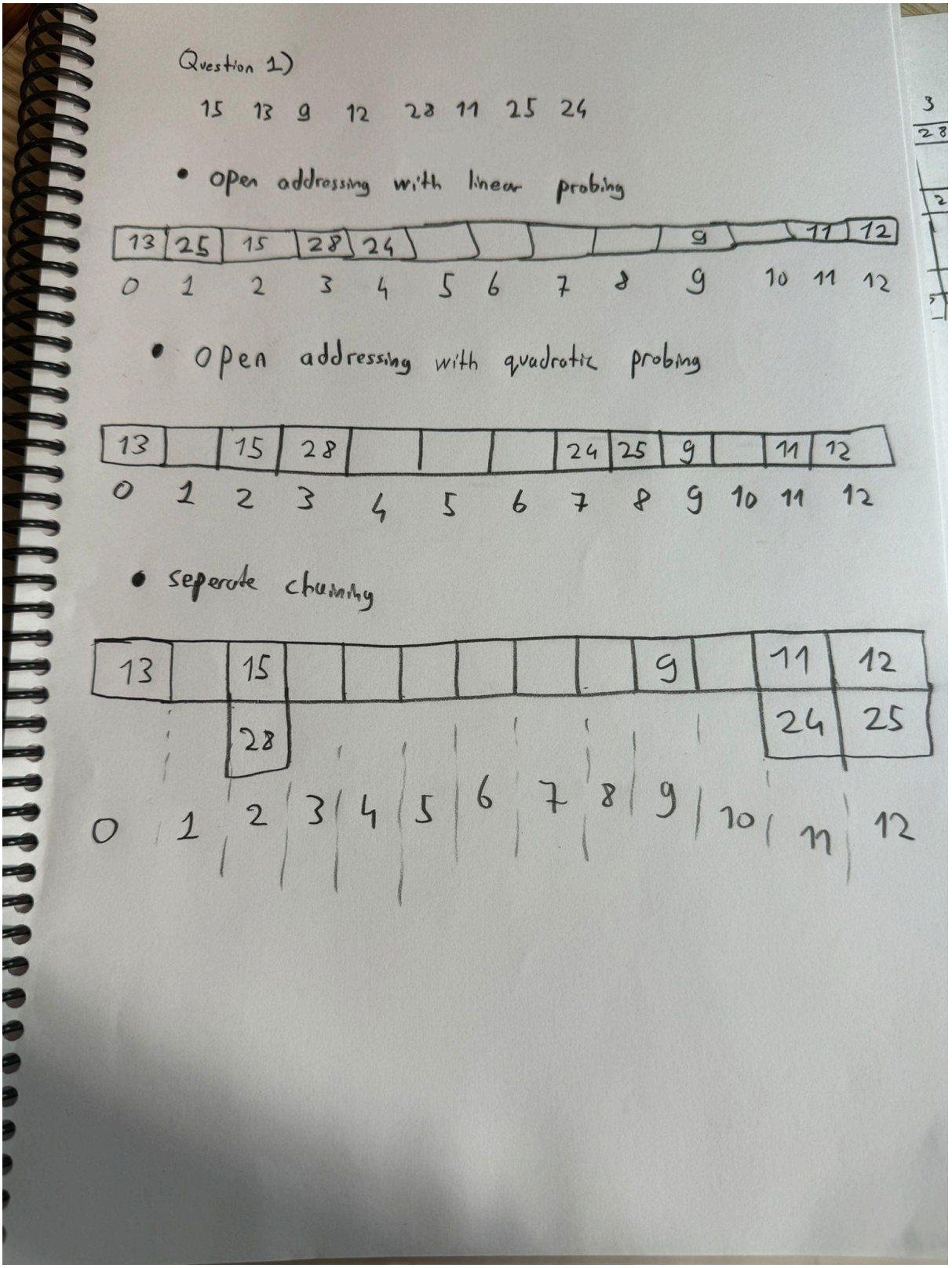
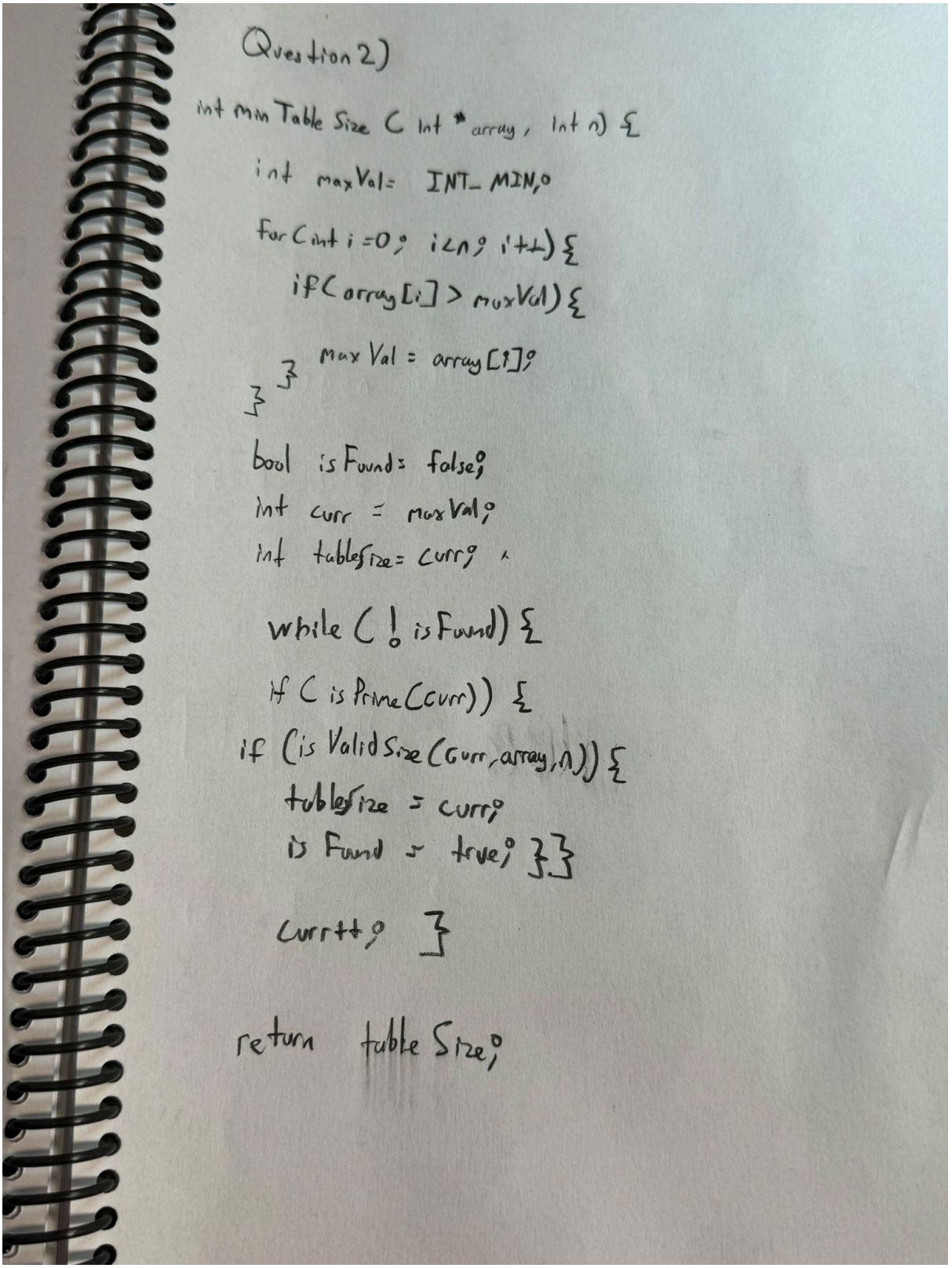
Berke Kuzey Ardıç
22103340
Section 1
CS202 HW 3

Question1)



Question2)



```
Question 2)

int minTableSize (int *array, int n) {
    int maxVal= INT_MIN;
    for(int i=0; i<n; i++) {
        if(array[i] > maxVal) {
            maxVal = array[i];
        }
    }

    bool isFound= false;
    int curr = maxVal;
    int tableSize= curr;

    while(!isFound) {
        if(isPrime(curr)) {
            if(isValidSize(curr, array, n)) {
                tableSize = curr;
                isFound = true; }}

        curr++; }

    return tableSize;
```

```
bool isPrime(int a){

if(a<=1)   return false;

if(a<=3)   return true;

for(int i=2; i<=sqrt(a); i++){

    if (divides(a,i))
     return false;
}

return true; }

bool divides( int a, int i){

  while(a>=i)
  a=a-i;

  return a==0; }

bool isValidSize(int curr, int array, int n){
 array.sort();
 for(int i=0; i<n-2; i++){
  for(int j=i+1; j<n; j++){
    if(divides(array[j]-array[i], curr))
    return false; } }

    return true;
}
```

Question4)

Subtask1:
Subtask1 was related to finding the number of suffix and prefixes in the n unique strings. The time complexity constraint was O(total_number_of_characters). My attempt aims to solve this by using a hashtable of my implementation and a simple rolling hash algorithm. My hashfunction and hashtable uses a polyndromical hashing function with a modulo of size which is chosen a very large prime number in order to avoid possible collision. My hashtable simply is a dynamic array of linkedlists, each linkedlist represents the strings with the same hashvalue. When a string is inserted it calculates the index using the hashfunction after that inserts to a linkedlists of the given index so it prevents collisions. While checking for an existence for elements it has a method which takes the string and it's hash value so it doesn't needs to compute the hash than it checks the linkedlist of the given hash index and if item exists there returns true. I designed this method to use with the rollinghash so at each iteration I compute the hash and time complexity at best case is equal to O(total_number_of_characters). In the worst case which is the case many collision happens and each time we pass the whole linkedlist time complexity becomes O($total\_number\_of\_characters^2$) in order to avoid this one of the possible ways is using different hash values for each string and holding them at the linkedlist. I also use one loop to calculate both prefixes and suffixes with a one index j prefix checks 0 to j and suffix checks j to string length.

Subtask2:
Subtask2 was about the finding the m unique pattern occurrences in a n length string and the time complexity constraint of subtask2 was O((n + m) * log m + total_number_of_characters) in this solution I didn't use hashtable instead I used a hashing function and store the hash values in an array. My hash function was very similar to the my hash function in the hashtable implementation but the size was bigger because test cases will be larger in the subtask2 I used the size as 1e9+9. At first I read the strings than I hashed them all and I initialized a shadow array to keep track of their indexes than using the hashes array and this shadow array I sorted the hashes and kept track of their indexes. After sorting the hashes I used a loop algorithm four time each time starts with min length + i because question prompt indicated that max difference could be five at most. At each traverse I used a rolling hash than binary search the value in my sorted hashes array and if it exists I incremented the count number at it's index using my shadow array. In order to sort and search with my shadow array I used my own sort and search methods. With this approach time constraint was exactly what was mentioned in the question. One possible

approach is in order to prevent the collision there is two hashes each uses different base and modulo and we can compare both in two strings to clarify whether they are equal or not and this doesn't changes the time complexity. There is no worst case in this subtask and also no bestcase to change time complexity significantly.

Subtask3:

In subtask3 the purpose was to find minimum subset of given strings with the shift and reverse operations and also with the minimum reverse operation number. The time constraint limit in this question was O(total_number_of_characters * log n). In this question my hash function was again a polyndromical hash function like the ones I used in subtask 1 and 2. The size I used in module operations was again 1e9+9 in order to avoid from the collisions in larger inputs. Again my attempt starts with independently hashing all strings and put the hashvalues in an array than sort the hash array with a shadow array which accounts for their original indexes in the strings array. After doing this I was starting at index 1 in the hashes array and use the string in this index and shifting it again using a rolling hash algorithm in order to decrease the time complexity. After each shift I make a binary search at the previous part of the array and check whether there is a shifted version exists if one is found I keep the record and they become in a subset if there isn't a shifted version than I look it's reverse it and repeat the process now if one is found again I mark them in a subset and increase the reverse count after that instead of the original hash value I put the reversed hash value. This solves the problem at the given time complexity because I precompute the reversestrings and reverse hashvalues. In order to avoid from collisions again the approach is to check the string's hash values with using two different hash functions and understand the equality or the collision. Again there isn't a worst case in this subtask.