# Project 1

## Part A

We developed a randomized fuzzy sorting algorithm. For this we first read the data from the text file using the pandas library. The data frame is not in the $[a_i, b_i]$ format so we call the create intervals function. This function gives us a list of intervals of a specific size

Algorithm 1: Fuzzy Sorting

```
1  fuzzy_sort(arr, begin, end):
2      if begin < end:
3          q, t = random_partition(arr, begin, end)
4          fuzzy_sort(arr, begin, q - 1)
5          fuzzy_sort(arr, t + 1, end)
6      end
```

Instead of just calling the partition function as in quick sort we call the randomized partition function which has slight change over partition. i.e. Instead of always choosing the last element to be the pivot we randomly choose a pivot element.

Algorithm 2: Random Partition

```
1  random_partition(arr, begin, end):
2      i = randint(begin, end)
       # swap pivot with last element this helps us to use the quicksort
       # algorithm even though we have a random pivot
3      swap(arr[i], arr[end])
4      return partition(arr, begin, end)
```

Algorithm 3: ]Partition [3]

```
1  partition(arr, begin, end):
2      pivot = arr[end]
3      q = begin - 1  # middle begin
4      t = begin - 1  # middle end
5      for i in range(begin, end):
6          if (arr[i][0] <= pivot[1] and pivot[0] <= arr[i][1]): # right
7              t++
8              swap(arr[t], arr[i])
9          end
10         elif arr[i][1] < pivot[0]: # left-middle
11             t++
12             q++
13             swap(arr[t], arr[q])
14             if t != i:
15                 swap(arr[i], arr[q])
16             end
17         end
18     end
19     swap(arr[t + 1], arr[end])
20     return q + 1, t + 1
```

# Part B

- For Small Overlapping:

  The algorithm for fuzzy sorting is very similar to randomized quick sort. Therefore the worst case running time complexity should also be similar to that of randomized quick sort i.e. $O(nlogn)$.

  If there is no overlapping then the middle region will be the pivot interval only and Algorithm 3 will run in $\Theta(n)$. As the pivot is selected randomly the left and right sub arrays will be $\frac{n}{2}$ just like quick sort. The overall time complexity of fuzzy sorting will be twice the length of each sub array plus the time taken by random partition.[1][2]

$$T(n) = \left\{ \begin{array}{ll} \Theta(0) & \text{if } n = 0, \\ \\ 2 \cdot T\left[\frac{n}{2}\right] + \Theta(n) & \text{if } n > 0 \end{array} \right\}$$

$$T(n) = O(nlogn) \text{ . . .from quick sort}$$

- For All Overlapping.

  In the best case scenario all the intervals will overlap.. Due to this the left and right sub arrays will be of size zero. Due to which the recursion functions will run in constant time $O(1)$ and the overall time complexity of fuzzy sort will be $\Theta(n)$ i.e. the time taken by the random partition function.[1][2]

# Part C

By passing different sizes to the create intervals function we create sub arrays of the given array of intervals. We then calculate the time required to fuzzy sort the sub array and plot a *Size vs Time* graph.

From Figure 1 we can see the time taken to sort a list with almost no over lap is in the time complexity $O(nlogn)$ where as the data with overlapping is sorted in linear time $O(n)$. By comparing Figure 1 with Figure 2 we can get a better understanding of the time complexities.

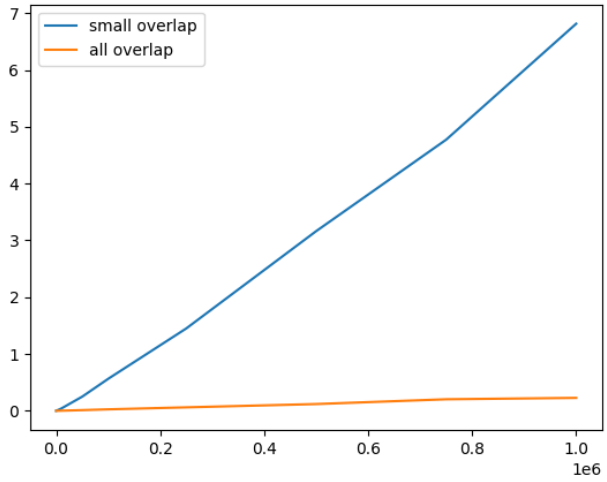


Figure 1: Plots for Fuzzy sorting

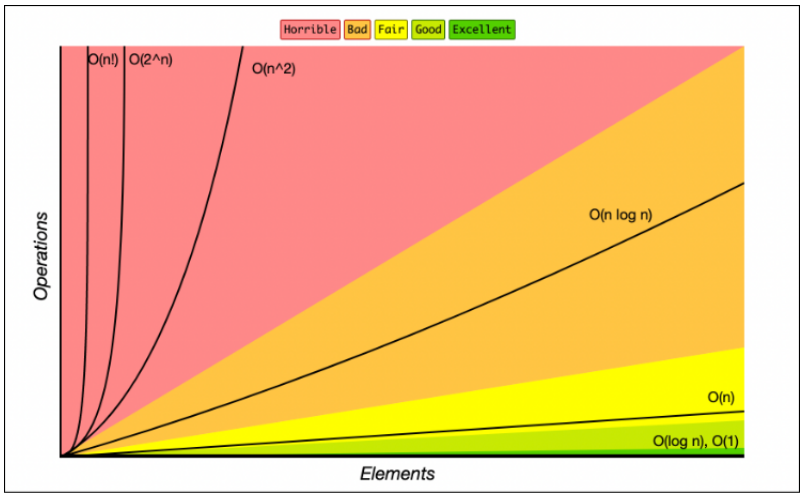


Figure 2: Big O Time Complexity Graph

# References

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. The MIT Press and McGraw-Hill, 2001.

2. Lanman, D. R. (2006, March 13). CS 157: Assignment 3 [Class handout]. Retrieved from http://alumni.media.mit.edu/ dlanman/courses/cs157/HW3.pdf

3. Fuzzy_sorting_of_intervals.Py. Frederick-S/Introduction-To-Algorithms-Code (Github.com). Retrieved March 27, 2023, from https://github.com/Frederick-S/Introduction-to-Algorithms-Code/blob/cca8aa46bbfb8f73c6ef1f7301ce574d5d4f64d8/src/chapter_7/fuzzy_sorting_of_intervals.py