Day 2

1. Sorting -> Sort, Sorted -> ASC and DEC
    - Bubble Sort
2. Searching - index
    - Binary Search
3. Stack -> ADT -> Abstract Data Type
    A. What is a Stack
    B. Properties
    C. Implementation
    D. Application of stack

In [3]:

```
li = [1,5,8,3,2,0,-1,55,125,99]
```

In [4]:

```
print(li)
```

```
[1, 5, 8, 3, 2, 0, -1, 55, 125, 99]
```

In [5]:

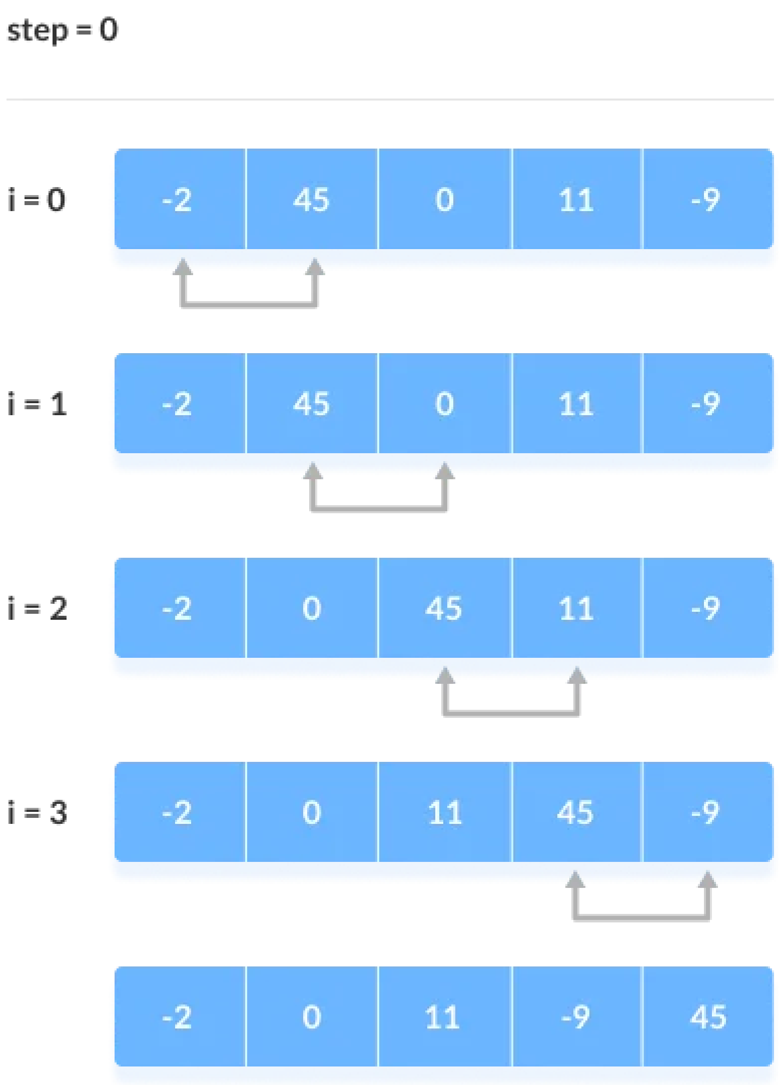
```
sorted(li)
```

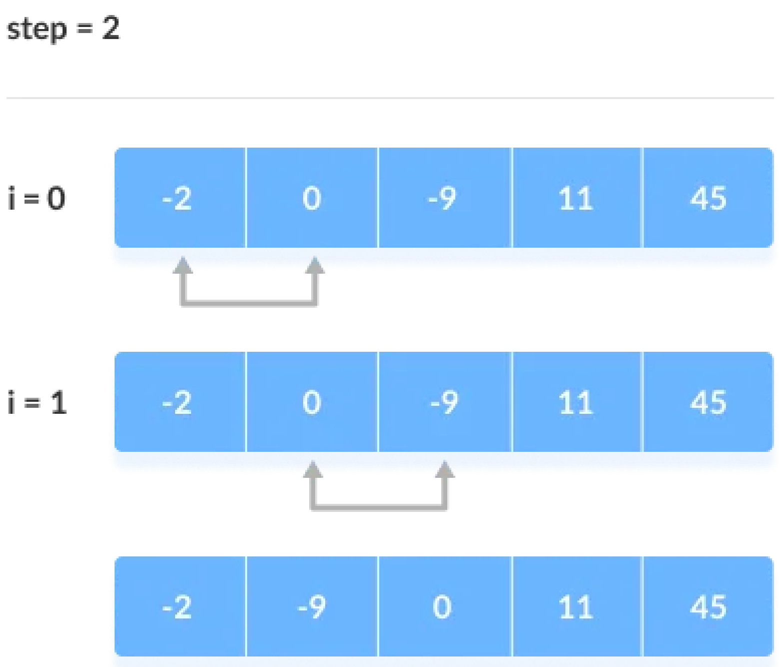
Out[5]:

```
[-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
```

Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.


step = 0
i = 0
-2 45 0 11 -9
i = 1
-2 45 0 11 -9
i = 2
-2 0 45 11 -9
i = 3
-2 0 11 45 -9
-2 0 11 -9 45


step = 2
i = 0
-2 0 -9 11 45
i = 1
-2 0 -9 11 45
-2 -9 0 11 45

In [7]:

```
li = [1,5,8,3,2,0,-1,55,125,99]

li = [-2, 45, 0, 11, -9]
```

In [11]:

```
def sort(lst):
    for i in range(0, len(lst)):
        for j in range(0, (len(lst) - i -1)):
            if lst[j] > lst[j + 1]:
                temp = lst[j]
                lst[j] = lst[j + 1]
                lst[j + 1] = temp
        print('Step:', i, lst)

```

In [12]:

```
sort(li)
```

```
Step: 0 [-2, 0, -9, 11, 45]
Step: 1 [-2, -9, 0, 11, 45]
Step: 2 [-9, -2, 0, 11, 45]
Step: 3 [-9, -2, 0, 11, 45]
Step: 4 [-9, -2, 0, 11, 45]
```

In [13]:

```
li = [1,5,8,3,2,0,-1,55,125,99]
```

In [14]:

```
sort(li)
```

```
Step: 0 [1, 5, 3, 2, 0, -1, 8, 55, 99, 125]
Step: 1 [1, 3, 2, 0, -1, 5, 8, 55, 99, 125]
Step: 2 [1, 2, 0, -1, 3, 5, 8, 55, 99, 125]
Step: 3 [1, 0, -1, 2, 3, 5, 8, 55, 99, 125]
Step: 4 [0, -1, 1, 2, 3, 5, 8, 55, 99, 125]
Step: 5 [-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
Step: 6 [-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
Step: 7 [-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
Step: 8 [-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
Step: 9 [-1, 0, 1, 2, 3, 5, 8, 55, 99, 125]
```

In [15]:

```python
def sort(lst):
    for i in range(0, len(lst)):
        for j in range(0, (len(lst) - i -1)):
            if lst[j] < lst[j + 1]:
                temp = lst[j]
                lst[j] = lst[j + 1]
                lst[j + 1] = temp
        print('Step:', i, lst)

```

In [16]:

```python
li = [1,5,8,3,2,0,-1,55,125,99]
```

In [17]:

```python
sort(li)
```

```
Step: 0 [5, 8, 3, 2, 1, 0, 55, 125, 99, -1]
Step: 1 [8, 5, 3, 2, 1, 55, 125, 99, 0, -1]
Step: 2 [8, 5, 3, 2, 55, 125, 99, 1, 0, -1]
Step: 3 [8, 5, 3, 55, 125, 99, 2, 1, 0, -1]
Step: 4 [8, 5, 55, 125, 99, 3, 2, 1, 0, -1]
Step: 5 [8, 55, 125, 99, 5, 3, 2, 1, 0, -1]
Step: 6 [55, 125, 99, 8, 5, 3, 2, 1, 0, -1]
Step: 7 [125, 99, 55, 8, 5, 3, 2, 1, 0, -1]
Step: 8 [125, 99, 55, 8, 5, 3, 2, 1, 0, -1]
Step: 9 [125, 99, 55, 8, 5, 3, 2, 1, 0, -1]
```

In [18]:

```python

li = [1,5,8,3,2,0,-1,55,125,99]
```

In [19]:

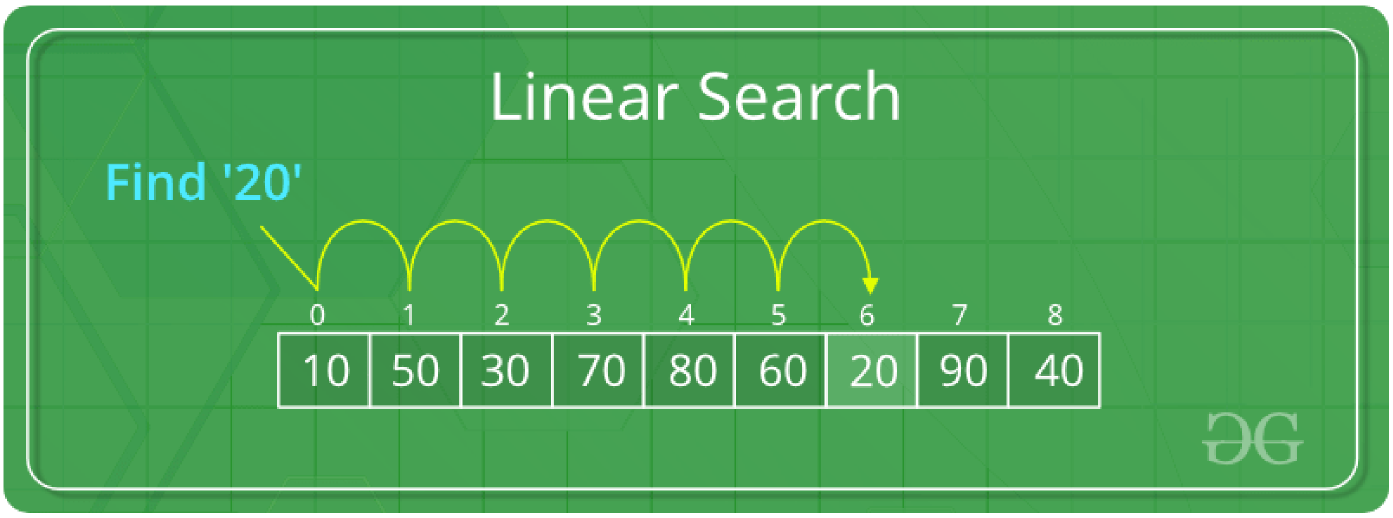
```python
li.index(55)
```

Out[19]:

```
7
```

Linear Search

Linear Search is defined as a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found, otherwise the search continues till the end of the data set. It is the easiest searching algorithm



In [20]:

```python
search = 55
for ele in range(len(li)):
    if li[ele] == search:
        index = ele
        print('index of 55 is', index)
```

```
index of 55 is 7
```

In [21]:

```python
li = [1,5,8,3,2,0,-1,55,125,99, 55, 452, 55, 56, 55, 666, 55]
```

In [22]:

```python
li.index(55)
```

Out[22]:

```
7
```

In [23]:

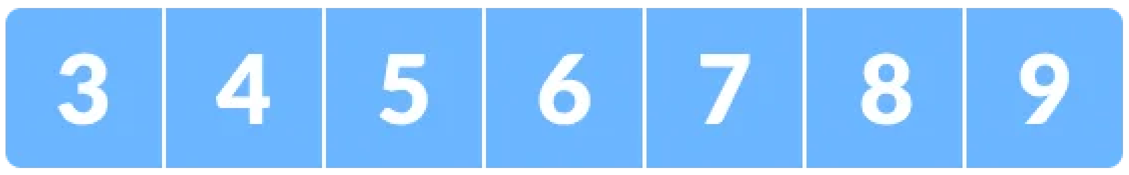
```python
index = []
search = 55
for ele in range(len(li)):
    if li[ele] == search:
        index.append(ele)
print('index of 55 is', index)
```

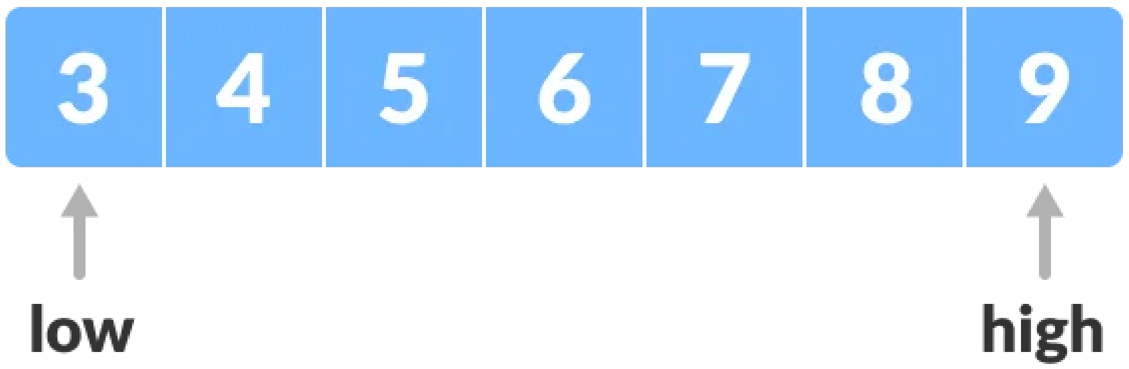
```
index of 55 is [7, 10, 12, 14, 16]
```

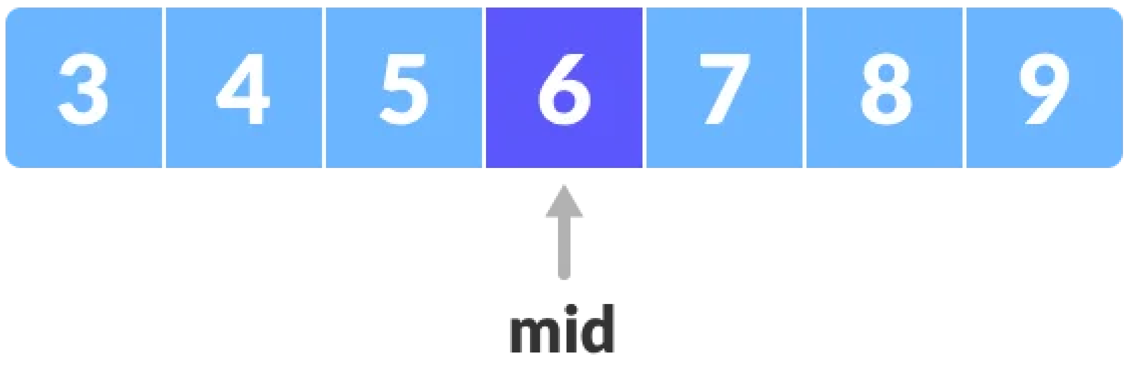
Binary Search

Binary Search is a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

- Condition to use Binary Search -> Data in sorted


3
4
5
6
7
8
9


3
4
5
6
7
8
9
low
high


3
4
5
6
7
8
9
mid

In [24]:

```python
def index(lst, low, high, search):
    while low <= high:
        # 5
        mid = low + (high - low) // 2
        # 0 3 6
        # 0 1 2
        # 2 2 2
        print(high, mid, low)
        if lst[mid] == search:
            return mid
            # 6 < 5
        elif lst[mid] < search:
            low = mid + 1
            # 6 > 5
        elif lst[mid] > search:
            high = mid - 1
            # 2

    return -1
```

In [25]:

```python
li = [3, 4, 5, 6, 7, 8, 9]
search = 5
index(li, 0, len(li) - 1, search)
```

```
6 3 0
2 1 0
2 2 2
```

Out[25]:

```
2
```

In [26]:

```python
li = [3, 4, 5, 6, 7, 8, 9]
search = 9
index(li, 0, len(li) - 1, search)
```

```
6 3 0
6 5 4
6 6 6
```

Out[26]:

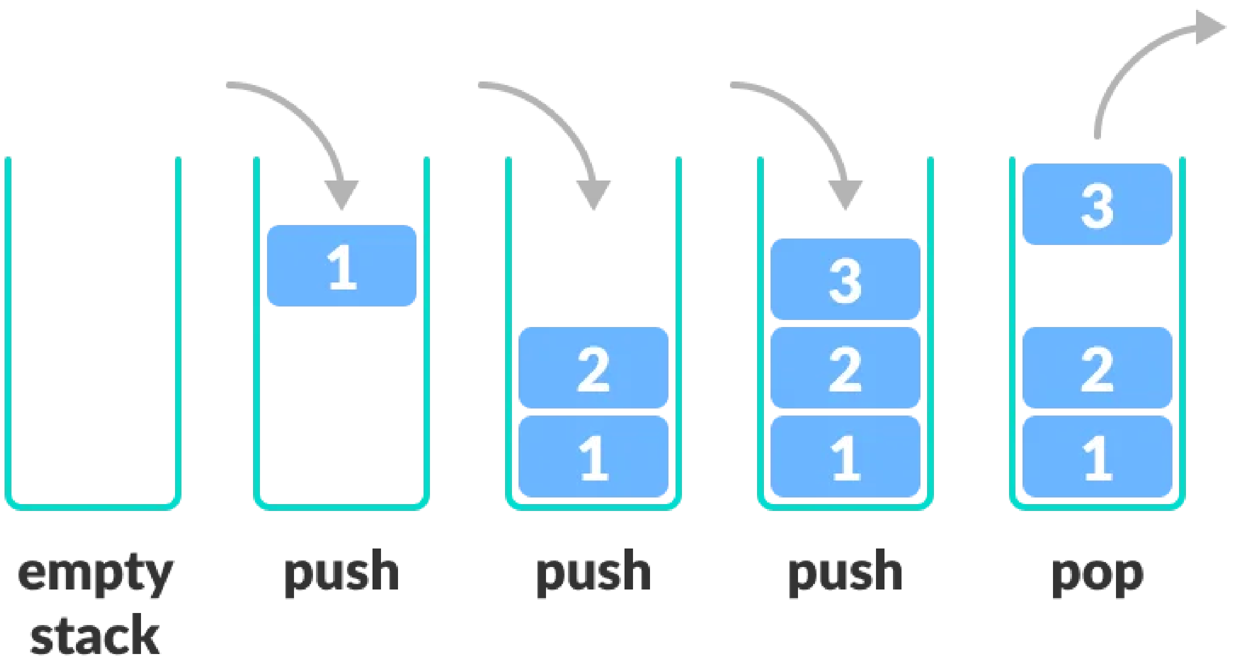
```
6
```

Stack

Stack - ADT: Abstract Data Type

Stack is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out).

1. What is a Stack
2. Properties
3. Implementation
4. Application of stack



- LIFO
- FIFO

Application of Stack in real life:

- CD/DVD stand.
- Stack of books in a book shop.
- Undo and Redo mechanism in text editors.

Basic Operations of Stack

There are some basic operations that allow us to perform different actions on a stack.

- **Push**: Add an element to the top of a stack
- **Pop**: Remove an element from the top of a stack
- **IsEmpty**: Check if the stack is empty
- **IsFull**: Check if the stack is full (arrays)
- **Peek**: Get the value of the top element without removing it

chrome > gmail > fb > instagram > zoom

gmail > teams

In [ ]:

```
li = [1, 2, 3]


print(li[-1])
```

```
3
```

In [ ]:

```
 list = []

 list -> 5
```