**SUBODH SHRESTHA**

1.

line = "i love to code in python"

y=[]

i = 0

p=-1

final=[]

while i < len(line):

if line[i]==' ':

y.append(line[p+1:i])

p=i

i += 1

y.append(line[p+1:len(line)])

print(y)

#takes a list and returns in ascending order

def ascending(y):

for i in range(len(y)-1,0,-1):

for j in range(i):

if y[j]>y[j+1]:

temp=y[j]

y[j]=y[j+1]

y[j+1]=temp

return y

#takes string to make list

def makelist(a):

g=[]

for x in a:

g.append(x)

return g

ascending(y)

print(y)

start=0

while start < len(y):

c=makelist(y[start])

v=ascending(c)

b=''.join(v)

final.append(b)

start+=1

print(final)

2.

**Bubble sort:**

Bubble sort is a simple sorting algorithm which sorts the elements by comparing the next element.

Steps:

1. Start with the first element and compare with the second element
2. If the current element is greater than the next element, we perform swap operation
3. If current element is less than the second element than we move to the next element and take it as a current element.
4. Repeat step 1 making iteration till N-(no of iteration). i.e in first repetition it will be N-1

Let us consider this example of a list A= [3,8,5,6,2,4]

**Iteration 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

The array has all elements in it

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

3 < 8 so no change

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

8> 5 so swap 8 and 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 8 | 6 | 2 | 4 |

8>6 so swap 8 and 6

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 6 | 8 | 2 | 4 |

8 > 2 so swap 8 and 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 6 | 2 | 8 | 4 |

8 > 4 so swap 8 and 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 6 | 2 | 4 | 8 |

**Iteration 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 6 | 2 | 4 | 8 |

3< 5 and 5 <6 so no swap

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 6 | 2 | 4 | 8 |

6>2 so swap 6 and 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 2 | 6 | 4 | 8 |

6>4 so swap 6 and 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 2 | 4 | 6 | 8 |

6<8 no swap

**Iteration 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 2 | 4 | 6 | 8 |

3<5 no swap

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 5 | 2 | 4 | 6 | 8 |

5> 2 so swap

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 5 | 4 | 6 | 8 |

5>4 so swap

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 4 | 5 | 6 | 8 |

Rest is all less than the next element

**Iteration 4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 2 | 4 | 5 | 6 | 8 |

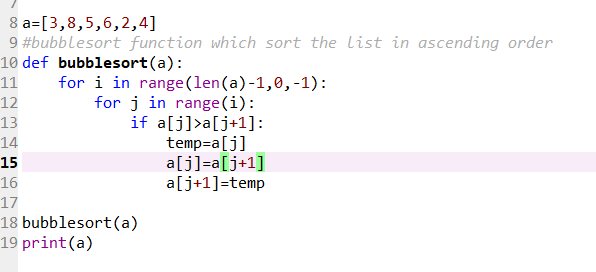
2<3 so swap

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6 | 8 |

Rest element is all less than next element

**Hence the sorted list for A=[3,8,5,6,2,4] is A[2,3,4,5,6,8]**

**Code for the algorithm:**

** 3**

**Time complexity:**

Here the worst time complexity of bubble sort is N^2. Best time complexity is N which occurs when we pass a sorted list. **A=[3,8,5,6,2,4]**

**Selection sort:**

Like bubble sort this is also comparison-based sorting algorithm.

Steps:

1. Set the first element as a min value.
2. Compare the min value with the next element. If min value> second element than update min value = second value.
3. Iterate through the list getting one min value, swap min value to the first element of the list.
4. Set second element as a min value.
5. Repeat 2 to 3 but swap the min value to the second element of the list.
6. This keeps up adding min value to the list staring from the first element and terminates with the sorted list at updating last element of the list.

Let us take a list A=[3,8,5,6,2,4]

**Iteration 1:**

Take first element as a min index. Min= 0 index.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min index = 0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min <8 no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min <5 no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min <6 no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min >2 so update Min index=4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | 8 | 5 | 6 | 2 | 4 |

Min< 4 so no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 8 | 5 | 6 | 3 | 4 |

Swap min index 4 with initial min index 0 with first location value

**Iteration2:**

Take second element as min index: Min index= 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 8 | 5 | 6 | 3 | 4 |

5<8 so update Min Min =5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 8 | 5 | 6 | 3 | 4 |

Min <6 so no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 8 | 5 | 6 | 3 | 4 |

Min >3 so update Min=4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 8 | 5 | 6 | 3 | 4 |

Min <4 so no update

Swap min index value to second index value

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 | 4 |

**Iteration 3:**

Take third element as Min. Min index =2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 | 4 |

Min <6 so no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 | 4 |

Min <8 so no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 | 4 |

Min >4 so update min =5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 8 | 5 |

Swap min index 5 to third index value 2.

Iteration 4:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 8 | 5 |

Take fourth element as min: Min index=3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 8 | 5 |

Min <8 so no update

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 6 | 8 | 5 |

Min >5 so update min index to 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 8 | 6 |

Swap min index 5 value with fourth element index 3

**Iteration 5:**

Take fifth element as min value: Min =4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 8 | 6 |

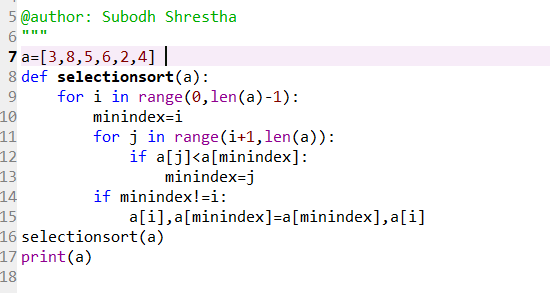
Min >6 so update min value=5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | 3 | 4 | 5 | 6 | 8 |

Swap min value 5 with fifth index value 4

**Hence the sorted list for A=[3,8,5,6,2,4] is A[2,3,4,5,6,8]**

Code for selection sort:



**Time complexity**:

Worst case time complexity: n^2(nested for loops)

**Quicksort:** Quick sort is recursive divide and conquer algorithm.

Steps for quick sort which is solved recursively:

1. Bring the pivot to the position such that the left of the pivot is smaller, and right is greater.
2. Quick sort left part (subpart I)
3. Quick sort right part (subpart II)

Algorithm:

1. Take a pivot at the last element of array
2. Take i as the index of smaller element and j as loop variable
3. Initialize i=-1 and j=0
4. Check if element is j is less than the pivot element. If yes increment i and swap i and j.
5. Else ignore current element and no swapping
6. Repeat 4-5 until j is equal to (size of array -1)
7. Swap array[i+1] with pivot value.

Let us consider the array with 7 elements in it. The pivot we have is in the last element which is 70

Let, i: index of smaller element and j: loop variable.

We change the numbers if number in the index j is less than the pivot value.

Else no swap and increment the current index C. initial value of i= - 1 and j =0

pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 80 | 30 | 90 | 40 | 50 | 70 |

i=10 j =10 . 10 <70(pivot) . now increment i=0,j=1 swap. i and j.

j

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 80 | 30 | 90 | 40 | 50 | 70 |

i

j>70 no swap j=2 ,i=0

j

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 80 | 30 | 90 | 40 | 50 | 70 |

i

j <70, increment i=1,j=3 swap 30 and 80

j

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 80 | 90 | 40 | 50 | 70 |

i

90> 70 no change increase j=4

j

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 80 | 90 | 40 | 50 | 70 |

i

40 < 70 increment i=2 swap 80 and 40

j

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 40 | 90 | 80 | 50 | 70 |

i

50 < 70 , increment i=3 and swap 50 and 90

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 40 | 50 | 80 | 90 | 70 |

j=6 we come out of the loop. We swap i+1 and pivot

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 40 | 50 | 70 | 80 | 90 |

The array is sub divided into two where the left sub array has the element number less than the pivot number and the right subarray has the element greater than pivot number 70.

Pivot Pivot

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 40 | 50 |  | 70 |  | 80 | 90 |

Again, we apply same algorithm to each of the sub array. The pivot will be the last element in the sub array. We get,

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 | 40 |  | 50 |  | {} |  | 70 |  | 80 |  | 90 |

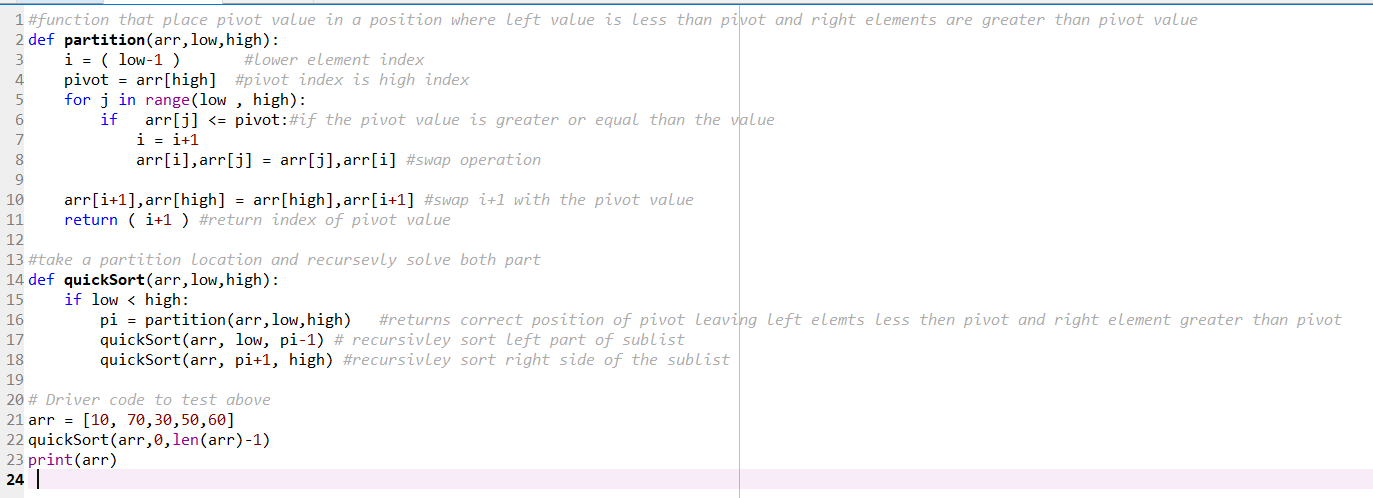
Again,

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 30 |  | 40 |  | {} |  | 50 |  | {} |  | 70 |  | 80 |  | 90 |

Finally

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 |  | 30 |  | 40 |  |  |  | 50 |  |  |  | 70 |  | 80 |  | 90 |

Code for Quick sort:



Time Complexity:

Best Case: when the middle element as taken as pivot. Hence, O (n log n).

Worst case time complexity: O(N^2)

Merge sort:

Merge sort is a divide and conquer algorithm. It keeps on dividing the list into halves recursively and merge back together to one piece comparing and sorting the individual elements. Merge sort has merge function which merges the halves into one piece.

Algorithm steps:

1. Cutting the list into two halves after getting the mid:
2. Call merge sort on first half
3. Call merge sort on second half
4. Merge two halves

Let us consider array A=[20,39,40,59,80,70,60,10]

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 39 | 40 | 59 | 80 | 70 | 60 | 10 |

The list is cut in halves

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 39 | 40 | 59 |  | 80 | 70 | 60 | 10 |

Recursively the sub list is cut into halves

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 39 |  | 40 | 59 |  | 80 | 70 |  | 60 | 10 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 |  | 39 |  | 40 |  | 59 |  | 80 |  | 70 |  | 60 |  | 10 |

The individual elements are compared and merged.

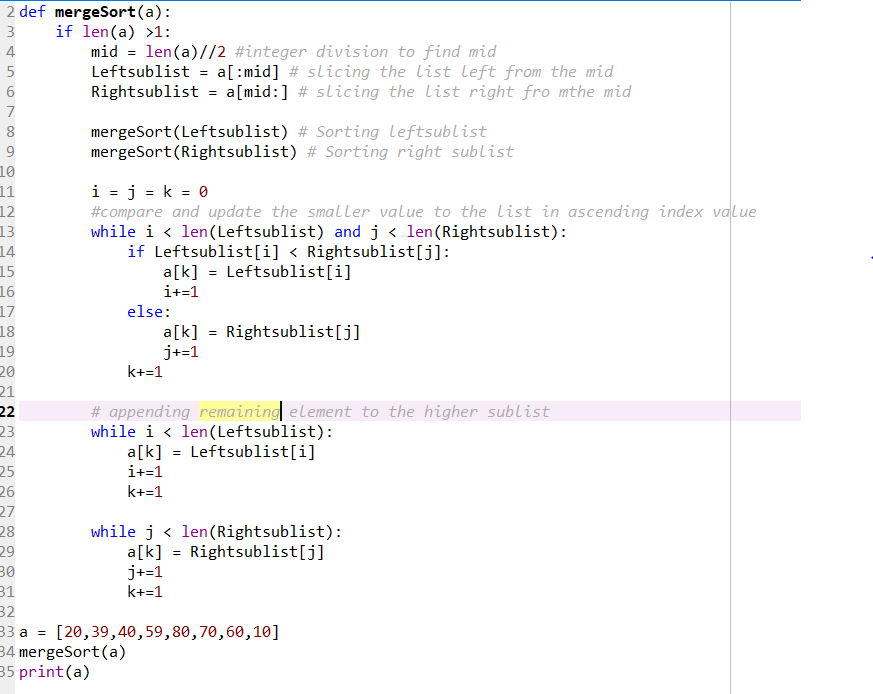
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 39 |  | 40 | 59 |  | 70 | 80 |  | 10 | 60 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 39 | 40 | 59 |  | 70 | 80 | 10 | 60 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 20 | 39 | 40 | 59 | 60 | 70 | 80 |

Finally, the sorted list is being merged

Code for merge sort:



Time complexity:

Average case and worst-case time complexity:(n log (n))