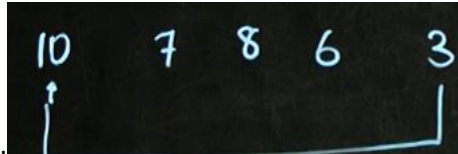
Worst case: Worst cases often if a list is reverse sorted. If you want to sort lowest to highest, if you're given a list highest to lowest, that's often the worst case for sorting algorithm.

Average Case: Randomly given numbers and you try to sort.

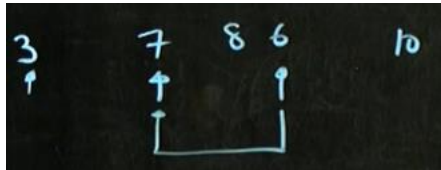
Best Case: we'll see soon*

Sorts 2 Selection Sort

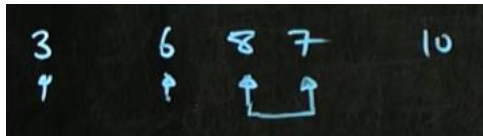
We start from position 0. We are going through the list and trying to find smallest number of the list. And



swap it with index of Zero.



Now we begin. Then we search again we find 6 then put 6 to index 1 places of 7



Now we find 7 and we put 7 to index of 8



We do that at the end of the list.

Complexity of Selection Sort

$$(n-1) + (n-2) + (n-3) + (n-4) \dots$$

$$= \frac{n(n-1)}{2}$$

TIME COMPLEXITY: $= O(n^2)$

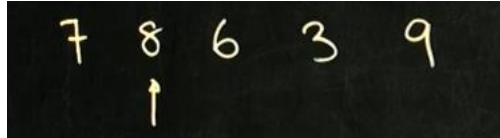
For the best case, average case or worst case we don't know actually where is the small element. Even in best case we have to go through to all list. And this sort is Inplace means we don't have to make a copy of a list* and put back in.

Sorts 3 Insertion Sort

It's a little bit like a Selected sort.

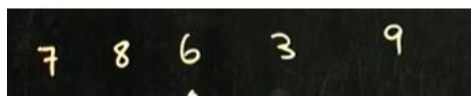
We ignore first element and we take second of element in list then we say is this second element is less than first element ?

For our list Is 8 less than 7 ? If it is we swap them if not we leave it in its place.



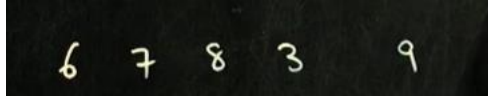
7 8 6 3 9
↑

Then we come to 6 Is 6 less than 8 yes also less than 7 yes ? lets swap it into first index



7 8 6 3 9

We got



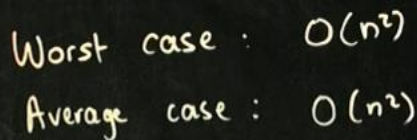
6 7 8 3 9

Now we come to 3 Is 3 less than 6 yes also 7 yeah Also 8 yeahh lets take him to First index. What a lovely number you are sweetie



3 6 7 8 9
↑

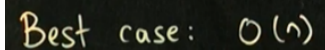
Then we come to end. We make same things for 9 but Nine is biggest number so he stays on its place.



Worst case : $O(n^2)$
Average case : $O(n^2)$

Time complexity of Insertion Sort

What happens if list is already sorted? In sorted list Insertion sort Best case



Best case: $O(n)$

For example u got dictionary or database they are already sorted and u wanna add something etc YOU DON'T have to resort elements so why u got $O(n)$

Insertion sort is typically used If you have already sorted list for example in a database you keep all keys and they are already sorted. You add to new things to database and you just quickly do insertion sort it keeps things sorted.

Insertion sort is stable

Insertion sort is Inplace.

```
def insertion_sort(list_):
    for index in range(len(list_)):
        value=list_[index]
        i=index - 1
        while i >= 0:
            if value < list_[i]:
                list_[i+1]=list_[i]
                list_[i]=value
                i=i-1
            else:
                break

a=[5,1,3,7]
insertion_sort(a)
print(a)
```

Code of insertion sort:

source:khanacademy

Sorts 5 Shell Sort

Shellsort is a variation of insertion sort

This sorting algorithm is devised by Donald L. Shell in 1959

Sometimes called as 'diminishing insertion sort'

Shell sort improves Insertion algorithm.

By breaking the original list into a number of smaller sublists, each sublist is sorted using the insertion sort. It will move the items nearer to the original index.

Insertion sort works efficiently in smaller list but not in big lists. So this Shell sort algorithm divides the lists smaller sub lists then it will sort sub list using insertion sort algorithm.

This algorithm can work on bubble sort algorithm or other algorithm*

This can be worked very well for medium sized arrays having elements up to a few thousands

Donald Shell saw with a problem of Insertion sort and tried to fix it in shell sort algorithm.

For insertion sort we have to move a lot of elements to make place for new element to be inserted in the sorted part of the array. For larger lists there would be a lot of element jumps, especially when we end up near the array elements that are small. To understand consider this example

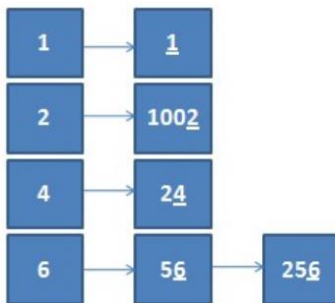
Sorts 11 Radix Sort

- It's a sorting algorithm that is used to sort numbers
- We sort numbers from least significant digit to the most significant digit.

	0	1	2	3	4
A	56	1002	24	1	256

1-First check units digit and sort(Birler basamagini kontrol et ve sort yap.)

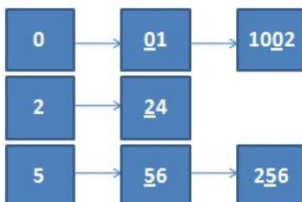
	0	1	2	3	4
A	5 <u>6</u>	100 <u>2</u>	2 <u>4</u>	<u>1</u>	25 <u>6</u>



	0	1	2	3	4
A	1	1002	24	56	256

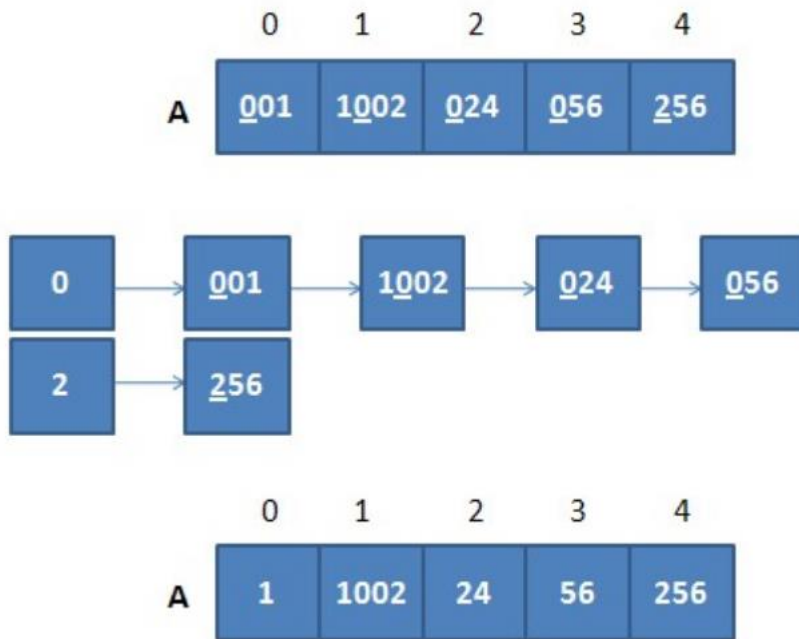
2-Second check tenth digit and sort(onlar basamagini kontrol et ve sort yap)

	0	1	2	3	4
A	<u>0</u> 1	10 <u>0</u> 2	<u>2</u> 4	<u>5</u> 6	2 <u>5</u> 6

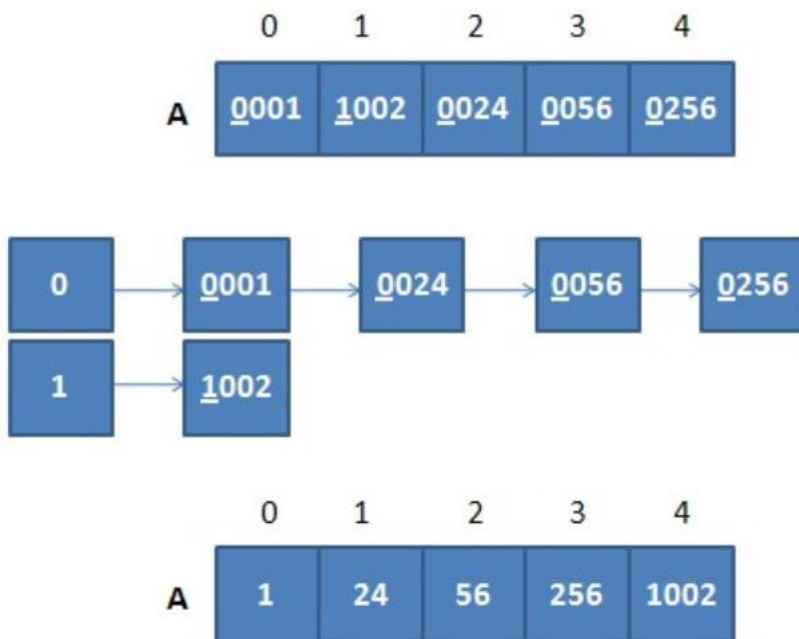


	0	1	2	3	4
A	1	1002	24	56	256

3-Check hundreds digit and sort(yuzler basamagini kontrol et ve sirala)

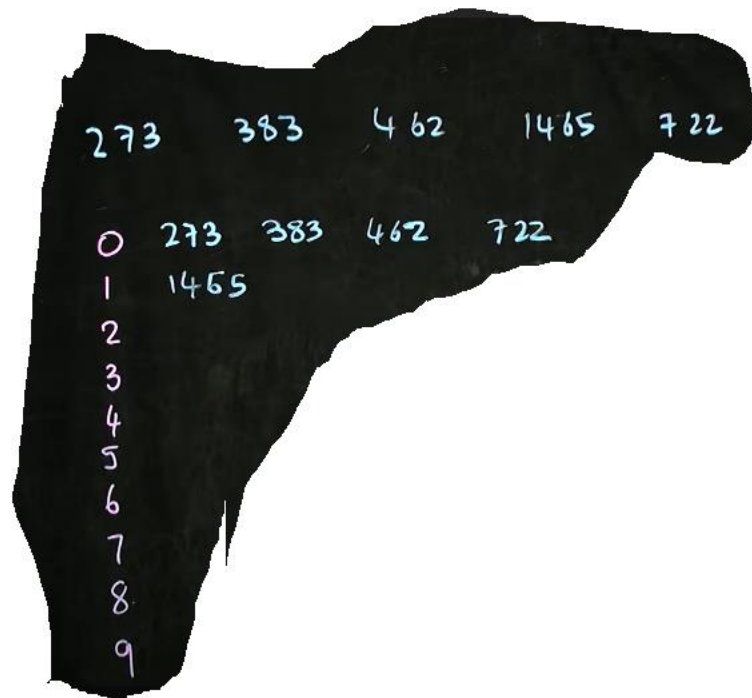


4-Check thousand digit and sort(binler basamagini kontrol et ve sort yap)



TIME COMPLEXITY OF RADIX SORT.

Time complexity of radix sort is based just length of the numbers.



For 1465, Its $4n$ because we are looking for 4 numbers. We look zeros tens the hundreds and thousands.

Burada time complexity $4n$ cunku 4 haneli sayinin birler onlar yuzler binler basamagina bakiyoruz.

In theory its super quick way to search for things its n complexity but in practice in not a very robust search you have to copy all of numbers and then copy them back out into the list and copy them back into the bins then copy them back out again into the list that means radix sort is not a good search of all that copying.our computers aren't really set up to move

data in that way its still used in mechanical sorting in terms of computing its not a robust sort.

Sorts 12 Sort Summary

	AV. COMPLEXITY	Stable	In place	NOTE
SELECT	$O(n^2)$	No	YES	
INSERT	$O(n^2)$	YES	YES	$O(n)$ in best case
SHELL	$O(n^{3/2})$	No	YES	
MERGE	$O(n \log n)$	YES	No	
QUICK	$O(n \log n)$	No	YES	$O(n^2)$ in worst case
RADIX	$O(n)$	YES	No	Very slow
HEAP	$O(n \log n)$	No	YES	

Select Sort: You select smallest thing in the list and you move it to front. It is an inplace sort but you can make it stable coding by additioning some code.

Insert Sort: You take one element and compare before it. And If it is smaller than the thing before it you swap it. And you go to the third element and you keep doing thing... It's not typically usefull unless you haven't had already sorted list. In which case the insert sort has complexity big O of N

Shell Sort: This is a modification of Insert sort. Instead of sorting each element adjacent element you take a gap and you sort elements that a gap distance apart. Open problem of computer science this gap how affects the average complexity of the shell sort. Shell sort is also not stable but it is in place.

Merge Sort: We divide things in a half we divide things in half. During that division we don't do any work. it's when we combining things back up together that we actually do the work. So the merge sort because of dividng in half $\log n$ complexity.

Quicksort: where we take pivot point and and we move everyting smaller than the pivot point to the left and everything larger than the pivot point to the right. We then go the midpoint to the left and we repeat. Everything small on the left and everything larger on the right. We divide in half so because of that we get a time complexity $n \log n$ complexity. Quick sort is not stable but it is in place sort. Remember If you choose either the largest thing or smallest thing as your pivot you gonna end up with

$O(n^2)$ in worst case

complexity. Quicksort it is inplace and because it is robust algorithm is really a sorting algorithm of choice. for most applications.

Radix sort: its is like mechanical sorter like you would use to sort mail and so you start by sorting the integers and then you sort by the tens you sort by the hundreds. And then by thousands so on... Radix sort is stable because you have to copy all of elements into the appropriate bucket and then copying back out again and radix sort is really slow because all of that copying. It is not typically used.

Heap sort: we take our heap maxi or min and we remove elements on the top of the heap. When we do that we put our last element to the heap we put that back in the top and then it has to trickle down and we compare that number with one half of the heap.