SORTING – 2

Given 2 sorted array A and B, we need to merge them and get sorted array C.

A = [1, 4, 6, 7, 8, 9]
B = [3, 5, 14, 18]
C = [1, 3, 4, 5, 6, 7, 8, 9, 14, 18]

CODE:

```
def Merge2SortedArray(A, B):
    C = list()
    for x in A:
        C.append(x)

    for x in B:
        C.append(x)

    C.sort() # Time Complexity: O(N logn)
    return C

"""
Can we do this in O(N) time complexity, this means without using sort
"""
```

We compare A & B to find the least number in the given two sorted arrays.

The first element from A is compared with the first element in B and the least value between a_1 and b_1 will be appended to the empty array C.

If a_1 is appended to C array, then in the next step, b_1 will be compared to a_2 and the lesser value is appended to the C array.

These steps are repeated again and again until all the values of A and B are added to the empty list C in ascending order.

This gives us a sorted array C. The code would be as follows:

CODE:

```
def Merge2SortedArray2(A, B):
    n = len(A)
    m = len(B)

    p1 = 0 # this is the pointer for A array
    p2 = 0 # this is the pointer for B array

    while p1 < n and p2 < m: # as long as p1 is in n and p2 is in m
        if A[p1] < B[p2]:
            C.append(A[p1])
            p1 += 1
        else:
            C.append(B[p2])
            p2 += 1

    while p1 < n: # one of the pointers might not be complete
        C.append(A[p1])
        p1 += 1

    while p2 < m:
        C.append(B[p2])
        p2 += 1

    return C

A = [1, 4, 6, 7, 8, 9]
B = [3, 5, 14, 18]

C = []
print(Merge2SortedArray2(A, B))
```

OUTPUT:

[1, 3, 4, 5, 6, 7, 8, 9, 14, 18]

Merge Sort

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sub-lists until each sub-list consists of a single element and merging those sub-lists in a manner that results into a sorted list. With worst-case time complexity being O (n log n), it is one of the most respected algorithms. Merge sort first divides the array into equal halves and then combines them in a sorted manner.

First Step: divide the array into two parts
Second Step: merge the parts.

For list = [2, 12, 3, 7, 5, 4];

[2, 12, 3, 7, 5, 4]

l = 0, r = 5, mid = 2

[2, 12, 3] [7, 5, 4]

l = 0, r = 2, mid = 1 l = 3, r = 5, mid = 4

[2, 12] [3] [7, 5] [4]

l = 0, r = 1, mid = 0 l = 3, r = 4, mid = 3

---

[2] [12] [3] [7] [5] [4]

NOTE: every time it is divided, the from 0^{th} index to the (mid) index is taken into the left part and (mid + 1) index to the last element into right part.

This is the first step of this algorithm. In the second part, we will merge all the elements into one array.

---

[2] [12] [3] [7] [5] [4]

[2] & [12] are sorted array just as [7] & [5]

[2, 12] [3] [5, 7] [4]

[2, 12] & [3] are sorted array just as [5, 7] & [4] so,

[2, 3, 12] [4, 5, 7]

[2, 3, 12] & [4, 5, 7] are sorted array

[2, 3, 4, 5, 7, 12]

So, we can write it as,

```
def mergeSort (A, left, right):
     if left >= right:
          return
     mid = (left + right // 2
     mergeSort (A, left, mid)
     mergeSort (A, mid + 1, right)
     merge (A, left, mid, mid + 1, right)
```

CODE:

```
def merge(A, start1, end1, start2, end2):
    # start1 is the starting index of the left Array
    # end1 is the ending index of the left Array
    # start2 is the starting index of the right Array
    # end2 is the ending index of the right Array
    p1 = start1
    p2 = start2

    temp = list()

    while p1 <= end1 and p2 <= end2:
        if A[p1] < A[p2]:
            temp.append(A[p1])
            p1 += 1
        else:
            temp.append(A[p2])
            p2 += 1

    while p1 < end1:
        temp.append(A[p1])
        p1 += 1

    while p2 < end2:
        temp.append(A[p2])
        p2 += 1

    idx = 0
    while idx < len(temp):
        A[start1 + idx] = temp[idx]
        idx += 1
```

```
def mergeSort(A, left, right):
    if left >= right:
        return

    mid = (left + right) // 2
    mergeSort(A, left, mid)
    mergeSort(A, mid + 1, right)

    merge(A, left, mid, mid + 1, right)

if __name__ == "__main__":
    A = [5, 6, 2, 3, 66, 7, 1, 2, 2, 34, 5]
    print("Unsorted Array is ", A)
    mergeSort(A, 0, len(A) - 1)
    print("Sorted Array is ", A)
```

The time complexity of merge is O(n) and 2 * T(N/2) as we have called mergeSort function twice. So, the Time Complexity for this equation would be **O(nlogn)**

Quick Sort is also an algorithm, whose time complexity is **O(nlogn)**.

https://www.geeksforgeeks.org/merge-sort/