HW1

Aditya Kumar (2022033), Nishant (2022326)

January 2024

1 Assumption:

- A, B, and C are three sorted arrays.
- Index starts from 0 to $n - 1$.
- *Smallest* is initialized as the minimum value among the first elements of the three arrays.
- *Largest* is initialized as the maximum value among the last elements of the three arrays.
- *Mid* is the mean of *smallest* and *largest*.

2 Code

```
FindKthSmallestElement(A, B, C, k):
    1. smallest = min(A[0], B[0], C[0])   // Smallest element of A U B U C
    2. largest = max(A[n-1], B[n-1], C[n-1])   // Largest element of A U B U C
    3. mid = smallest + (largest - smallest) // 2

    4. while smallest < largest:
        5. count = BinarySearch(A, mid) + BinarySearch(B, mid) + BinarySearch(C, mid)

        6. if count < k:
            smallest = mid + 1
        7. else:
            largest = mid

    8. return smallest
```

Algorithm

This function performs binary search on *mid* in all three arrays and returns the sum of elements on the left side of *mid*.

If $count$ **is less than** k**, it means the** k**-th smallest element is in the right subarray, so** $smallest = mid + 1$**.**

If $count$ **is greater than or equal to** k**, it means the** k**-th smallest element is in the left subarray, so** $largest = mid$**.**

The while loop ends when $smallest$ **and** $largest$ **converge.**

The function returns the k**-th smallest element.**

```
def BinarySearch(arr, target):
    9. low, high = 0, n-1

    10. while low < high:
        11. mid = low + (high - low) // 2

        12. if arr[mid] <= target:
            low = mid + 1
        13. else:
            high = mid

    14. return low
```

3 Time Complexity

1, 2, 3: Time complexity $O(1)$

4: Binary search time complexity $O(\log n)$

5: BinarySearch() time complexity $3 \cdot \log n$ **(binary search)** $= O(\log n)$

6, 7: Time complexity $O(1)$

Time complexity of the entire function is $O(\log n)$

Comparison between $O(n)$ **and** $O((\log n)^2)$

Differentiate:

$$\frac{d}{dn}(n) = 1$$

$$\frac{d}{dn}(\log n^2) = \frac{2 \log n}{n}$$

$$\lim_{n \to \infty} \frac{2 \log n}{n} = 0$$

As n tends to infinity, n increases much faster than $\log n$. $\lim_{n \to \infty} \frac{2 \log n}{n} = 0$. So, $O(n) > O((\log n)^2)$.