CS 325: Analysis of Algorithms
Group Assignment 1

Hyun Taek Oh
Busra Dedeoglu
Yu-Hao Shih

January 30, 2024

1 Report A:

There are two arrays A[1,...,m] and B[1,...,n], which include integers between t_{min} and t_{max}. We wrote a code, which finds when a∈A and b∈B and a+b in [t_{min}, t_{max}], it finds valid pairs' count.

Step 1: The code reads inputs, which are given in input.txt. We have vecA and vecB, which represent array A and Array B. We used merge_sort function, which sorts an input array in non-decreasing order. Merge sort function sorts a vector in $O(n \log n)$ time complexity.

Step 2: To find the number of valid pairs, we needed to find number of pairs, T^{-}_{max} whose sum is less than t_{max}, which contains pairs that also sums to less than t_{min}, T^{-}_{min}. Therefore, we also find the number of pairs that sum to less than t_{min} which are out of the desired interval $[t_{min}, t_{max}]$. The difference $T^{-}_{max} - T^{-}_{min}$ results with the number of pairs that sum to a value in $[t_{min}, t_{max}]$. For a given element $a \in A$, it only takes $O(\log n)$ complexity to find the maximum element $b \in B$ such that $a + b \leq t_{max}$ or $a + b < t_{min}$ using binary search algorithm. Binary search algorithm uses division and creates a tree for search with $\log n$ stages. It first finds the mid index in A and checks if this value is less/higher than t_{max}. If it is higher, then it only searches upper portion of A recursively using binary search again.

Step 3: The get_sum_of_pairs function goes through each number in the first array vecA. For each of these numbers, it modifies the range based on that specific number. Then, it counts how many pairs can be formed by combining this number with elements from the second array vecB within the adjusted range. This counting is done using binary search functions. Therefore, the total complexity becomes $O(n \log m)$.

Step 4: The print_to_output function saves the final result to output.txt file. It takes the calculated value and writes it into the output.txt file. The code uses specific instructions given when starting the program from the command line. It takes the names of input and output files as inputs. Then, it reads the numbers from the input file, figures out how many pairs meet certain conditions, shows that number on the screen, and saves it in the output file.

2 Calculating Running Time Of Problem A

The code uses a method called merge sort to arrange numbers in a certain order. This method takes a reasonable amount of time, specifically $O(n \cdot \log n)$, where n is the number of elements. Mergesort uses a divide-and-conquer methodology, where it recursively splits input array into 2 pieces and applies mergesort again to each parts. This results with at most *logn* many stages in the recursion tree. At every stage it takes n operations to merge two split parts in sorted order by using 2 pointers, resulting with a total complexity of $O(n \log n)$.

In addition to this, another for loop is used to span vector A. For every element $a \in A$, we need to search for $b \in B$ that sum to values less than t_{min} and t_{max}. Binary search algorithm uses division and creates a tree for search with $\log n$ stages (similar to splits from half as in merge sort). It first finds the mid index in A and checks if this value is less/higher than t_{max}. If it is higher, then it only searches upper portion of A recursively using binary search again, resulting with complexity $O(\log m)$

Since these two sections are run one-after-another, the total complexity is always less than $O((n + m)\log(n + m))$ depending on the values of n, m.

3 Proof of Correctness Of A

To prove that the proposed algorithm solves the problem, we need to show that mergesort always successfully sorts vectors and binary search is guaranteed to find the max element in B less than t_{min}, t_{max}. For any given vector mergesort will process every value in an vector during divide stage. During merge stage, it can be shown that a vector X and a vector Y will be merges in a sorted manner, i.e., there will be 2 pointers and the smallest two elements $x \in X, y \in Y$ at any given time will be compared. Therefore, there cannot be any $x \in X, y \in Y$ such that $x < y$ and sorted$= [y, x]$, because otherwise x must be compared to an $y^* < y$ and $x > y^*$, which is a contradiction. Therefore, at every merge, the output of the recursion must be sorted. Since same happens with every recursion, mergesort is guaranteed to provide a sorted output.

Similarly, binary search will search for a value z by dividing the input vector into 2 halves. Then one half H will be selected if $z \in H^1$. Since this will be done recursively, $z \in H^k, \forall k = 1, ..., \log n$. Therefore, binary search is also guaranteed to find z. Since binary search will work for any selected $a \in A$ and hence z, the proposed algorithm finds all possible number of valid pairs.