# ADSA Programming Assignment 2

EE19MTECH01008

28 November 2020

## 1 Problem

Consider the 2-SUM problem: given an array of n integers (possibly with repetitions), and a target integer t find if there exist two distinct elements x, y in the array such that x + y = t. There are multiple approaches to find a O(n log n) solution to this problem. We would like to implement a dynamic version of this problem. In this version, you do not know the number of elements in advance. The input arrives as a sequence of operations. We start with the empty multiset S. Operations are of three types:

1. Insert(k) : Inserts a number k into S.
2. Delete(k): Deletes an instance of the key k from S. If k is not present, no change is made to the data structure. If k is present multiple times, any one instance is deleted.
3. Query (a, b): Prints the number of target values in the closed interval [a, b] such that there are distinct elements x, y in the multiset with x + y = t.

Note that by "distinct" above, we mean two distinct elements of the multiset, which could have the same integer value.

## 2 Solution

### 2.1 Algorithm Explanation with time/space complexity:

1. **Insertion into linked list**: A small modification is done in the insert new node function, **sortedInsert()**, which enables the insertion in a sorted manner. The head link is used as an argument since it might get changed if the new value is smaller than the head. This is required for implementing the binary search later.

   Time Complexity: O(n)

   Space Complexity: O(n)

2. **Deletion from linked list**: The **deleteNode()** function is pretty much straight forward. It takes care of null list too.

   Time Complexity: O(n)

   Space Complexity: O(n)

   A function is created to find the length of the linked list which is used to create an array and copy the distinct keys to the array **(arr[])**. While copying, the index of the repeated keys in the linked list is stored in another array **(arr1[])** of the same length.

3. **Query function to find the number of pairs**: The function **aquery()** takes both **arr[]** and **arr1[]** as arguments. The first loop finds the lower bound ( **lower_bound()**) and the upper bound (**upper_bound()**) for a specific target in the array arr[]using binary search. The basic idea is that in x+y=t, where x and y belong to the multiset and t belongs to [a,b], fix x to the initial array element and change y to get corresponding t values. Change the value of x in an iterative manner. So, x=t-y, find the upper and lower bound of (t-y). The time complexity for this loop is:

$$O(n * \sum_{i=1}^{n-1} 2log(n-i)) = O(nlogn)$$

   Here 2 is multiplied because each for upper and lower bound O(log(n-i)) time complexity. n is multiplied since it runs under the loop for traversing through arr[]. This summation is upper bounded by O(nlogn). The space complexity is O(n). The second loop basically takes care of the repeated elements. The **arr[arr1[i]]** returns the position where the element is repeated in the initial array. The element is summed with itself since it is repeated. The value is checked if it lies between the bounds provided by the user. The maximum size of the array arr1[] is n/2 since at max all the elements have duplicates.

   The time complexity for this loop: O(n)

   Space complexity: O($\frac{n}{2}$)

Thus, the overall complexity is given by:
**Time complexity: *O(n log n)***
**Space complexity: *O(n)***

## 2.2 Proof of correctness:

The loop invariant method is used to prove the correctness of the algorithm.

1. **Initiation:** For the condition where there is no or only one element is present in the linked list, the count is always 0 as two minimum elements are required to get a sum. This is maintained in the algorithm.

2. **Maintenance:** Given two numbers as query the algorithm finds the number of pairs from the previously inserted elements in the linked list whose sum lies between the user given range. Even the case of duplicates also considered and is taken care of in the second loop of the aquery() function. The first loop in the function corresponds to finding the upper and lower bound using binary search algorithm from the given linked list.

3. **Termination:** At the last iteration the total count is returned as the answer to the user query. Once the count is returned the program stops.

This proof of correctness is for the **aquery()** method. The functions (**upper_bound()**) and (**lower_bound()**) are basically binary search whose correctness is proved by induction method.

## 2.3 Pseudo-code:

**Algorithm 1** Pseudo code of the solution

```
1:  Read user key (I,D,Q,E)
2:  while (1) do
3:    if key == I then
4:      sortedInsert(head, new node)
5:    else if key == D then
6:      deleteNode(head, node)
7:    else if key == Q then
8:      Enter a,b //bounds [a,b]
9:      Initialize arr[], arr1[] with length of the linked list
10:     Initialize index1=index2=old_index=0 // these are for copying keys to
        array
11:     Copy keys to arr[]
12:     Copy the index of repeated elements in arr1[]
13:     aquery(arr,arr1,index1,index2,a,b)
14:   else if key == E then
15:     exit
16:   end if
17: end while
```