Name - Tun Myat

ID - 51705354

3. Part B (c) compare the asymptotic running time of the two sorting algorithms

Insertion sort uses an incremental method and merge-sort uses dive and conquer method by recursive. Merge-sort runs in O(n log(n)) time and insertion sort runs in $O(n^2)$ time. So, we can say that merge-sort runs faster in large input elements size. But insertion sort will run faster in small input elements size.

Binary Insertion Sort Algorithm

Insertion sort uses an incremental method. Binary Search is being used to reduce the number comparisons to find the necessary position. In the worst case, Binary Search will perform $\lceil \log^2(n) \rceil$ comparisons. But the whole algorithm will still run for $O(n^2)$ run time because we still have to shift all the elements greater than the selected value towards the end of the array to insert the value. So, the insertion sort will take very long time to finish all the sorting. For example, the insertion sort takes 2075.71 seconds to finish sorting for 2^{22} elements. If we compare the run time with merge-sort, merge-sort will take 0.763615 seconds only for 2^{22} elements.

Worst-case for insertion sort: input elements are reverse sorted

$$T(n) = \sum_{j=2}^{n} \theta(j) = \theta(n^2)$$

Binary Insertion Sort pseudo-code

```
BinarySearch (A, value, low, high)
        if high <= low
                if value > A[low]
                        return low+1
                else return low
        mid = (low + high)/2
        if value == A[mid]
                return mid+1
        else if value > A[mid]
                return BinarySearch (A, value, mid+1, high)
        else BinarySearch (A, value, low, mid-1)


BinaryInsertionSort ( N, A)
        for i← 1 to N, i++
                value = A[i]
                j = i – 1
                position = BinarySearch (A, value, 0, j)
                for j ← position from j, j--
                        A[j+1] = A[j]
                A[j+1] = value
```

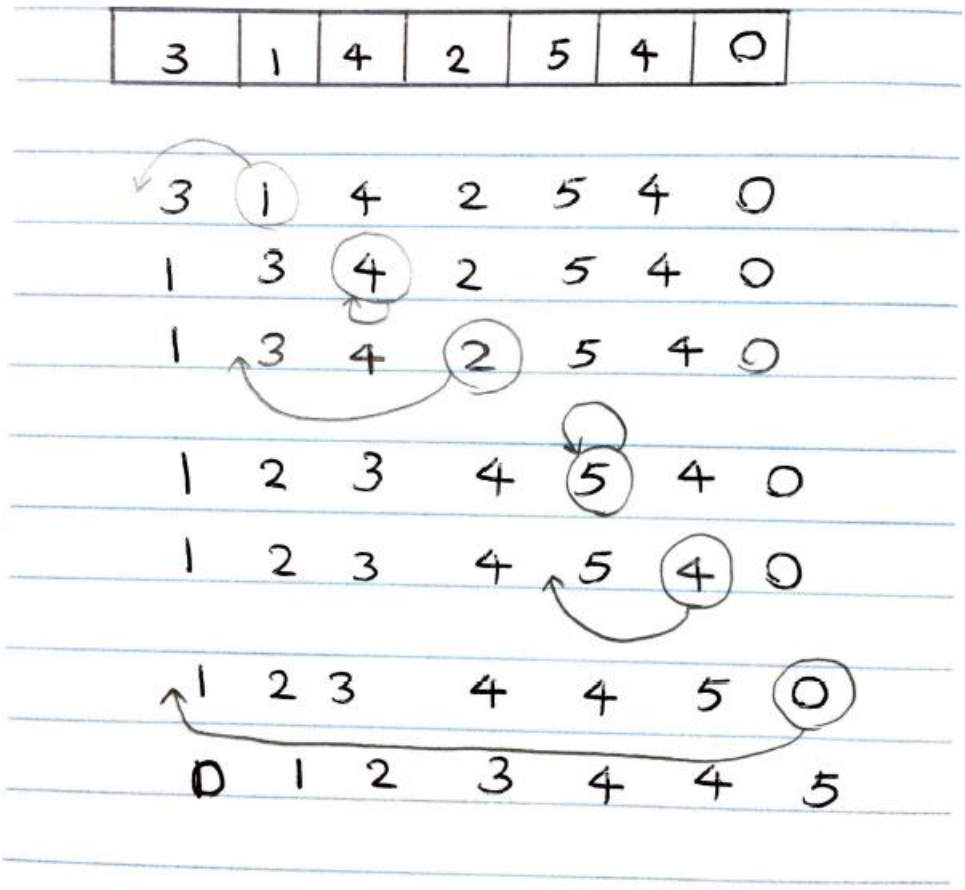


Fig 1. Sample working diagram for Binary Insertion Sort

```
Unsorted array:
3 1 4 2 5 4 0
Sorted array:
0 1 2 3 4 4 5
--------------------------------
Process exited after 0.03445 seconds with return value 0
Press any key to continue . . .
```

Fig 2. Sample working example for Binary Insertion Sort

Merge-sort Algorithm

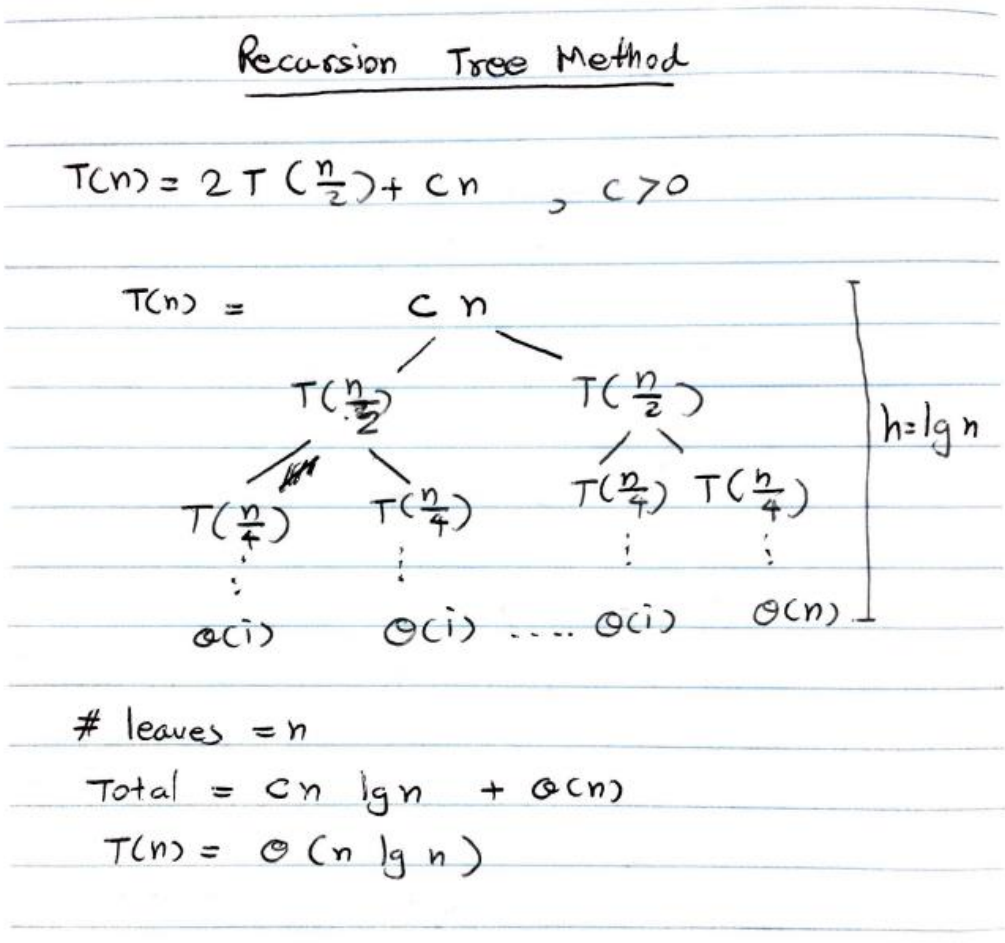
There is an extra function that is being used inside MergeSort function which is called Merge. Merge function will connect all the broken arrays into the original array.

MergeSort divides the original array into two smaller arrays. It will continue the division recursively until the array is broken down into single element. Then, it will compare two arrays at one time and connect them back into array in increasing order. The connecting process will continue recursively until all the elements are connected back into one original single array.

MergeSort is a divide and conquer algorithm by recursive. It has average and worst-case run time for O(n log (n)). It is a lot faster than the Insertion Sort in huge number of elements. For example, MergeSort runs for 13.8085 seconds to sort 2^{26} elements.

Algorithm	Run Time
Merge Sort: A[1, 2, 3,…n]	T(n)
1. If n = 1, done	$\theta(1)$
2. Recursively sort A[1,…$\lceil \frac{n}{2} \rceil$]	$2T(\frac{n}{2})$
3. Merge two sorted lists	$\theta(n)$

$$T(n) = \begin{cases} \theta(1) \; if \; n = 1 \\ 2T\left(\frac{n}{2}\right) + \theta(n) \; if \; n \geq 1 \end{cases}$$



Merge-sort pseudo-code

```
Merge(A, leftArr, leftSize, rightArr, rightSize)
while i < leftSize && j < rightSize
        if leftArr[i] < rightArr[i], k++, i++
                A[k] = leftArr[i]
        else k++, j++
                A[k] = rightArr[j]
while i < leftSize, k++, i++
        A[k] = leftArr[i]
while j < RightSize , k++, j++
        A[k] = rightArr[j]


MergeSort(N, A)
        if N <= 1
                return NIL
        mid = N/2
for i ← 0 to mid, i++
        leftArray[i] = A[i]
for j ← mid to N
        rightArray[ j - mid] = A[j]
MergeSort(mid, leftArray)
MergeSort(N - mid, rightArray)
Merge(A, leftArray, mid, rightArray, N-mid)
```

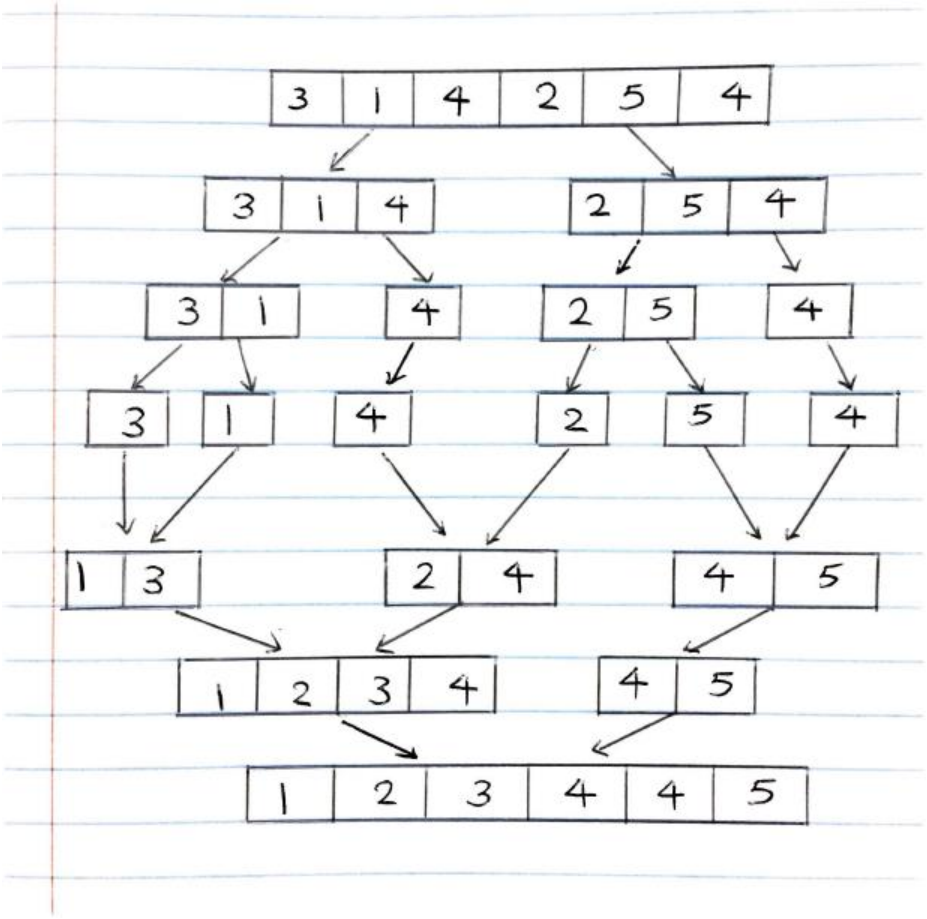


Fig 3. Sample working diagram for Merge-sort

```
Unsorted Array:
3 1 4 2 5 4
Sorted Array:
1 2 3 4 4 5

--------------------------------
Process exited after 0.03282 seconds with return value 0
Press any key to continue . . .
```

Fig 4. Sample working example for Merge-sort

Running time for different Sorting Algorithm				
		Quicksort	Binary Insertion Sort	Merge-sort
N Value	N Value	Time (Sec)	Time (Sec)	Time (Sec)
100		9.00E-06	5.00E-06	2.90E-05
1000		0.000182	0.000151	0.000119
2^0	1	3.00E-06	0	0
2^1	2	2.00E-06	0	2.00E-06
2^2	4	2.00E-06	0	2.00E-06
2^3	8	3.00E-06	0	4.00E-06
2^4	16	3.00E-06	0	4.00E-06
2^5	32	4.00E-06	1.00E-06	6.00E-06
2^6	64	6.00E-06	3.00E-06	9.00E-06
2^7	128	9.80E-05	6.00E-06	1.70E-05
2^8	256	9.50E-05	1.80E-05	3.00E-05
2^9	512	0.000113	5.10E-05	5.80E-05
2^10	1024	0.000162	0.000156	0.000139
2^11	2048	0.000246	0.000522	0.000267
2^12	4096	0.000436	0.001924	0.000584
2^13	8192	0.000865	0.007252	0.001148
2^14	16384	0.001736	0.027616	0.002227
2^15	32768	0.003655	0.111348	0.004794
2^16	65536	0.007668	0.486039	0.009362
2^17	131072	0.016204	2.20582	0.01942
2^18	262144	0.034265	8.59904	0.040408
2^19	524288	0.070234	34.964	0.08367
2^20	1048576	0.14833	132.105	0.172331
2^21	2097152	0.310963	520.286	0.352463
2^22	4194304	0.636911	2075.71	0.763615
2^23	8388608	1.34714		1.56306
2^24	16777216	2.7798		3.37687
2^25	33554432	5.80006		6.70121
2^26	67108864	12.0615		13.8085