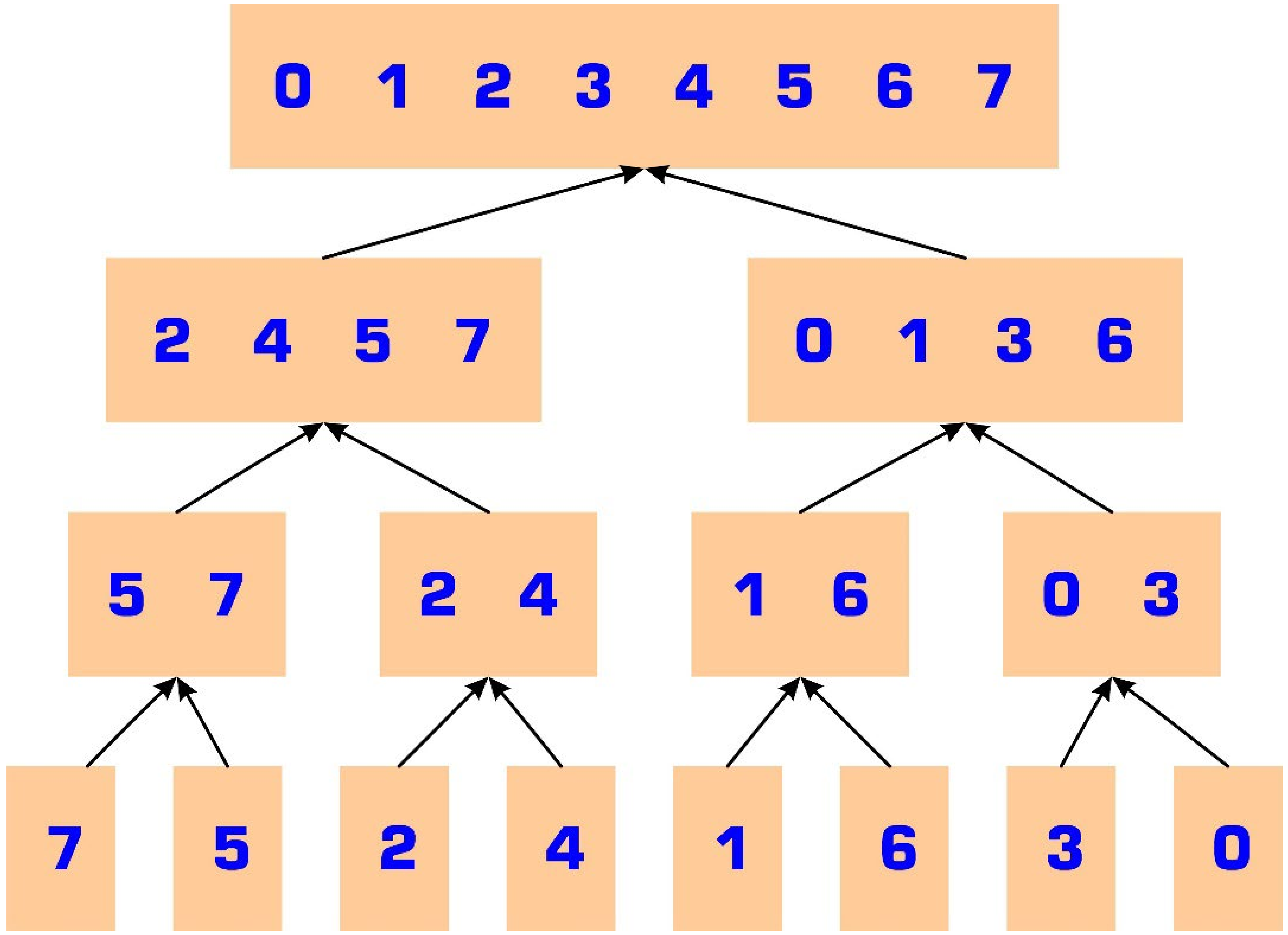
Analysis of Merge Sort

(Class 8)

From Book’s Page No 36 (Chapter 2)

- We started divide and conquer.
- We also discussed shortly the process of the merge sort.
- It is based on recursion.
- Recursive algorithms should end at some point otherwise it will stick in infinite loop.

- In the merge sort, the divide recursion will end when we reach atomic level.
- Means all the integers are divided until we got single integer.
- Then we will start the merging process.


0 1 2 3 4 5 6 7
2 4 5 7
0 1 3 6
5 7
2 4
1 6
0 3
7
5
2
4
1
6
3
0

- According to the diagram the individual parts will combine.
- Here we will say this combining phase of divide and conquer as sorting.

Merge Sort Algorithm (Book's Page No 39)

```
MERGE-SORT (array A, int p, int r)
if p ≥ r                        // zero or one element?
    return
q ← ⌊(p + r) / 2⌋              // midpoint of A[p:r]
MERGE-SORT (A, p, q)           // recursively sort A[p:q]
MERGE-SORT (A, q+1, r)         // recursively sort A[q+1:r]
// Merge A[p:q] and A[q+1:r] into A[p:r].
MERGE (A, p, q, r)
```

- At line 4, the algorithm will first keep dividing left side of the input array up to single integers.
- Then at line 5, the right side of the array will be divided.
- If we have parallel computer or parallel processing, we can run line 4 and 5 at the same time.
- Then at the end on line 7 the merging process will be performed.

The Merge Part

```
MERGE (array A, int p, int q, int r)
int B[p…r];                           // a temporary array
int i ← k ← p
int j ← q + 1
while (i ≤ q) and (j ≤ r)
if (A[i] ≤ A[j])
    then B[k++] ← A[i++]
    else B[k++] ← A[j++]
while (i ≤ q)
    do B[k++] ← A[i++]
while (j ≤ r)
    do B[k++] ← A[j++]
for i ← p to r
    do A[i] ← B[i]
```

Analysis of Merge Sort

- First, consider the running time of the procedure merge(A, p, q, r).
- Let $n = r - p + 1$ denote the total length of both left and right sub-arrays, i.e., sorted pieces.
- Take any two subarrays from previous diagram and check their number of elements.
- The merge procedure contains four loops, none nested in the other.

- Each loop can be executed at most n times.
- So, running time of merge part is:

$$T(n) = n + n + n + n$$

$$T(n) = 4n$$

- Thus, running time of merge is:

$$\boldsymbol{T(n) = \theta(n)}$$

- The merge procedure will take at least n running time.
- All the 4 loops will nun n times in worst case.
- Let $T(n)$ denote the worst-case running time of MergeSort on an array of length n.
- If we call MergeSort with an array containing a single item ($n = 1$) then the running time is constant.
- We can just write $T(n) = 1$, ignoring all constants.

- For $n > 1$, MergeSort splits into two halves. sorts the two and then merges them together.
- The left half is of sizer $\left\lceil \frac{n}{2} \right\rceil$ and the right half is $\left\lfloor \frac{n}{2} \right\rfloor$.
- Both halves will not equal if the array has odd number of elements.
- To show this we use floor and ceiling operators.

- How long does it take to sort elements in sub array of sizer $\left\lceil\frac{n}{2}\right\rceil$?
- We do not know this but because $\left\lceil\frac{n}{2}\right\rceil < n$ for $n > 1$, we can express this as:

$$T(\left\lceil\frac{n}{2}\right\rceil)$$

- Similarly, the time taken to sort right sub array is expressed as $T(\left\lfloor \frac{n}{2} \right\rfloor)$.
- In conclusion we have:

$$T(n) = \begin{cases} 1\,, & if\ n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n, & if\ n > 1 \end{cases}$$

- This is called recurrence relation, i.e., a recursively defined function.

- In mathematics, a recurrence relation is an equation according to which the n^{th} term of a sequence of numbers is equal to some combination of the previous terms.
- For example, in mathematics, the *Fibonacci numbers*, commonly denoted F_n, form a sequence, the Fibonacci sequence, in which each number is the sum of the two preceding ones.
- The sequence commonly starts from 0 and 1, the first few values in the sequence are:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

- You studied recurrence relations in discrete mathematics.

- Here in the second part of the equation, the $T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ is the required time of sorting two sub arrays.
- While the n term is the running time of the merge procedure we calculated earlier.

$$T(n) = \begin{cases} 1, & if\ n = 1 \\ T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n, & if\ n > 1 \end{cases}$$

- Here, $n = 1$ is the terminating condition of this recurrence relation.

Solving the Recurrence

- $T(1) = 1$
- $T(2) = T(1) + T(1) + 2 = 1 + 1 + 2 = 4$
- $T(3) = T(2) + T(1) + 3 = 4 + 1 + 3 = 8$
- $T(4) = T(2) + T(2) + 4 = 8 + 8 + 4 = 12$
- $T(5) = T(3) + T(2) + 5 = 8 + 4 + 5 = 17$
- $\dots$
- $T(8) = T(4) + T(4) + 8 = 12 + 12 + 8 = 32$
- $\dots$
- $T(16) = T(8) + T(8) + 16 = 32 + 32 + 16 = 80$
- $\dots$
- $T(32) = T(16) + T(16) + 32 = 80 + 80 + 32 = 192$

- Now, how long we continue this recurrence?
- Our want to make a closed form expression.
- What is the pattern here?

- Let's consider the ratios $\frac{T(n)}{n}$ for powers of 2.
- For example, $\mathrm{n} = 2^1, 2^2, 2^3$, etc.
- $T(1)/1 = 1$
- $T(2)/2 = 2$
- $T(4)/4 = 3$
- $T(8)/8 = 4$
- $T(16)/16 = 5$
- $T(32)/32 = 6$

- This suggests:

$$\frac{T(n)}{n} = \log_2 n + 1$$

$$T(n) = n\,(\log_2 n + 1)$$

$$T(n) = n \log_2 n + n$$

- The dominating term is $n \log_2 n$.
- Therefore, the running time of the merge sort is:

$$\boldsymbol{O(n\, log_2\, n)}$$