Python - Sorting Algorithms

Sorting refers to arranging data in a particular format. Sorting algorithm specifies the way to arrange data in a particular order. Most common orders are in numerical or lexicographical order.

The importance of sorting lies in the fact that data searching can be optimized to a very high level, if data is stored in a sorted manner. Sorting is also used to represent data in more readable formats. Below we see five such implementations of sorting in python.

- Bubble Sort
- Merge Sort
- Insertion Sort
- Shell Sort
- Selection Sort

Bubble Sort

It is a comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order.

Example

```
def bubblesort(list):

# Swap the elements to arrange in order
   for iter_num in range(len(list)-1,0,-1):
      for idx in range(iter_num):
         if list[idx]>list[idx+1]:
            temp = list[idx]
            list[idx] = list[idx+1]
            list[idx+1] = temp
list = [19,2,31,45,6,11,121,27]
bubblesort(list)
print(list)
```

Output

When the above code is executed, it produces the following result −

```
[2, 6, 11, 19, 27, 31, 45, 121]
```

Merge Sort

Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Example

```
def merge_sort(unsorted_list):
   if len(unsorted_list) <= 1:
      return unsorted_list
# Find the middle point and devide it
   middle = len(unsorted_list) // 2
```

```
      left_list = unsorted_list[:middle]
      right_list = unsorted_list[middle:]

      left_list = merge_sort(left_list)
      right_list = merge_sort(right_list)
      return list(merge(left_list, right_list))

   # Merge the sorted halves
   def merge(left_half,right_half):
      res = []
      while len(left_half) != 0 and len(right_half) != 0:
         if left_half[0] < right_half[0]:
            res.append(left_half[0])
            left_half.remove(left_half[0])
         else:
            res.append(right_half[0])
            right_half.remove(right_half[0])
      if len(left_half) == 0:
         res = res + right_half
      else:
         res = res + left_half
      return res
   unsorted_list = [64, 34, 25, 12, 22, 11, 90]
   print(merge_sort(unsorted_list))
```

Output

When the above code is executed, it produces the following result −

```
[11, 12, 22, 25, 34, 64, 90]
```

Insertion Sort

Insertion sort involves finding the right place for a given element in a sorted list. So in beginning we compare the first two elements and sort them by comparing them. Then we pick the third element and find its proper position among the previous two sorted elements. This way we gradually go on adding more elements to the already sorted list by putting them in their proper position.

Example

```
def insertion_sort(InputList):
   for i in range(1, len(InputList)):
      j = i-1
      nxt_element = InputList[i]
# Compare the current element with next one
   while (InputList[j] > nxt_element) and (j >= 0):
      InputList[j+1] = InputList[j]
      j=j-1
   InputList[j+1] = nxt_element
list = [19,2,31,45,30,11,121,27]
insertion_sort(list)
print(list)
```

Output

When the above code is executed, it produces the following result −

```
[2, 11, 19, 27, 30, 31, 45, 121]
```

Shell Sort

Shell Sort involves sorting elements which are away from each other. We sort a large sublist of a given list and go on reducing the size of the list until all elements are sorted. The below program finds the gap by equating it to half of the length of the list size and then starts sorting all elements in it. Then we keep resetting the gap until the entire list is sorted.

Example

```
def shellSort(input_list):
   gap = len(input_list) // 2
   while gap > 0:
      for i in range(gap, len(input_list)):
         temp = input_list[i]
         j = i
# Sort the sub list for this gap
   while j >= gap and input_list[j - gap] > temp:
      input_list[j] = input_list[j - gap]
      j = j-gap
      input_list[j] = temp
# Reduce the gap for the next element
   gap = gap//2
list = [19,2,31,45,30,11,121,27]
shellSort(list)
print(list)
```

Output

When the above code is executed, it produces the following result −

```
[2, 11, 19, 27, 30, 31, 45, 121]
```

Selection Sort

In selection sort we start by finding the minimum value in a given list and move it to a sorted list. Then we repeat the process for each of the remaining elements in the unsorted list. The next element entering the sorted list is compared with the existing elements and placed at its correct position.So, at the end all the elements from the unsorted list are sorted.

Example

```
def selection_sort(input_list):
   for idx in range(len(input_list)):
      min_idx = idx
      for j in range( idx +1, len(input_list)):
         if input_list[min_idx] > input_list[j]:
         min_idx = j
# Swap the minimum value with the compared value
   input_list[idx], input_list[min_idx] = input_list[min_idx], input_list[idx]
l = [19,2,31,45,30,11,121,27]
selection_sort(l)
print(l)
```

Output

When the above code is executed, it produces the following result −

```
[2, 11, 19, 27, 30, 31, 45, 121]
```

Print Page