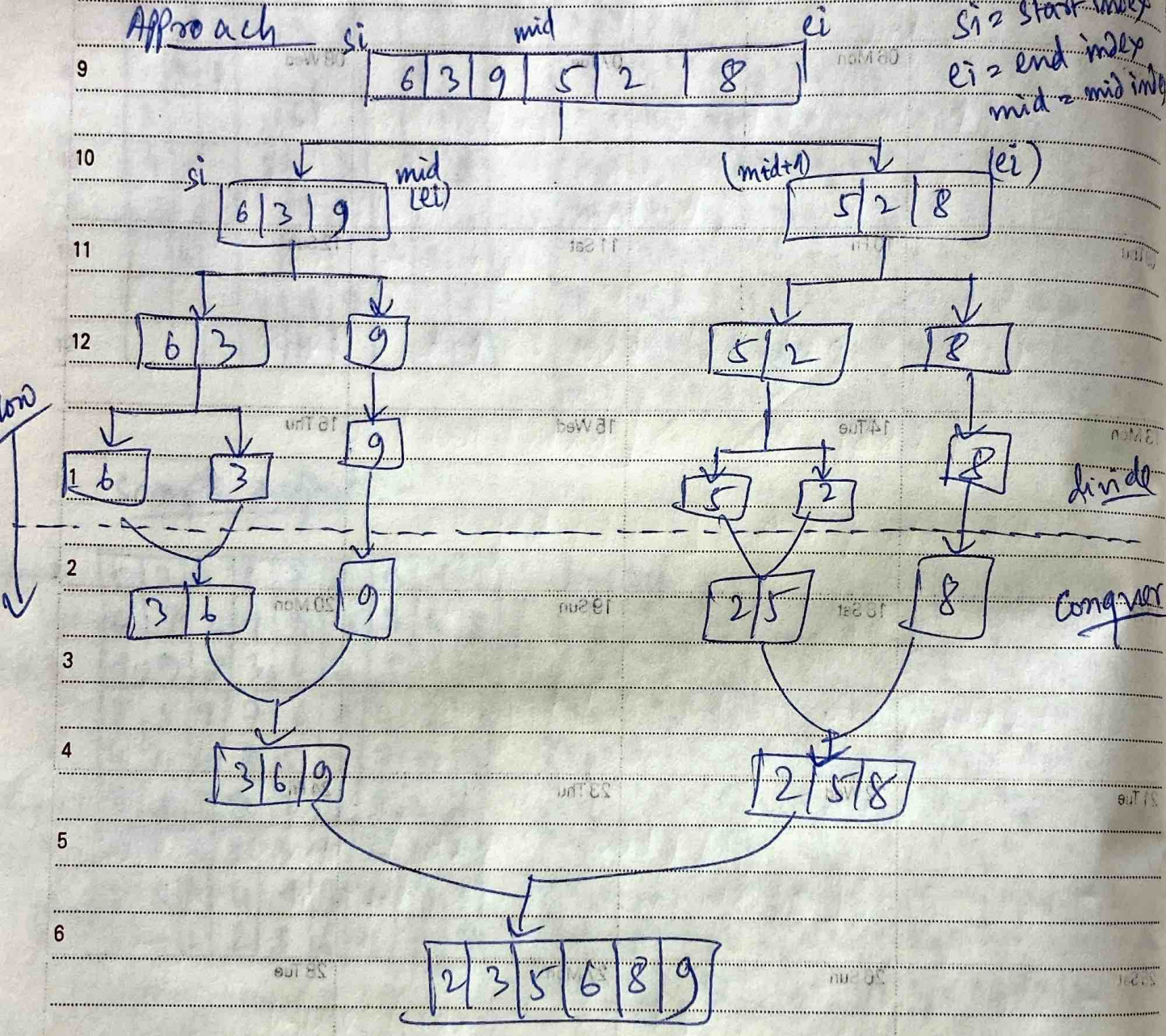
# Merge Sort

✓ It is based on a technique called 'divide & conquer'.

Approach




For the divide part, we will always find the starting index (si), middle index (mid) & the ending index (ei)

✓ Now in every step, we shall divide the above array into 2 parts - 1st part from starting index to mid, then the 2nd part from (mid+1) index to end (ei) index and so on.

2022 JULY

June 23rd Wk • 153-212

02 Thursday

For the conquer part, we simply check which is the smallest element and put it in first in the new array. Similarly, we keep on putting new elements by comparing in the new array. Finally, we copy the new array to the original array.

sorted

| 3 | 6 | 9 |
|---|---|---|

index = i

sorted

| 2 | 5 | 8 |
|---|---|---|

index = j

Step 1:- Compare i=0 with j=0 & check

new array = | 2 | | | | | |

Step 2:- Compare i=0 with j=1

new array = | 2 | 3 | | | | |

Step 3:- Compare i=1 with j=1

new array = | 2 | 3 | 5 | | | |

Step 4:- Compare i=1 with j=2

new array = | 2 | 3 | 5 | 6 | | |

step 5:-

Compare i=2 with j=2

new array = | 2 | 3 | 5 | 6 | 8 | |

Now, j=2 is the last element, so we will put i=2 in the new array and finally we copy the new array in the original array.

newarray = | 2 | 3 | 5 | 6 | 8 | 9 |

03 June
23rd Wk • 154-211
Friday

2022 JUNE

| M | T | W | T | F | S | S |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | | | |

```
class mergeSort
{
    Public static void conquer (int arr[], int si, int mid,
                                int ei)
    {
        int merged[] = new int [ei-si+1];
        int idx1 = si;
        int idx2 = mid+1;
        int x = 0;

        while (idx1 <= mid && idx2 <= ei)
        {
            if (arr[idx1] <= arr[idx2])
            {
                merged[x++] = arr[idx1++];
            }
            else
            {
                merged[x++] = arr[idx2++];
            }
        }

        while (idx1 <= mid) {
            merged[x++] = arr[idx1++];
        }

        while (idx2 <= ei) {
            merged[x++] = arr[idx2++];
        }

        for (int i=0, j=si; i<merged.length; i++, j++)
        {
            arr[j] = merged[i];
        }
    }
}
```

```
Public static void divide (int arr[], int si, int ei)
{
    if (si >= ei)
    {
        return;
    }
    int mid = si + (ei-si)/2;
    divide (arr, si, mid);
    divide (arr, mid+1, ei);
    conquer (arr, si, mid, ei);
}

Public static void main (String args[])
{
    int arr[] = {6, 3, 9, 5, 2, 8};
    int n = arr.length;
    divide (arr, 0, n-1);
    // Print the resulting sorted array
    for (int i=0; i<n; i++)
    {
        System.out.print (arr[i] + " ");
    }
    System.out.println();
}
```

* Time complexity = $O(n \log n)$. To divide all the elements and finally coming to singular element takes $\log_2 n$ times. on the other hand, to join/conquer it's taking n times. So, $O(n \log n)$ is the total time of the whole process.