Algorithms and Lab

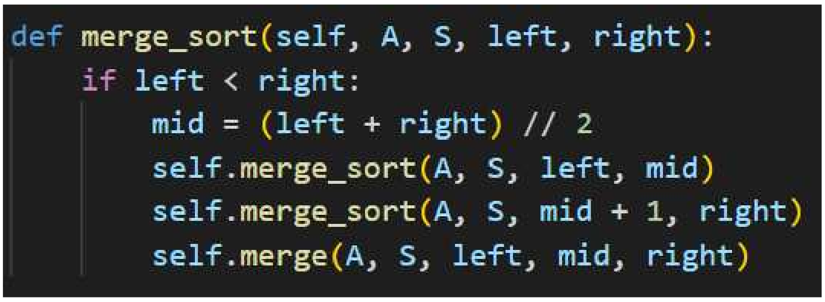
한국기술교육대학교

KOREATECH

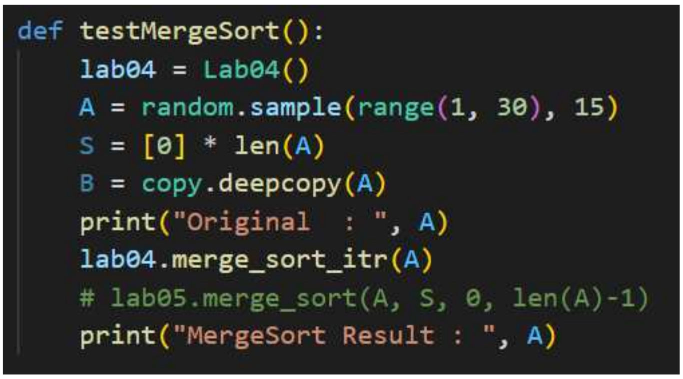
	Lab 04
ID	2020136149
Name	김태섭

2023. 04. 05

1-1. Merge Sort

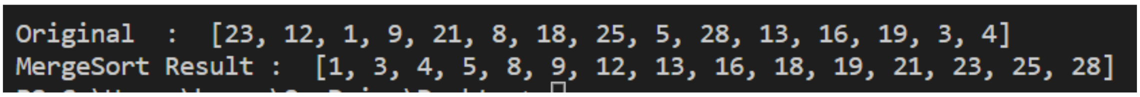


```python
def merge_sort(self, A, S, left, right):
    if left < right:
        mid = (left + right) // 2
        self.merge_sort(A, S, left, mid)
        self.merge_sort(A, S, mid + 1, right)
        self.merge(A, S, left, mid, right)
```



```python
def testMergeSort():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 15)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print("Original  : ", A)
    lab04.merge_sort_itr(A)
    # lab05.merge_sort(A, S, 0, len(A)-1)
    print("MergeSort Result : ", A)
```

Output:

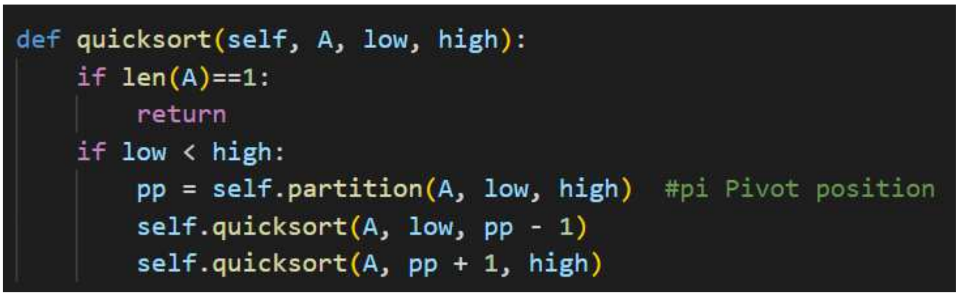


```
Original  :  [23, 12, 1, 9, 21, 8, 18, 25, 5, 28, 13, 16, 19, 3, 4]
MergeSort Result :  [1, 3, 4, 5, 8, 9, 12, 13, 16, 18, 19, 21, 23, 25, 28]
```

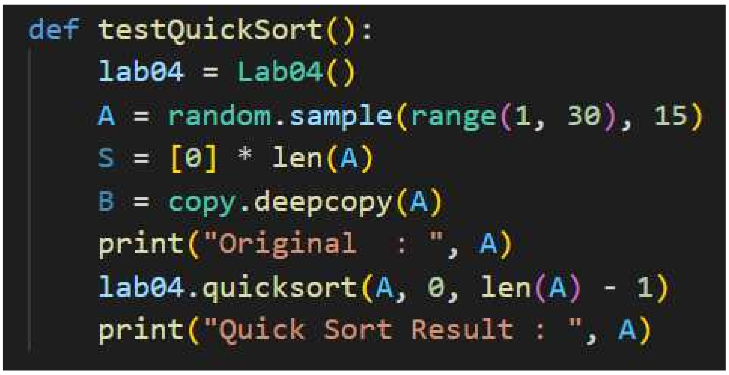
complexity Function:

$$T(h,m) = \begin{cases} 0 & n = 1 \\ T(h) + T(m) + T(h,m) & n > 1 \end{cases}$$

1-2. Quick-Sort

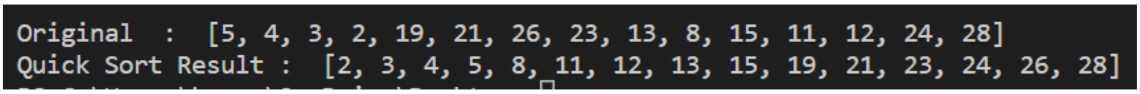


```
def quicksort(self, A, low, high):
    if len(A)==1:
        return
    if low < high:
        pp = self.partition(A, low, high)  #pi Pivot position
        self.quicksort(A, low, pp - 1)
        self.quicksort(A, pp + 1, high)
```



```
def testQuickSort():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 15)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print("Original   : ", A)
    lab04.quicksort(A, 0, len(A) - 1)
    print("Quick Sort Result : ", A)
```

Output:

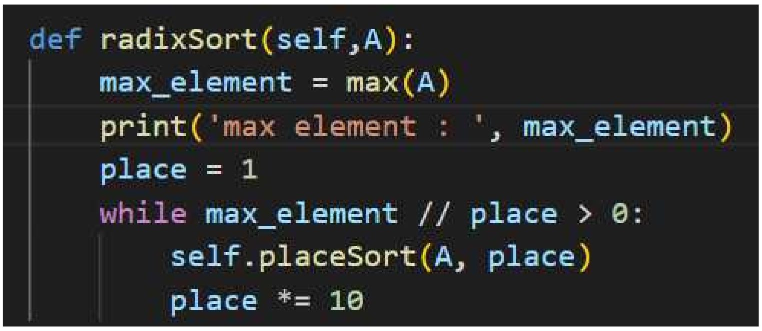


```
Original  :  [5, 4, 3, 2, 19, 21, 26, 23, 13, 8, 15, 11, 12, 24, 28]
Quick Sort Result :  [2, 3, 4, 5, 8, 11, 12, 13, 15, 19, 21, 23, 24, 26, 28]
```

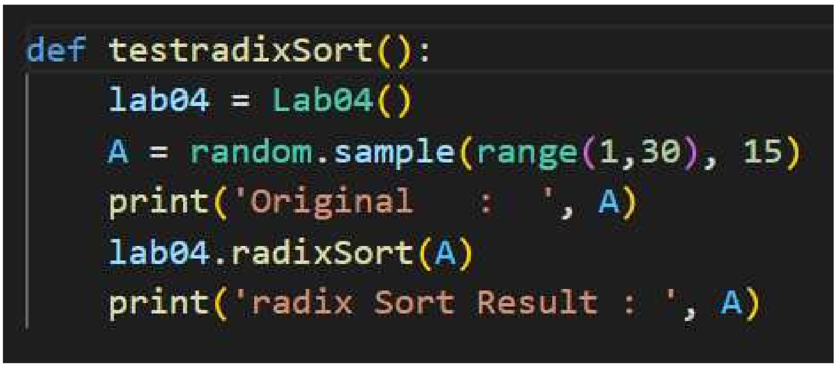
complexity Function:

$$T(n) = \frac{n(n-1)}{2}$$

1-3. Radix Sort

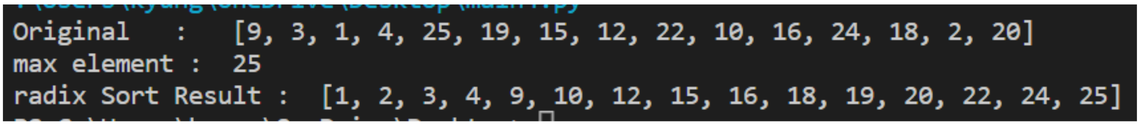


```
def radixSort(self,A):
    max_element = max(A)
    print('max element : ', max_element)
    place = 1
    while max_element // place > 0:
        self.placeSort(A, place)
        place *= 10
```



```
def testradixSort():
    lab04 = Lab04()
    A = random.sample(range(1,30), 15)
    print('Original    :   ', A)
    lab04.radixSort(A)
    print('radix Sort Result : ', A)
```

Output:

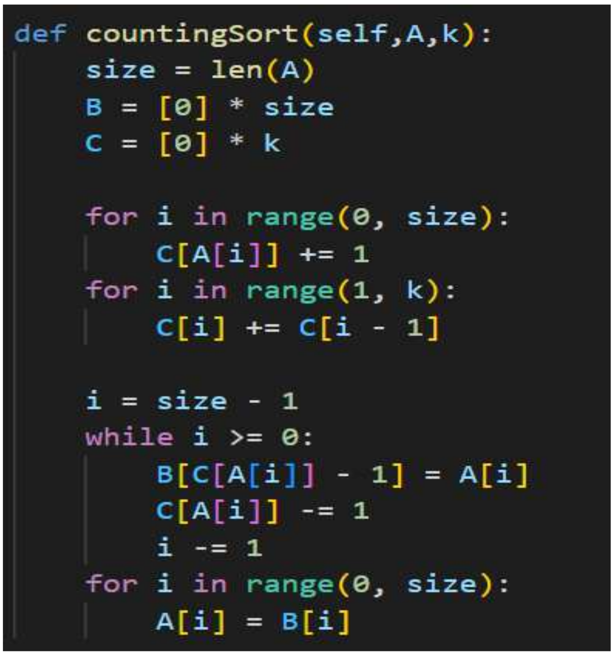


```
Original    :   [9, 3, 1, 4, 25, 19, 15, 12, 22, 10, 16, 24, 18, 2, 20]
max element :   25
radix Sort Result :  [1, 2, 3, 4, 9, 10, 12, 15, 16, 18, 19, 20, 22, 24, 25]
```

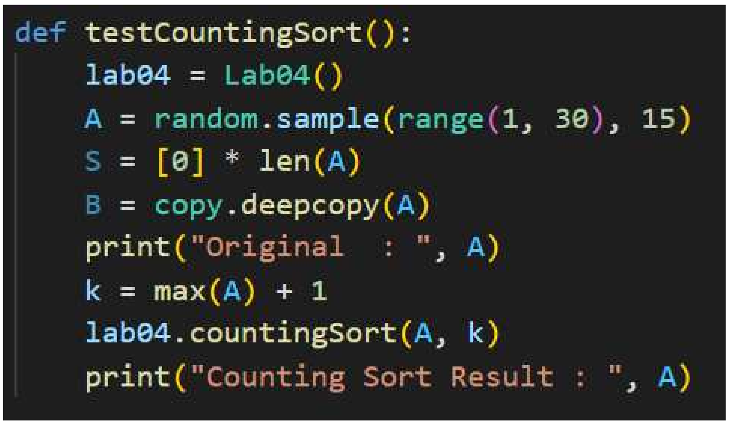
complexity Function:

$$
\begin{aligned}
T(d,n,k) &= \sum_{m=1}^{d}\left[\sum_{i=1}^{k}1+\sum_{j=1}^{n}1+\sum_{i=1}^{k-1}1+\sum_{j=1}^{n}1\right]\\
&= \sum_{m=1}^{d}\left[k+n+k-1+n\right]\\
&= d\left[2k+2n-1\right]\\
&\in \Theta(d(n+k))
\end{aligned}
$$

1-4. Counting Sort

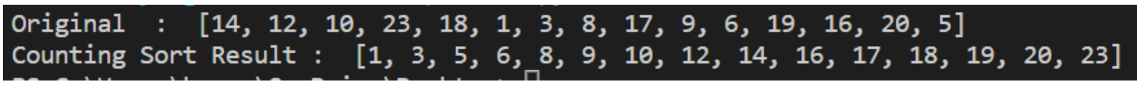


```
def countingSort(self,A,k):
    size = len(A)
    B = [0] * size
    C = [0] * k

    for i in range(0, size):
        C[A[i]] += 1
    for i in range(1, k):
        C[i] += C[i - 1]

    i = size - 1
    while i >= 0:
        B[C[A[i]] - 1] = A[i]
        C[A[i]] -= 1
        i -= 1
    for i in range(0, size):
        A[i] = B[i]
```



```
def testCountingSort():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 15)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print("Original  : ", A)
    k = max(A) + 1
    lab04.countingSort(A, k)
    print("Counting Sort Result : ", A)
```

Output:

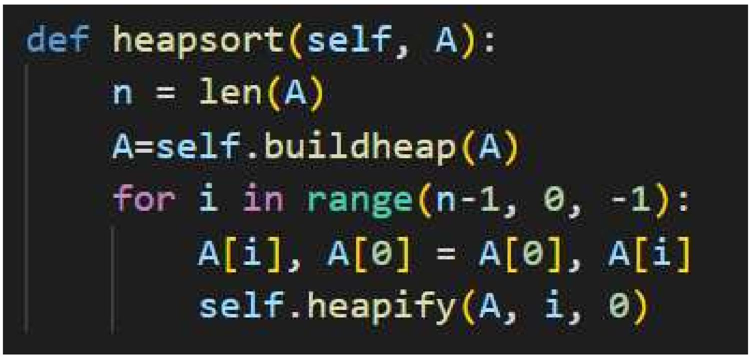


```
Original  :  [14, 12, 10, 23, 18, 1, 3, 8, 17, 9, 6, 19, 16, 20, 5]
Counting Sort Result :  [1, 3, 5, 6, 8, 9, 10, 12, 14, 16, 17, 18, 19, 20, 23]
```

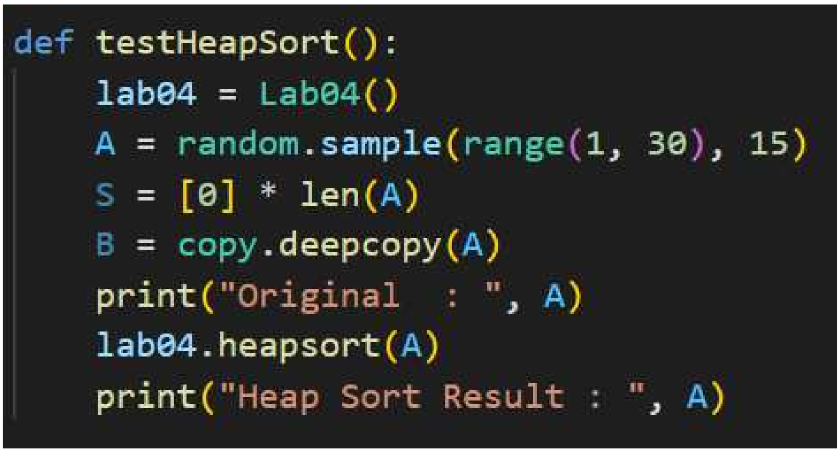
complexity Function:

$$
\begin{aligned}
T(n,k) &= \sum_{i=1}^{k} 1 + \sum_{j=1}^{n} 1 + \sum_{i=1}^{k-1} 1 + \sum_{j=1}^{n} 1 \\
&= k + n + k - 1 + n \\
&= 2k + 2n - 1 \\
&\in \ominus(n+k)
\end{aligned}
$$

1-5. Heap Sort

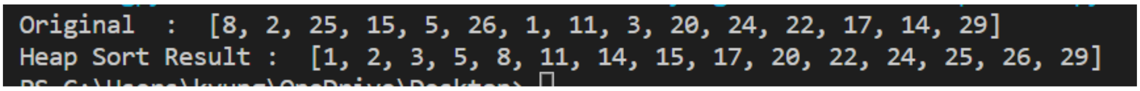


```python
def heapsort(self, A):
    n = len(A)
    A=self.buildheap(A)
    for i in range(n-1, 0, -1):
        A[i], A[0] = A[0], A[i]
        self.heapify(A, i, 0)
```



```python
def testHeapSort():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 15)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print("Original   : ", A)
    lab04.heapsort(A)
    print("Heap Sort Result : ", A)
```

Output:

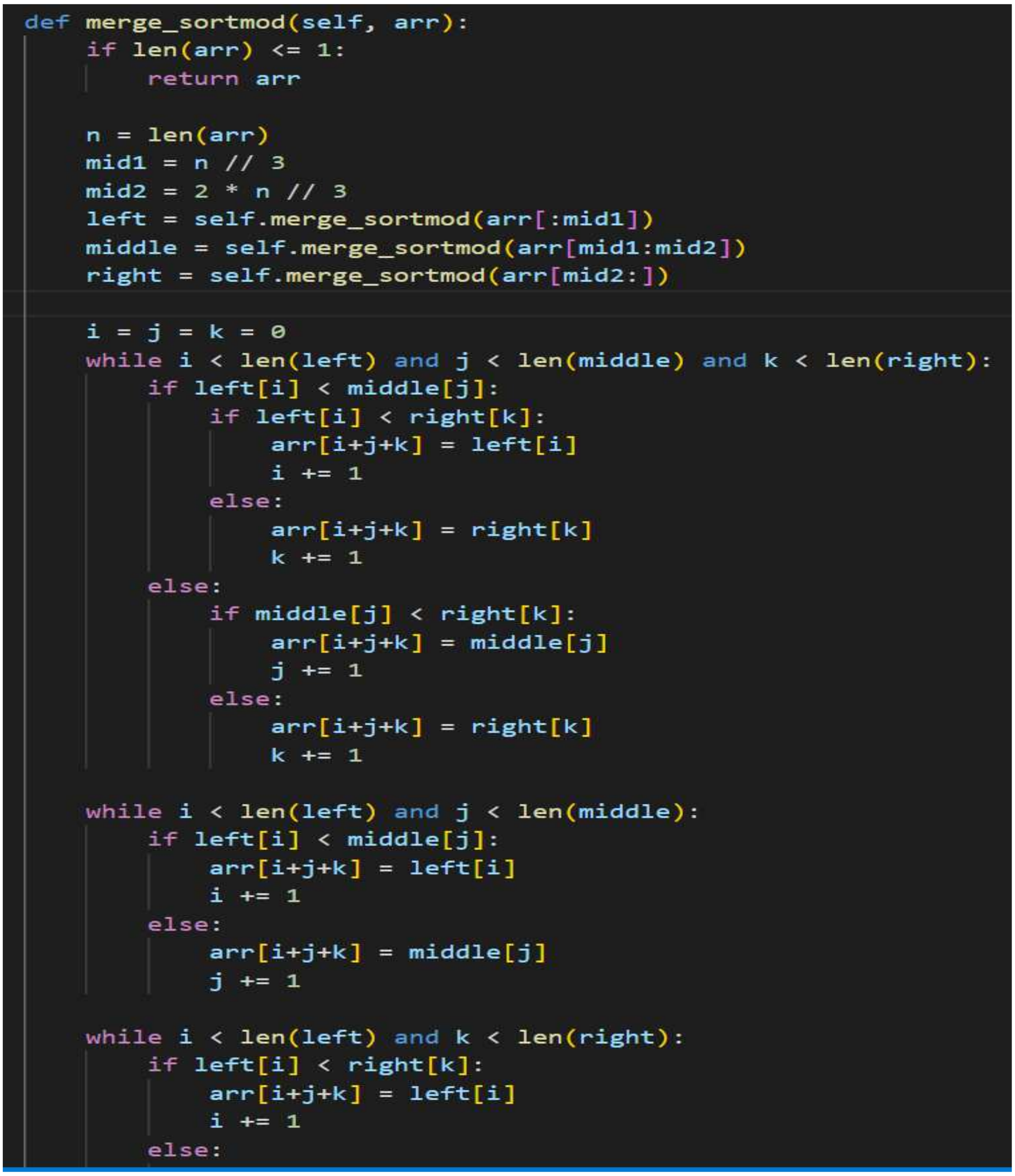


```
Original   :  [8, 2, 25, 15, 5, 26, 1, 11, 3, 20, 24, 22, 17, 14, 29]
Heap Sort Result :  [1, 2, 3, 5, 8, 11, 14, 15, 17, 20, 22, 24, 25, 26, 29]
PS C:\Users\kyung\OneDrive\Desktop> 
```

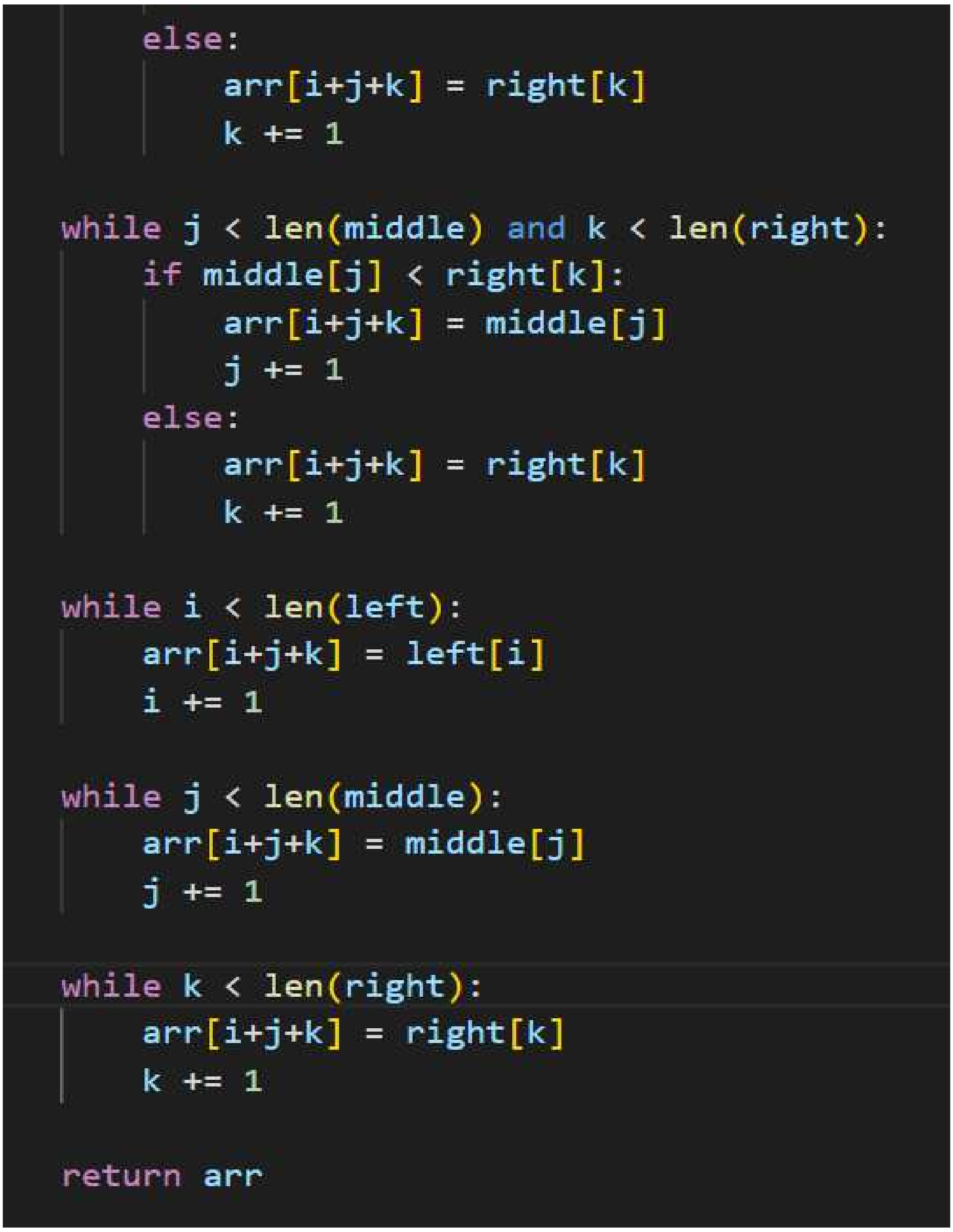
complexity Function:

O(n log n)

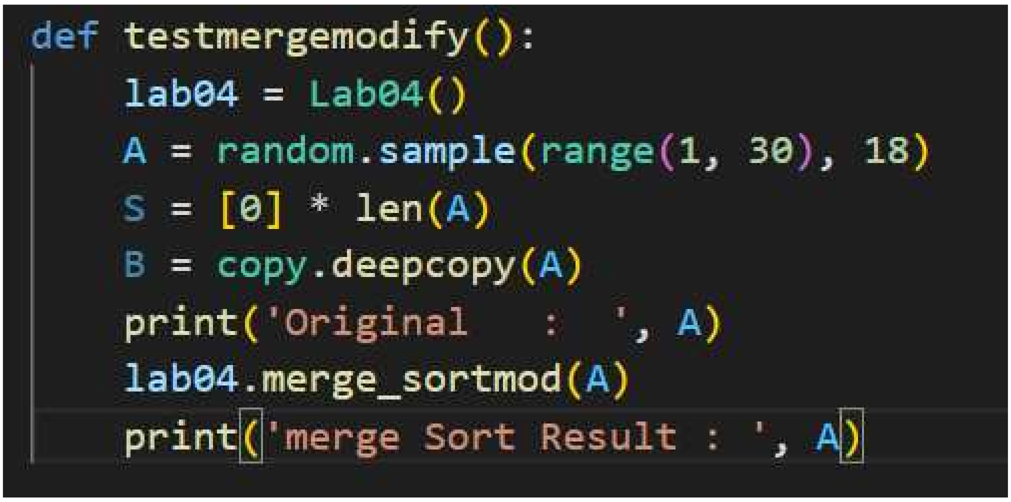
2.



```python
def merge_sortmod(self, arr):
    if len(arr) <= 1:
        return arr

    n = len(arr)
    mid1 = n // 3
    mid2 = 2 * n // 3
    left = self.merge_sortmod(arr[:mid1])
    middle = self.merge_sortmod(arr[mid1:mid2])
    right = self.merge_sortmod(arr[mid2:])

    i = j = k = 0
    while i < len(left) and j < len(middle) and k < len(right):
        if left[i] < middle[j]:
            if left[i] < right[k]:
                arr[i+j+k] = left[i]
                i += 1
            else:
                arr[i+j+k] = right[k]
                k += 1
        else:
            if middle[j] < right[k]:
                arr[i+j+k] = middle[j]
                j += 1
            else:
                arr[i+j+k] = right[k]
                k += 1

    while i < len(left) and j < len(middle):
        if left[i] < middle[j]:
            arr[i+j+k] = left[i]
            i += 1
        else:
            arr[i+j+k] = middle[j]
            j += 1

    while i < len(left) and k < len(right):
        if left[i] < right[k]:
            arr[i+j+k] = left[i]
            i += 1
        else:
```

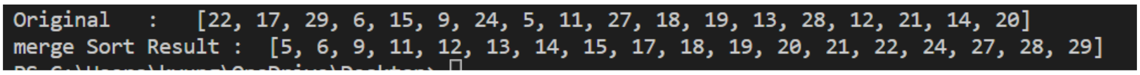


```python
    else:
        arr[i+j+k] = right[k]
        k += 1

while j < len(middle) and k < len(right):
    if middle[j] < right[k]:
        arr[i+j+k] = middle[j]
        j += 1
    else:
        arr[i+j+k] = right[k]
        k += 1

while i < len(left):
    arr[i+j+k] = left[i]
    i += 1

while j < len(middle):
    arr[i+j+k] = middle[j]
    j += 1

while k < len(right):
    arr[i+j+k] = right[k]
    k += 1

return arr
```



```python
def testmergemodify():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 18)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print('Original    :  ', A)
    lab04.merge_sortmod(A)
    print('merge Sort Result : ', A)
```

Output:



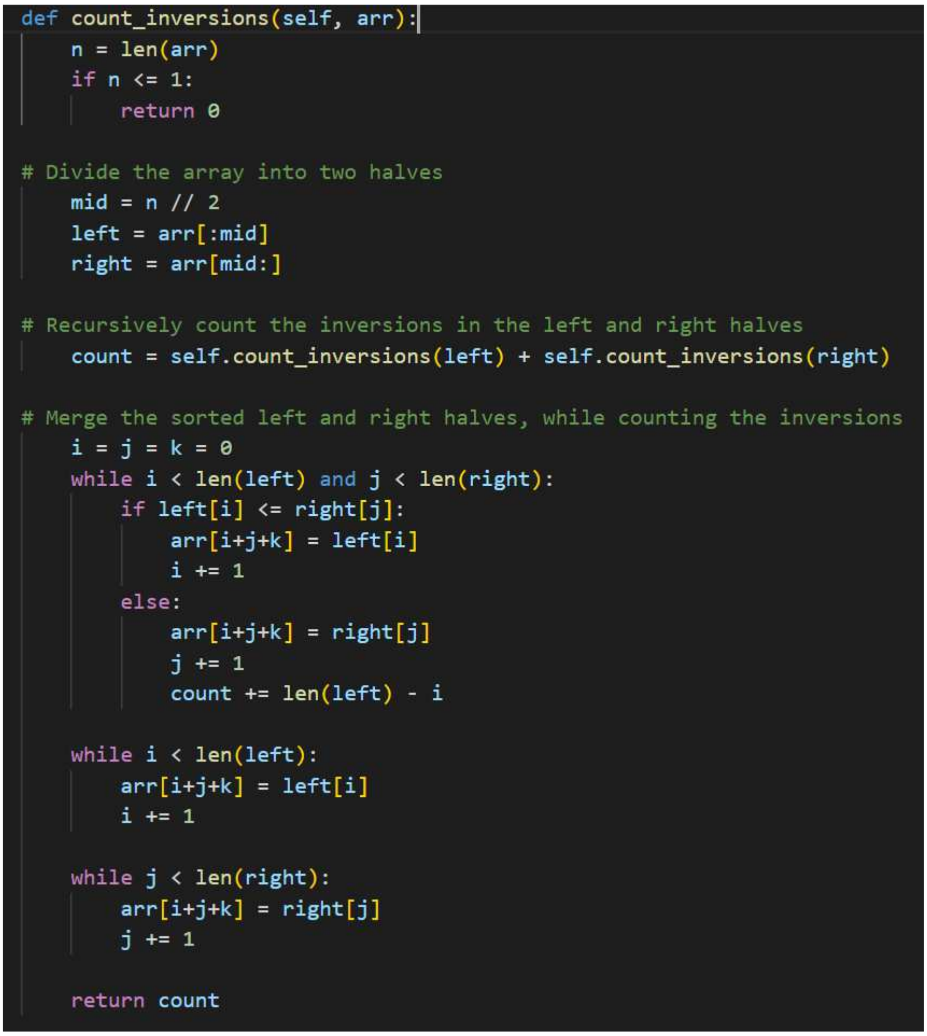
```
Original    :   [22, 17, 29, 6, 15, 9, 24, 5, 11, 27, 18, 19, 13, 28, 12, 21, 14, 20]
merge Sort Result :  [5, 6, 9, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 24, 27, 28, 29]
```

complexity Function:

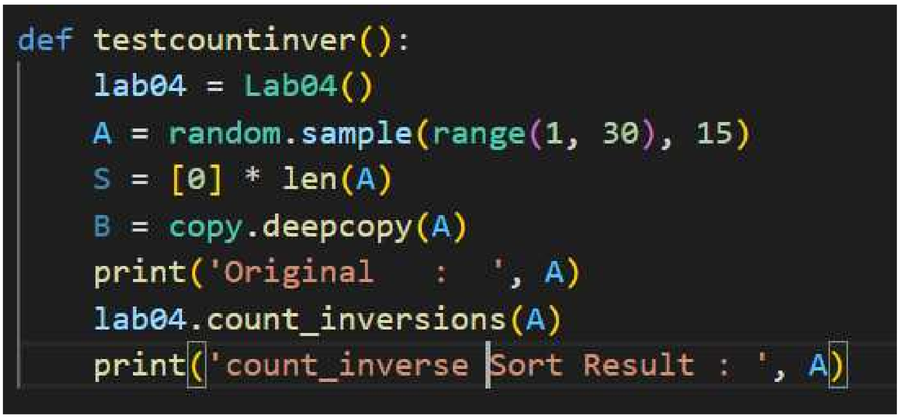
$$T(n) = 3T(n/3) + O(n)$$

$$O(n\log n)$$

3.

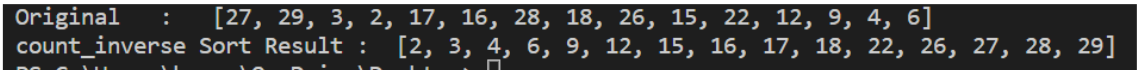


```
def count_inversions(self, arr):
    n = len(arr)
    if n <= 1:
        return 0

# Divide the array into two halves
    mid = n // 2
    left = arr[:mid]
    right = arr[mid:]

# Recursively count the inversions in the left and right halves
    count = self.count_inversions(left) + self.count_inversions(right)

# Merge the sorted left and right halves, while counting the inversions
    i = j = k = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            arr[i+j+k] = left[i]
            i += 1
        else:
            arr[i+j+k] = right[j]
            j += 1
            count += len(left) - i

    while i < len(left):
        arr[i+j+k] = left[i]
        i += 1

    while j < len(right):
        arr[i+j+k] = right[j]
        j += 1

    return count
```



```
def testcountinver():
    lab04 = Lab04()
    A = random.sample(range(1, 30), 15)
    S = [0] * len(A)
    B = copy.deepcopy(A)
    print('Original    :  ', A)
    lab04.count_inversions(A)
    print('count_inverse Sort Result : ', A)
```

Output:



```
Original    :   [27, 29, 3, 2, 17, 16, 28, 18, 26, 15, 22, 12, 9, 4, 6]
count_inverse Sort Result :  [2, 3, 4, 6, 9, 12, 15, 16, 17, 18, 22, 26, 27, 28, 29]
```

complexity Function:

O(n log n)

$$T(n) = 2T(n/2) + O(n)$$

4.

Table 1: Any 10 comparisons based sorting algorithms

Sno	Name	Best	Average	Worst	Stable(Y/N)	In-Place(Y/N)
1	MergeSort	nlogn	nlogn	nlogn	Y	Y
2	QuickSort	nlogn	nlogn	n^2	Y	Y
3	HeapSort	nlogn	nlogn	nlogn	N	Y
4	IntroSort	nlogn	nlogn	nlogn	N	N
5	InsertionSort	n	n^2	n^2	Y	Y
6	BlockSort	n	nlogn	nlogn	Y	Y
7	TimeSort	n	nlogn	nlogn	Y	N
8	CubeSort	n^2	n^2	n^2	N	N
9	ShellSort	nlogn	n^(4/3)	n^(3/2)	N	Y
10	BubbleSort	n	n^2	n^2	Y	Y

Table 2: Any 5 non-comparisons based sorting algorithms

Sno	Name	Best	Average	Worst	Stable(Y/N)	In-Place(Y/N)
1	PigeonholeSort	-	n+2^k	n+2^k	Y	Y
2	BucketSort	-	n+k	n^2+k	Y	N
3	CountingSort	-	n+r	n+R	Y	Y
4	LSD RadixSort	n	n*k/d	n*k/d	Y	Y
5	MSD RadixSort	-	n*k/d	n*k/d	Y	N