4 3 2 1 5 = len=5

1 3 2 4 5

len = 3-1

1 3 2

selection Sort:

```
for i in range(len(arr) -1)   1
    index=i   # min value
    for j in range ( i+1, len(arr) )
        if arr[j] < arr[index]
        index = j   2

    if index != i:
        arr[index],arr[i]=arr[i],arr[index]
```

Insertion Sort : ->

1 5 3 6 4

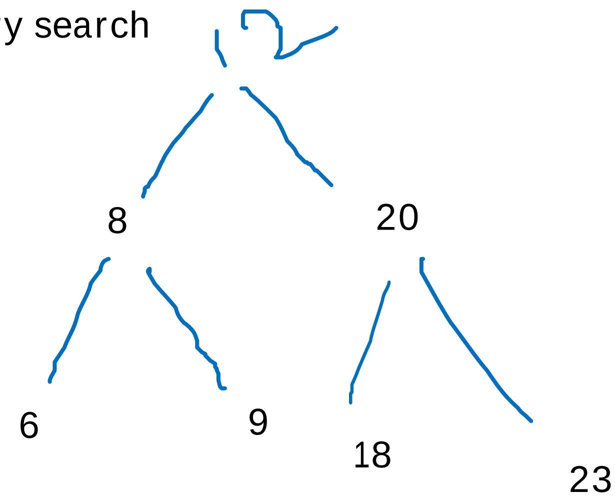
```
for i in range(len(arr)):
    j=i
    while j>=0 and arr[j-1]>arr[j]:
        arr[j-1],arr[j]=arr[j],arr[j-1]
        j=j-1
```

quick sort :

```
class QuickSort:
    def __init__(self,data):
        self.data=data

    def sort(self):
        self.quick_sort(0,len(self.data)-1)


    def quick_sort(self,low,high):

        if low>=high:
            return

        pivot_index=self.partition(low,high)
        self.quick_sort(low,pivot_index-1)
        self.quick_sort(pivot_index+1,high)

    def partition(self,low,high):
        pivot_index=(low+high)//2
        self.data[pivot_index],self.data[high]=self.data[high],self.data[pivot_index]

        for i in range(low,high):
            if self.data[i]<=self.data[high]
                self.data[i],self.data[low]=self.data[low],self.data[i]
                low+=1
        self.data[high],self.data[low]=self.data[low],self.data[high]
        return low
```

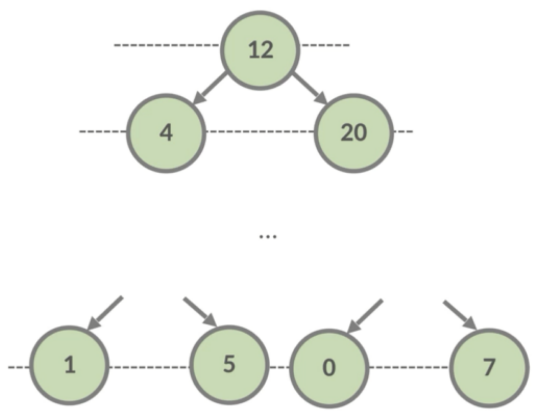
```
if __name__=="__main__":
    x=[2,13,5,1]
    quickSort=QuickSort(x)
    quickSort=sort()
    print(x)
```

binary search




```
def linear search(arr,x):
    for i in range(len(arr)):
        if arr[i]==x:
            print(f"index {i} : {x}")
```

Binary Search Trees



layer 1 with 2^0 nodes

layer 2 with 2^1 nodes

layer h with 2^{h-1} nodes

*how many **N** nodes are there in a complete binary search tree with **h** height?*

$$2^{h-1} = N$$
$$\log_2 2^{h-1} = \log_2 N$$
$$h = \log_2 N + 1$$
$$h = O(\log N)$$

def common(a,b,c):

n1,n2,n3=len(a),len(b),len(c)

i

w

1, 2, 3
3, 4, 5
3, 6, 9

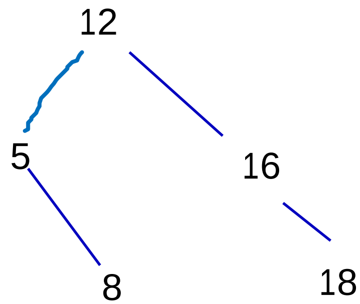
i : 0 1
j : 0
k: 0

```
while i<n1 and j<n2 and k < n3:
    if a[i] == b[j]  and b[j] == c[k]:
        arr.append(a[i])
        i+=1
        j+=1
        k+=1

    elif a[i] < b[j]:
        i+=1

    elif b[j] < c[k]
        j+=1

    else:
        k+=1
```

it: 1

p

12 5 8 16 18



Bubble sort :

time complexity : O(n^2) # two for loop is needed

space complexity : O(1)

swap no additional space is required that's why it is O(1)

```
for i in range(len(arr)):
    for j in range(len(arr)-1-i):
        if arr[i]>arr[j+1] :
            arr[i],arr[j+1]=arr[j+1],arr[i]
```

```
class QuickSort:
    def __init__(self,data):
        self.data=data

    def quick_sort():
        length=len(self.data)
        for i in range (length-1)
            for j in range (length-1-i):
                if self.data[j]>self.data[j+1]
                    swap(j,j+1)
```

8, 22, 7, 9, 31, 5, 13

```
8 7 22 9 31 5 13
8 7 9 22 31 5 13
8  7  9  22  5  31  13
8  7  9  22  5  13  31
```

4

Selection Sort:

Time Complexity: O(n^2)

Space Complexity : O(1)

```
class SelectionSort:
    def __init__(self,data):
        self.data=data

    def selection_sort():
        length =len(self.data)
        for i in range(length-1):
            min_index=itw
            for j in range(min_index+1,length):
                if self.data[j]<self.data[min_index]:
                    min_index=j
        if i != min_index:
            swap(self.data[i],self.data[min_index])
```

Insertion Sort :

Time Complexity : $O(N^2)$

Space Complexity : $O(1)$

```
def insertion_sort(data):
    for i in range(len(data)):
        j=i
        while j>0 and data[j-1] >data[j]:
            swap(data[j],data[j-1])
            j -=1
```