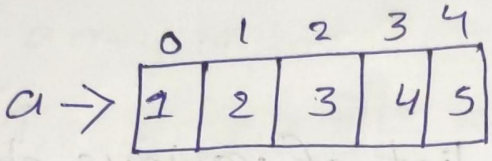
1. What is array? array differ from List? Types of arrays?

→ collection of similar datatype under a single variable

→ All elements are in continguous memory location.

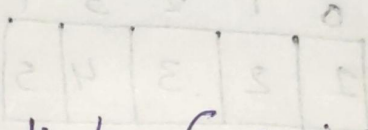
→ 

→ Indexing starts from 0 to n-1

Array	List
1. Similar data	1. Different types of data
2. User defined	2. builtin
3. easy to acess	3. not easy to acess
4. Huge amount of data	4. small amount of data
5. Less memory	5. more memory
6. Dynamic	6. static

Types of Arrays:-

- one dimensional
- Two dimensional
- Multi dimensional



*one dimensional:-

→ It stores a single list of various elements of similar data type

→ It represents multiple data in the form of list

→ It has only one dimensional

ex:-

```
import numpy as np
list=[100,200,300]
n=np.array(list)
print(n)
```

o/p: [100, 200, 300]

*Two dimensional:

-> set of Bielemental lists ([12,3],[6,7])

->It has a total two dimensional

ex:

```
from array import *
T=[[1,2,3],[6,7,8]]
print(T[0])
print(T[1])
```

o/p: [1, 2, 3]
[6, 7, 8]

2) Difference b/w Data type and Data structure?

Data Type	Data structure
→ What type of data it accepts	→ collection of data type
→ It holds single value	→ Multiple value
→ abstract implementation	→ concrete implementation
→ No involvement of time complexity	→ Involvement of time complexity
→ assign direct value to a variable	→ We need to perform some operations
→ Less data maintaing ex:- int, float e.t.c..	→ Huge data maintaing ex:- array, List etc..

* Data Structure
  i) Linear Data structure
  ii) Non-Linear Data structure

3) Explain Linear Search?

→ It is a sequential search used to search for an element within a list or array.

Steps :-

1) Compare 'key' element with every element of List.

2) If element found return the position of element.

Ex :-

0	1	2	3	4	5
10	27	1	4	5	100

key = 1

1 == 10 ✗

1 == 27 ✗

1 == 1 ✓ → Element found at index 2

Program

```
def linearsearch(arr, x):
    for i in range(len(arr)):
        if arr[i] == x:
            return
import array as arr
arr = [10, 20, 30, 60, 80]
x = 60
Print("The element is found at index", (linearsearch
(arr, x))
```

Output:-

The element is found at index 3

4) Explain Binary search?

→ It is used to search for a particular element in a sorted list or array

→ Arrangin data in sorting order

Steps:-

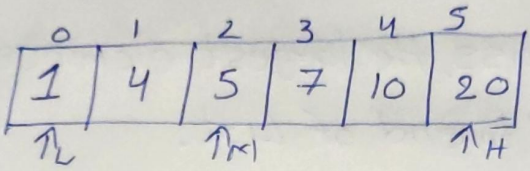
1) First sort the elements
2) Find mid element

Mid ⇒ (low + high)//2

3) compare 'key' element with mid value. if element found, Print message and stop
4) else if key > mid then we have to check right side list
5) else if key < mid, then we have to check left side list
6) else the element not found

ex:

10 7 4 1 20 5



Mid = (low + high)//2

⇒ 0 + 5//2

⇒ 2

key = 10

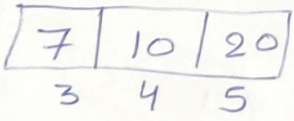
i) key == List[Mid]

10 == 5 ✗

ii) key > Lst[Mid]

10 > 5

#check Right side List



Mid ⇒ (3+5)//2

⇒ 8//2

⇒ 4

a) key == List[Mid]

10 == 10 ✓

Element at index 4 is found

5) Explain Bubble Sort?

→ In the every pass, the largest elements in list 'bubbles up' to end of List.

Steps:-

1) First element is compared with adjacent element.
2) if adjacent element is smaller than the first element, swap elements
3) Then second element is compared with its adjacent element.
4) repeat until end of List
5) Large element Move towards end of list
6) Repeat until you get the sorted list.

Ex:-

(I)

15 16 6 8 5

NS

15 16 6 8 5

15 6 16 8 5

4 comparisons

15 6 8 16 5

15 6 8 5 16

(II)

15 6 8 5 16

6 15 8 5 16

6 8 15 5 16

6 8 5 15 16

3 comparisons

(III)

6 8 5 15 16

6 8 5 15 16

6 5 8 15 16

2 comparisons

(IV)

6 5 8 15 16

5 6 8 15 16 => Sorted list

1 comparisons

6) Explain Merge Sort?

→ It works based on Divide & conquer rule

Steps:

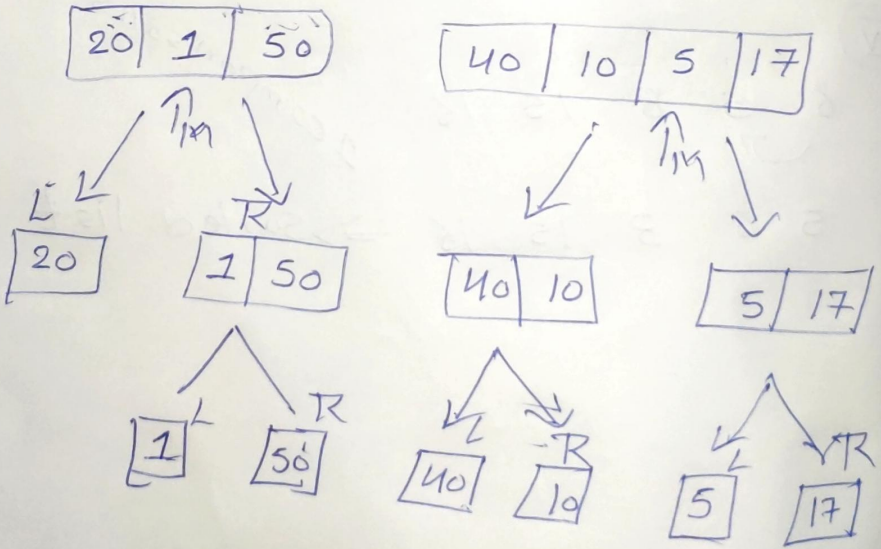
1) divide the list into sublist
2) Divide the sublist until the list contain single element
3) Merge elements until you get sorting list

Ex:

0	1	2	3	4	5	6
20	1	50	40	10	5	17

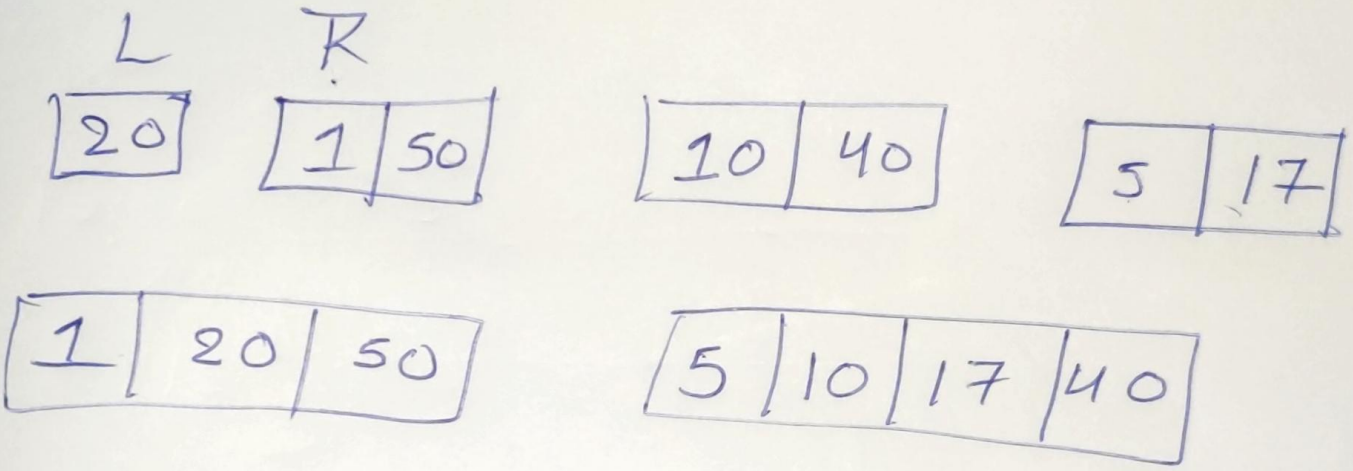
LSL ← | IM | → RSL

Mid ⇒ 6/2 ⇒ 3



i) Left < Right = Left

ii) Left > Right = Right



$\Rightarrow$ 1 5 10 17 20 40 50