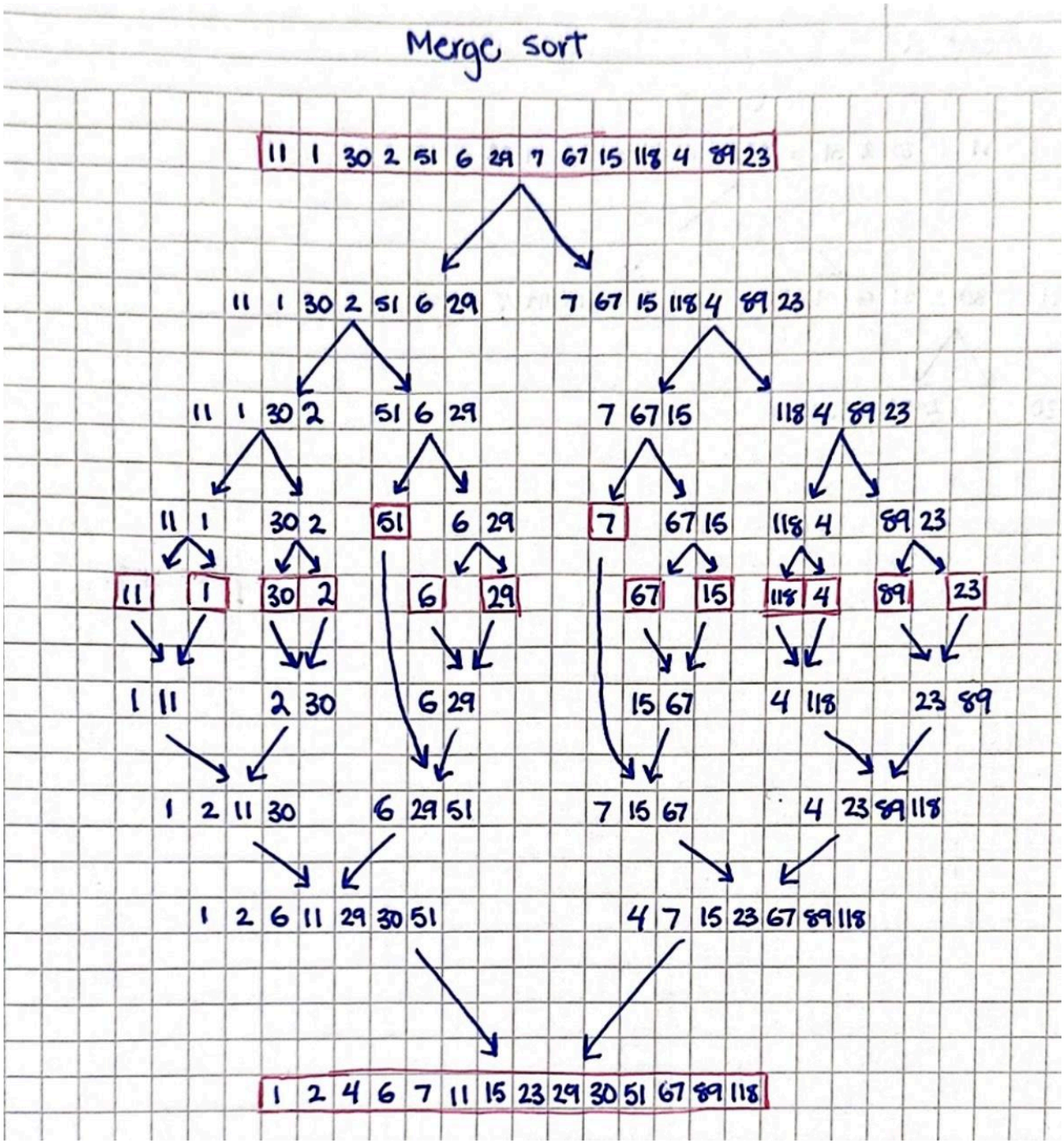
1. Introduction

→ the objective of this assignment is to gain an understanding of sorting an array of product ID with merge sort by tracing their execution, and visualizing the sorting process along with a sound effect.

2. Detailed visualization



→ Best case: O(n log n)
→ Worse case: O(n log n)

3. Code implementation

https://github.com/ayshabilai/algorithms-and-data-sturctures-assignment-two/blob/main/assignment%20two

```
##############################################################################

import winsound
import os

def merge_sort(arr, depth=0):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]

        merge_sort(L, depth + 1)
        merge_sort(R, depth + 1)

        i = j = k = 0

        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
            winsound.Beep(1000, 100)  # Sound effect for swap
            print(f"Depth {depth}: {arr}")

        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
            winsound.Beep(1000, 100)  # Sound effect for swap
```

```
            print(f"Depth {depth}: {arr}")

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
            winsound.Beep(1000, 100)  # Sound effect for swap
            print(f"Depth {depth}: {arr}")

if __name__ == '__main__':
    arr = [11, 1, 30, 25, 16, 29, 7, 67, 15, 18, 4, 89, 23]
    print(f"Original array: {arr}")
    merge_sort(arr)
    print(f"Sorted array: {arr}")

################################################################################end
    Original array: [11, 1, 30, 25, 16, 29, 7, 67, 15, 18, 4, 89, 23]
    Depth 3: [1, 30]
    Depth 3: [1, 30]
    Depth 2: [1, 1, 30]
    Depth 2: [1, 11, 30]
    Depth 2: [1, 11, 30]
    Depth 3: [16, 29]
    Depth 3: [16, 29]
    Depth 2: [16, 16, 29]
    Depth 2: [16, 25, 29]
    Depth 2: [16, 25, 29]
    Depth 1: [1, 1, 30, 25, 16, 29]
    Depth 1: [1, 11, 30, 25, 16, 29]
    Depth 1: [1, 11, 16, 25, 16, 29]
    Depth 1: [1, 11, 16, 25, 16, 29]
    Depth 1: [1, 11, 16, 25, 29, 29]
    Depth 1: [1, 11, 16, 25, 29, 30]
    Depth 3: [15, 15]
    Depth 3: [15, 67]
    Depth 2: [7, 67, 15]
    Depth 2: [7, 15, 15]
    Depth 2: [7, 15, 67]
    Depth 3: [4, 4]
    Depth 3: [4, 18]
    Depth 3: [23, 23]
    Depth 3: [23, 89]
    Depth 2: [4, 4, 89, 23]
    Depth 2: [4, 18, 89, 23]
    Depth 2: [4, 18, 23, 23]
```

```
Depth 2: [4, 18, 23, 89]
Depth 1: [4, 67, 15, 18, 4, 89, 23]
Depth 1: [4, 7, 15, 18, 4, 89, 23]
Depth 1: [4, 7, 15, 18, 4, 89, 23]
Depth 1: [4, 7, 15, 18, 4, 89, 23]
Depth 1: [4, 7, 15, 18, 23, 89, 23]
Depth 1: [4, 7, 15, 18, 23, 67, 23]
Depth 1: [4, 7, 15, 18, 23, 67, 89]
Depth 0: [1, 1, 30, 25, 16, 29, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 30, 25, 16, 29, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 25, 16, 29, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 16, 29, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 29, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 7, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 67, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 15, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 25, 18, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 25, 29, 4, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 25, 29, 30, 89, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 25, 29, 30, 67, 23]
Depth 0: [1, 4, 7, 11, 15, 16, 18, 23, 25, 29, 30, 67, 89]
Sorted array: [1, 4, 7, 11, 15, 16, 18, 23, 25, 29, 30, 67, 89]
```

4. Conclusion

→ detailed and implemented the Merge Sort algorithm, demonstrating its divide-and-conquer approach to sorting an array. provided a step-by-step visualization of the sorting process and implemented it in Python, ensuring clarity on its operation. The algorithm's time complexity is consistently O(n log n), making it efficient for large datasets.