ADSA Assignment -2 Report

(CS6013)

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Submitted By:

Shivangi Parashar

AI20MTECH14012

Course Instructor

Dr. Rakesh Venkat

Contents

1.Dynamic 2-SUM

1.1 System Configurations

- OS:macOS Catalina
- Compiler version:10.15.7

1.2 Problem Description

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.A dyanamic version of this problem needs to be implemented.A user could enter any number of elements so that the number of elements are not known in advance.

1.3Operations Performed

Consider empty multiset S,operations in S:

- Insert(k):Inserts a number k into S
- 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.
- Query(a,b):Prints the number of tarhet values in the closed interval[a,b]such that distinct elements x,y in the multiset with x+y=t

1.4Specifications:

- **I** for insert
- **D** for delete
- **Q** for query
- **E** for program termination

1.5Inputs

- Inputs will be taken from user by standard input
- User should provide the input in valid format(like I for insertion,Q for query,D for delete,E for exit)

1.6 Outputs:

- Outputs will be printed in standard output
- For every Query Q entered by the user in the input stream,immediately a single integer to standard output is printed specifying the answer to that query ,followed by a new line.

2.Implementation Algorithm

2.1Logic used

This solution runs in o(n^2logn) time.

- nlogn to sort the array using merge sort when Q(a,b) is entered by the user,when n elements are there in the sumit will take o(n^2logn) .
- Space taken by merge sort is o(n)
- After the sorting binary search is used to find the pairs such that x+y=t in the given range,which will take o(nlogn) time.

2.2Implementation steps

Insertion in a multiset S

Steps summarization

- Insert operation inserts the element into the multiset S at the end.

Example:S=[1,2,9,5]
Insert(19)
S=[1,2,9,5,19]

Before Insertion:

Index0	Index1	Index2	Index3
1	2	9	5

After Insertion:

Index0	Index1	Index2	Index3	Index4
1	2	9	5	19

2.3Key Points during Insert

1. Input given by the user would be added in the multiset at the end.
2. An intial default capacity of 10 is provided to the table but if the table becomes completely filled and to prevent overflow it's capacity is increased by 1.
3. Dynamic reallocation will be performed and all the elements are copied to the new array to prevent overflow.

2.4Pictorial representation of insert during dynamic allocation

Example insert {1,2,3,4,5,6,7,8,9,10}

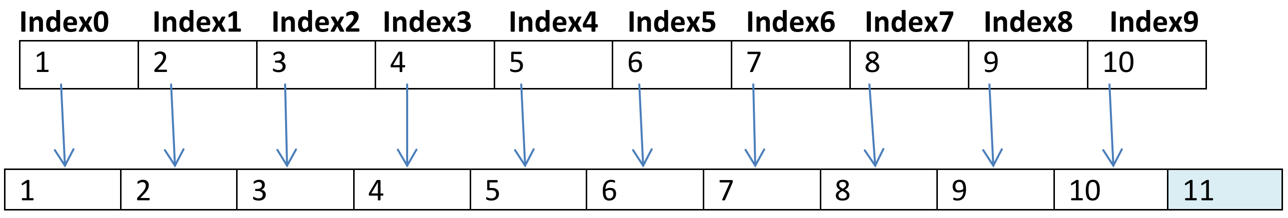
Table with Default capacity(10)

Index0	**Index1**	**Index2**	**Index3**	**Index4**	**Index5**	**Index6**	**Index7**	**Index8**	**Index9**
1	2	3	4	5	6	7	8	9	10

Table Dynamic Reallocation:

To prevent overflow we would dynamically increase the size of array by 1

Insert 11:Since table becomes full this will happen



Elements are copied in newly array and size is increased to allocate 11

2.5Pseudo Code for Insertion

Pseudo code for addition of element

```
Add (Integer e)

//e is the element to be inserted

If tracker==table_size://Table size indicates size of table and tracker keeps the count of number of elements in array

         increaseCapacity();
else:
        element[tracker]=e
        tracker->tracker+1
```

Pseudo Code for increasing capacity

```
If tracker==table_size:

       Table_size->Table_size+1

for i=0 to table_size-1

        increasetable[i]->elements[i]

//increasetable is an array with increased Table_size

elements=increase_table

table_size->table_size+1

//Table size gets increased by 1 to dynamically allocate element
```

2.6 Edge Case Check

Following Conditions are checked which could result in failure in insert operation:

- Default capacity of table is 10,to prevent overflow table size has to be increased so that it could accommodate new element.
- Tracker which contains the number of elements in the list has to be updated.

2.7Time Complexity Analysis

Expected/Amortized/Worst case complexity

Worst Case Time Complexity

- Time complexity for insertion for per operation O(n).It can be observed from pseudo code that each insertion takes O(1) time and at the same time when array gets full we need to copy each element to a new array which will take O(n) time .
- Hence for 1 operation it is O(n) and there are n elements to be inserted at the worst case.Hence,Insertion for n elements could take O(n)^2 time.

Expected Time Complexity

- Expected time complexity for insertion is O(n).Since we will expect in average case atmost n copies operations won't be required which would take every time o(n)to perform copy operation.

Amortized Time Complexity

- **∑Amortized Cost(output) ≥ ∑Actual Cost(output)**

Aggregate Method:

Amortized cost(output)=Total Cost of K operations/Number of operations

- Here total cost of n operations =O(n)^2
- Total operations =O(n)
- Hence Amortized cost=O(n)

2.8 Space Complexity

Space complexity for insertion is O(n).
Copying from one array to other elements+Original array elements=O(n)

2.9 Deletion

- A delete operation deletes the number from the set S if found.

Ex:S=[1,2,3,4,5]
Delete(4)
S=[1,2,3,5]

- If the element is not found it won't do anything.

Ex:S=[1,2,3,4,5]
Delete(9)
S=[1,2,3,4,5]

- If duplicates are present than it would find the index of duplicate occurring first and would delete that particular element.

Ex:S=[1,2,3,3,3]
Delete(3)
Index of first occurring duplicate is 2.Hence it would delete the element from index 2

2.10 Concept used for Deletion

- Deletion of element from array is carried on based on index value.
- If the element to be deleted is found in the list then it's index is returned so that deletion of element could be done for that particular index.

Pictorial Representation

Index	0	1	2	3	4	5
Set,S	1	3	3	9	8	4

Delete 3,8,4

Key	**Index Returned**
1	0
3	1
3	2
9	3
8	4
4	5

After Deletion of 3,8,4 and after shifting elements table is:

Index	**Set,S**
0	1
1	3
2	9

2.11Algorithm for Delete

- Element to be deleted is passed as argument from which index is calculated and based on index element would be deleted from that particular position
- If the current size of the array is 0 than no deletion is performed since atleast one element has to be present in the array to get deleted.
- Elements shifting would be carried out after deletion takes place.

  1. If the key to be deleted is present at index 0,then overall shifting of all n elements by 1 position need to be done .Total n-1 shifts to be done.Hence, Time complexity T(n)=O(n)
  2. If the key to be deleted is present at index(n-1),then just delete the element at the last index and there would be no need of shifting to be carried out.Since this won't require any shiftings and rearrangements .Hence Time complexity would be T(n)= O(1).

- If Duplicate elements are present then first matched index is returned.All the elements from that index will be shifted left by 1 position.Hence this takes T(n)=O(n).

Pictorial Representation

Index0	Index1	Index2	Index3	Index4	Index5
1	2	3	4	5	6

If element to be deleted is present at last index then just delete no shifting required

Index0	Index1	Index2	Index3	Index4
1	2	3	4	5

If the element to be present at some index:Ex:2

Index0	Index1	Index2	Index3	Index4
1	2	3	4	5

Shift from Index 3 to Index2 Shift from Index 4 to Index 3

Hence table becomes:

Index0	Index1	Index2	Index3
1	2	4	5

2.12Pseudo Code for deletion:

Pseudo code for Index :

```
removeElement(element)//element is the element to be deleted
        for i=o to elements.length
                   if elements[i]=element
                            index= i//Here I is index of element

                   if index==elements.length-1:
                           removeAtEnd();
                  else:
                            remove(index);
```

Pseudo code for Deletion:

```
remove(int ind)
if ind!=elements.length-1 and ind>0:
        for i=index to tracker-1//tracker keeps count of number of elements present in array
             elements[i]=elements[i+1]//shift all elements by 1 position
 elements[tracker-1]=0
  return tracker->tracker-1


removeAtEnd()
        if tracker>0:
            elements[tracker-1]=0//If element to be removed is last then no shifting required
        return tracker--
```

2.13Time Complexity for deletion:

Worst case time complexity of Deletion:

If the key to be deleted is present at index 0,then overall shifting of all n elements by 1 position need to be done .Total n-1 shifts to be done.Hence Time complexity for 1 operation is T(n)=O(n)

Expected time Complexity of Deletion:

If the key to be deleted is present at index(n-1),then just delete the element at the last index and there would be no need of shifting to be carried out.Since this won't require any shiftings and rearrangements .Hence Time complexity would be T(n)= O(1).
If the key to be deleted is present at index 0,then overall shifting of all n elements by 1 position need to be done .Total n-1 shifts to be done.Hence Time complexity per operation T(n)=O(n)
Hence Expected complexity would be O(n)

Amortized Time Complexity for Deletion:

Amortized cost for delete for 1 operation is O(1) and for n operations it is O(n)

- **∑Amortized Cost(output) ≥ ∑Actual Cost(output)**

Aggregate Method:

Amortized cost(output)=Total Cost of K operations/Number of operations

- Here total cost of n operations =O(n)^2
- Total operations =O(n)
- Hence Amortized cost=O(n)

2.14Query Operation:

The query operation takes place as:

For every query in the input stream ,you should immediately print a single integer specifying answer to the query followed by a new line.

Input from user:

- The input is taken from the user should be a string separated with space.
- String Tokenizer is used for getting tokens it allows us to break a string into tokens.
- String nextToken() is used which returns the next token from the String Tokenizer object.

Note:Since there can be duplicates present ,every duplicates need to be considered only once while adding with another element of array.

2.15Concept Used:

- When user enters Q(a,b) then it would first sort the list using merge sort which has a T(n)=O(nlogn) time complexity.
- After sorting the list it would search for the pairs such that “x+y=t” using binary search which would have T(n)=O(logn).It would count all the targets in that particular range and would output the same in standard output.

Pictorial Representation:

Set S after insert operation

Index	0	1	2	3
Set,S	10	3	5	6

Q(10,13).Once this is entered by the user first sorting would be performed in O(nlogn)time.

Index	0	1	2	3
Set,S	3	5	6	10

After sorting using binary search pairs are counted in O(logn) time.(x+y=10,11,12,13)

Index	0	1	2	3
Set,S	3	5	6	10

Output=2;6+5=11 and 10+3=13

2.16 Counting Targets

- To count targets we would use Binary search since array is sorted using binary search.
- We would count the pairs that equals to the target in the range x to y.

2.17Pseudo Code to count targets:

```
twoSumCheck(String a,String b,MyArrayList da)
    sort da//using mergesort
    for i=x to y:
      while(l<r)
          if(sortedArray[l]+sortedArray[r]==i)//if target found increment count
              count->count+1
          elseif(sortedArray[l]+sortedArray[r]<i)If sum is left than target increment left pointer
              l->l+1
         else
              r->r-1
  return count
```

Note:If the size of the array is less than 2 ,then 0 will be displayed in the standard output.

2.18 Time Complexity for Query

Worst case time complexity:

For every query operation we would sort our list and then perform sum .Hence for 1 operation sort would take O(nlogn) but there are n such operations .Hence T(n) =o(n)^2logn.

For counting the number of targets in the range x and y there we are using Binary search which would require o(logn) and this could at the worst case occur n time .T(n)=O(nlogn).Hence overall worst case complexity would be T(n)= 0(n)^2logn

Expected Time complexity:T(n)=O(nlogn) considering worst case would not occur always.

Amortized Time Complexity using aggregate analysis:T(n)=O(n)^2logn/n=O(n)

2.19Space Complexity:

Space complexity for merge sort=O(n) and for rest allocations it is constant.Hence,space complexity=O(n)

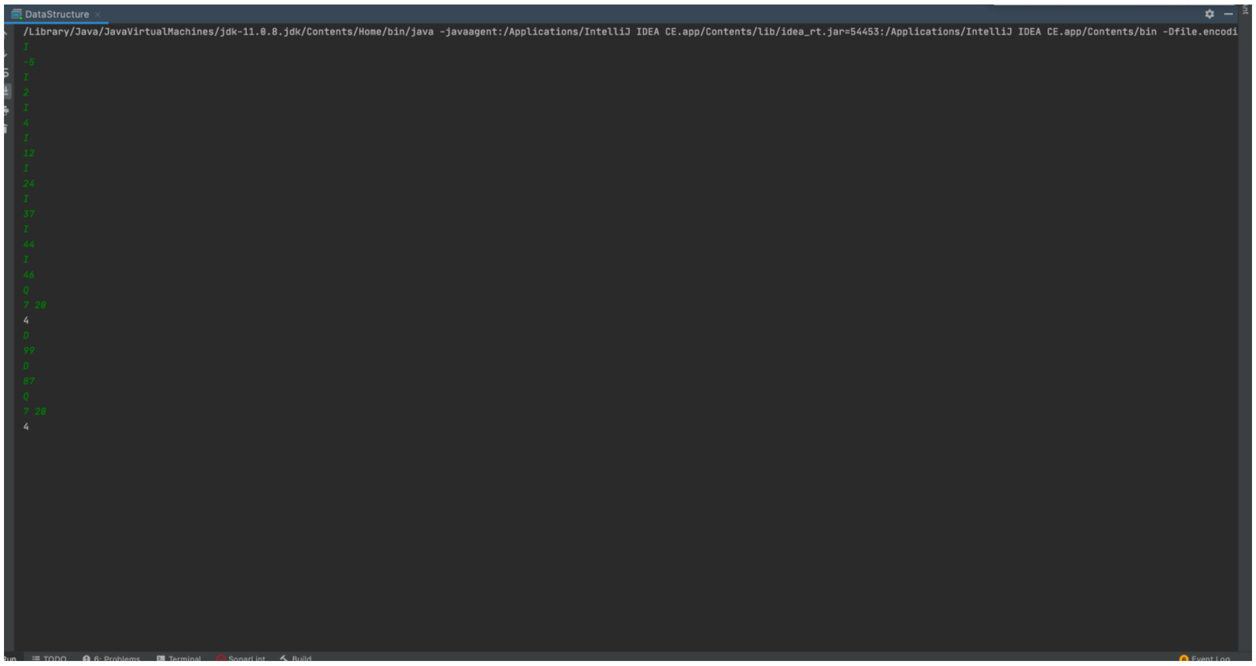
2.19 Exit Operation

E will be entered as input by the user from the standard input terminal to terminate the program.

Language used:

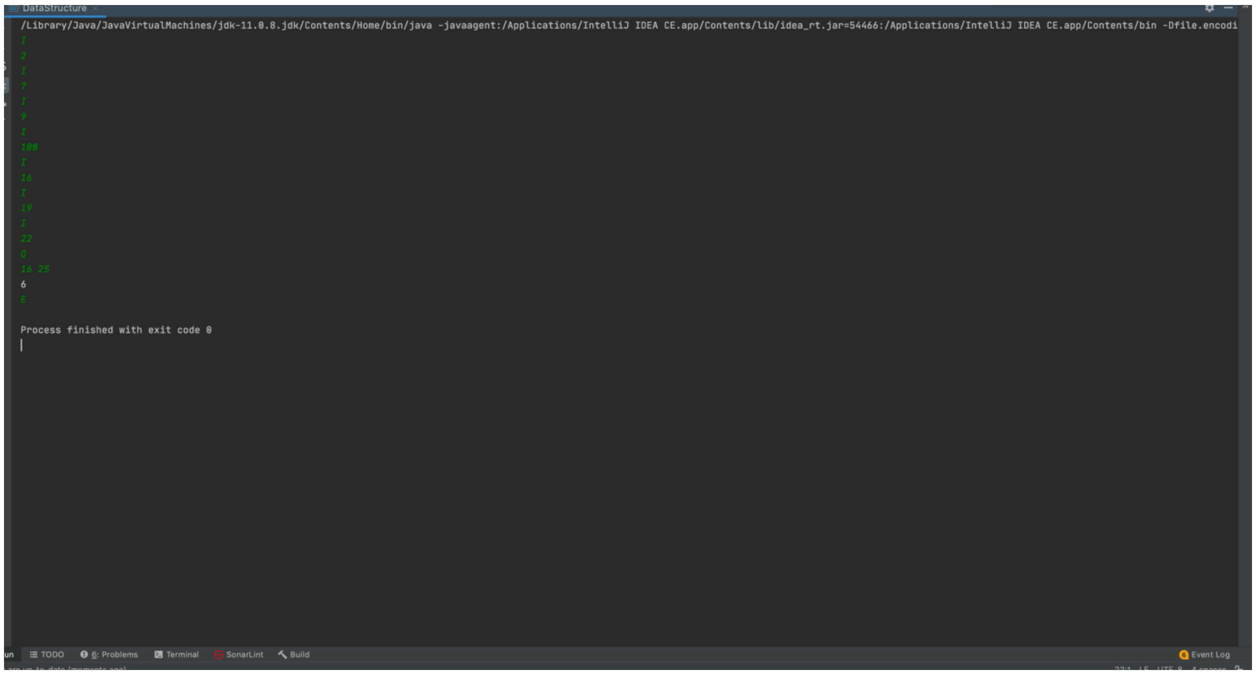
Java is used as a programming language .

Testcases/Screenshots



DataStructure

```
/Library/Java/JavaVirtualMachines/jdk-11.0.8.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=54453:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encodi
I
-5
I
2
I
4
I
12
I
24
I
37
I
44
I
46
Q
7 20
4
D
99
D
87
Q
7 20
4
```



DataStructure

```
/Library/Java/JavaVirtualMachines/jdk-11.0.8.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar=54466:/Applications/IntelliJ IDEA CE.app/Contents/bin -Dfile.encodi
I
2
I
7
I
9
I
100
I
16
I
19
I
22
Q
16 25
6
E

Process finished with exit code 0
```