Algorithms

Assignment #1 (Deadline 24/12/2021)

Guidelines:

- Solve 10 problems extra consider as bonus
- This assignment is group of 4
- Please submit your solutions via blackboard
- You should submit a zip file (name this file with your ids A1_ID1_ID2_ID3_ID4.zip), put all files (codes) for each problem (name code file p1 , p2 , .. etc)

Problem#1 - Insertion Sort

Insertion Sort is a classical sorting technique.
As you probably already know, the algorithm can be really slow. To study this more, you want to find out the number of times the swap operation is performed when sorting an array.

Sample 1 - Input:
1 1 1 2 2

Sample 1 - Output:
0

Sample 2 - Input:
2 1 3 1 2

Sample 2 - Output:
4

Problem#2 - Quick Sort

Given an array A of integers and an integer k, you're to get the kth minimum integer in this array.

Since the array size maybe large, the array is not given as input. You've to generate it yourself. To generate the array elements, consider this function that generates a random number:

```
long long m_w, m_z;
int get_random()
{
m_z = 36969 * (m_z & 65535) + (m_z >> 16);
m_w = 18000 * (m_w & 65535) + (m_w >> 16);
long long res = (m_z << 16) + m_w;
return res % 1000000000;
}
```

You'll be given the initial values of m_w and m_z, and the array size N. Then you've to call this function N times to generate the N array elements.

Sample 1 - Input:
3 1 1 2

Sample 1 - Output:
643275087

Sample 2 - Input:
3 2 1 2

Sample 2 - Output:
692891904

Problem#3 - Count Inversions (Merge Sort)

Given an array A of integers, you're to calculate the number of inversions in this array. Number of inversions is the number of pairs (i,j) of array indices with i<j and A[i] > A[j] - (more info. https://www.youtube.com/watch?v=EJwR1FG8vGk)

Sample 1 - Input:
1 2 3 4

Sample 1 - Output:
0

Sample 2 - Input:
3 2 1 4

Sample 2 - Output:
3

Problem#4 - Clear the Multiset (Divide and conquer)

You have a multiset containing several integers. Initially, it contains a1 elements equal to 1, a2 elements equal to 2, ..., an elements equal to n.
You may apply two types of operations:

- choose two integers l and r ($l \le r$), then remove one occurrence of l, one occurrence of l+1, ..., one occurrence of r from the multiset. This operation can be applied only if each number from l to r occurs at least once in the multiset
- choose two integers i and x ($x \ge 1$), then remove x occurrences of i from the multiset. This operation can be applied only if the multiset contains at least x occurrences of i

What is the minimum number of operations required to delete all elements from the multiset?

Sample 1 - Input:
1 4 1 1

Sample 1 - Output:
2

Sample 2 - Input:
1 0 1 0 1

Sample 2 - Output:
3

Explanation of sample 1
1 4 1 1
a1 = number of occurrence for number 1 is 1
a2 = number of occurrence for number 2 is 4
a3 = number of occurrence for number 3 is 1
a4 = number of occurrence for number 4 is 1

This multiset equivalent to {1, 2,2,2,2, 3, 4}
solution :

- Use first operation from (l = 1 to r = 4) result: {2,2,2}
- Use second operation (i = 2 and x= 3) result: {}

Problem#5 - SkipList

The worst case search time for a sorted linked list is O(n) as we can only linearly traverse the list and cannot skip nodes while searching. For a Balanced Binary Search Tree, we skip almost half of the nodes after one comparison with root. For a sorted array, we have random access and we can apply Binary Search on arrays.

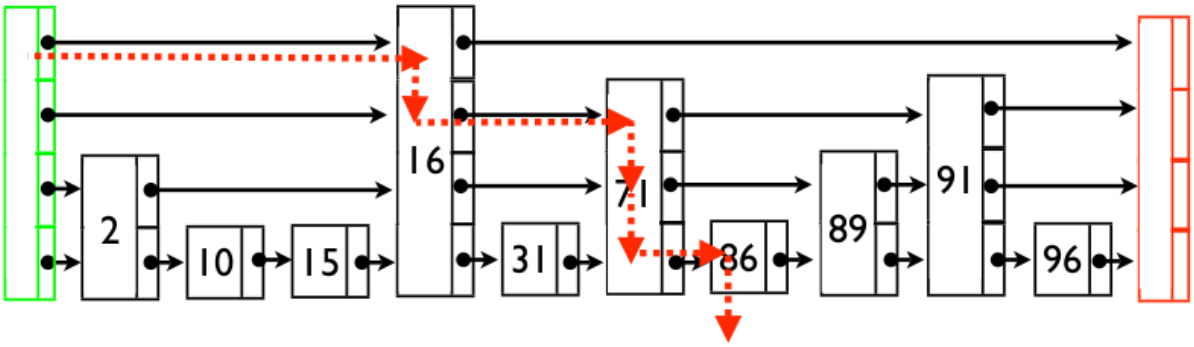
Can we augment sorted linked lists to make the search faster? The answer is Skip List. The idea is simple, we create multiple layers so that we can skip some nodes.

Implement randomized SkipList insert, delete and search functions then write some testcase to test these functionalities

Create SkipList class with the following features:

- Insert an element: skiplist.insert(num)
- Search for an element and print the number of search steps if exists otherwise print -1
    - skipList.search(86) -> 3
    - skipList.search(16) -> 1
    - skipList.search(160) -> -1
- Print number of layers
    - skipList.getlayers() -> 4
- Print the numbers of certain layer ex: printLayer(int layerNum)
    - printLayer(1)
        - 2, 16 ,71, 89, 91
    - printLayer(4)
        - 16
    - printLayer(0)

- 2,10,15,16,31,71,89,91,96



Problem#6 - Hashing

Similar idea to hash joins, Assume we have 2 tables Employees(ID, Name, Department) and Transactions(Sold Product, Sold By, Sold To). “Sold By” in Transactions is a foreign key to “ID” in Employees. We would like to join the 2 tables and calculate the joins. Knowing that a row in Employees can at most have 1 mapped row in Transactions. Propose a better way to store the table records and calculate the joins.

	ID	Name	Department
Row 1			
Row 2			
Row 3			

	Sold Product	Sold By	Sold To
Row 1			
Row 2			
...			
...			
Row 34349			

Problem#7 - Matrix Multiplication - Strassen

In Computer Graphics transformations are applied on many many vertices on the screen. Translation, Rotations and Scaling.

Assume you're operating on a vertex with 3 values (X, Y, 1). X, Y being the X Y coordinates and 1 is always constant

A Translation is done on X as X = X + X' and on Y as Y = Y + Y'
A scaling is done on X as X = aX and on Y as Y = bY

Propose the best way to store these linear equations and an optimal way to calculate them on each vertex

Problem#8 - Binary Search

A toy factory found a defect in one of it's equipment resulting some toys be defective. The problem however is that they don't know which series of toys has that defect. Knowing that all toys are produced in order with a special serial propose a way to quickly find the serial numbers of the defective toys.

// Red black tree problems.

1) In this problem you are given two type of query

    1. Insert an integer to the list.
    2. Given an integer **x**, you're about to find an integer **k** which represent x's index if the list is sorted in ascending order. Note that in this problem we will use 1-based indexing.

As the problem title suggest, this problem intended to be solved using $O(\log_2(n))$ for both insertion and getting X's index

Input

The first line contains an integer Q, which denotes how many queries that follows. The next Q lines will be one of the type queries which follow this format:
1 x means insert x to the list
2 x means find x's index if the list is sorted in ascending order.

Output

For each query type 2, print a line containing an integer as the answer or print "-1" no quotes if the requested number does not exist in the current list

Input Example:

10

1 100

1 74

2 100

2 70

1 152

1 21

1 33

2 100

2 21

2 1

Output:

2 -1 4 1 -1

Problem#9 - (Extended of problem 10)

We extended previous Problem by adding extra queries:
This problem is simple. Initially, there is a list and it's empty. Then you are given four types of query.

1. Insert data to the list
2. Remove data from the list
3. Print an index (1-based) from a specified data after the list was sorted ascendingly
4. Print data from a specified index (1-based) after the list was sorted ascendingly

We want to make insertion, deletion, getting operations in $O(\log_2(n))$

Input

Input contains several lines. Each line follows one of these formats.

1 n: Insert **n** ($0 <= n <= 2^{31} - 1$) to the list

2 n: Remove **n** from the list. If **n** was not found, do nothing

3 n: Print **n**'s index (1-based) after the list was sorted ascendingly

4 i: Print data on **i**-th index (1-based) after the list was sorted ascendingly ($0 <= i <= 2^{31} - 1$)

-1: End of input

Output

For each query 3, print **n**'s index in one line. If **n** was not found, just print -1

For each query 4, print data on **i**-th index in one line. If the index is not valid, just print -1

Example:
Input:
1 3
1 5
3 6
3 5
1 2
4 2
2 2
4 2
4 5
-1
Output:
-1
2
2
3
-1

Problem#10 - Maximum Sum of Array

Given an array of integers, find the subset of non-adjacent elements with the maximum sum. Calculate the sum of that subset. It is possible that the maximum sum is 0 , the case when all elements are negative.

Example

1) arr=[-2, 1, 3, -4, 5]
   The following subsets with more than 1 element exist. These exclude the empty subset and single element subsets which are also valid.

Subset	Sum
[-2, 3, 5]	6
[-2, 3]	1
[-2, -4]	-6
[-2, 5]	3
[1, -4]	-3
[1, 5]	6
[3, 5]	8

   The maximum subset sum is 8 . Note that any individual element is a subset as well.

2) arr = [-2, -3, -1]
   In this case, it is best to choose no element: return 0.

Sample 1 - Input:
3 7 4 6 5

Sample 1 - Output:
13

Sample 2 - Input:
3 5 -7 8 10

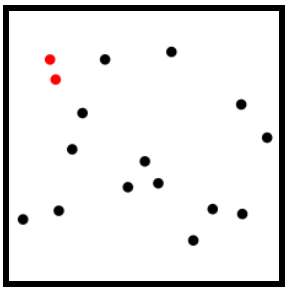
Sample 2 - Output:
15

Explanation sample 1

Our possible subsets are $[3,4,5], [3,4], [3,6], [3,5], [7,6], [7,5]$ and $[4,5]$ and $[4,5]$. The largest subset sum is 13 from subset [7,6]

Problem#11 - The Closest Pair Problem (Divide and conquer)

We are given an array of n points in the plane, and the problem is to find out the closest pair of points in the array. This problem arises in a number of applications. For example, in air-traffic control, you may want to monitor planes that come too close together, since this may indicate a possible collision. Recall the following formula for distance between two points p and q.
Euclidean distance d(p , q) = sqrt[(qx − px)^2 + (qy − py)^2]
The Brute force solution is O(n^2), compute the distance between each pair and return the smallest. We can calculate the smallest distance in O(nLogn) time using Divide and Conquer strategy.



Problem#12 - Optimal Array Multiplication Sequence (Divide and conquer)

Given two arrays *A* and *B*, we can determine the array *C* = *A B* using the standard definition of matrix multiplication:

The number of columns in the *A* array must be the same as the number of rows in the *B* array. Notationally, let's say that *rows*(*A*) and *columns*(*A*) are the number of rows and columns, respectively, in the *A* array. The number of individual multiplications required to compute the entire *C* array (which will have the same number of rows as *A* and the same number of columns as *B*) is then *rows*(*A*) *columns*(*B*) *columns*(*A*). For example, if *A*is a array, and *B* is a array, it will take , or 3000 multiplications to compute the *C* array.

To perform multiplication of more than two arrays we have a choice of how to proceed. For example, if *X*, *Y*, and *Z* are arrays, then to compute *X Y Z* we could either compute (*X Y*) *Z* or *X* (*Y Z*). Suppose *X* is a array, *Y* is a array, and *Z* is a array. Let's look at the number of multiplications required to compute the product using the two different sequences:

(*X Y*) *Z*

- multiplications to determine the product (*X Y*), a array.
- Then multiplications to determine the final result.
- Total multiplications: 4500.

X (*Y Z*)

- multiplications to determine the product (*Y Z*), a array.
- Then multiplications to determine the final result.
- Total multiplications: 8750.

Clearly we'll be able to compute (*X Y*) *Z* using fewer individual multiplications.

Given the size of each array in a sequence of arrays to be multiplied, you are to determine an optimal computational sequence. Optimality, for this problem, is relative to the number of individual multiplications required.

Sample 1 - Input:
1 5
5 20
20 1

Sample 1 - Output:
(A1 x (A2 x A3))

Sample 2 - Input:
30 35
35 15
15 5
5 10
10 20
20 25

Sample 2 - Output:
((A1 x (A2 x A3)) x ((A4 x A5) x A6))

Problem#13 - Find a pair with the given sum in an array (Hashing)

Given an unsorted integer array, find a pair with the given sum in it. We can use hashing to solve this problem in linear time.
For example,

Input:

nums = [8, 7, 2, 5, 3, 1]
target = 10

Output:

Pair found (8, 2) Or Pair found (7, 3)

Input:

nums = [5, 2, 6, 8, 1, 9]
target = 12

Output: Pair not found

Problem#14 - Find the minimum index of a repeating element in an array (Hashing)

Given an integer array, find the minimum index of a repeating element in linear time and doing just a single traversal of the array. We can use hashing to solve this problem in linear time.
For example,

Input: { 5, 6, 3, 4, 3, 6, 4 }

Output: The minimum index of the repeating element is 1

Input: { 1, 2, 3, 4, 5, 6 }

Output: Invalid Input

Problem#15 - Group elements of an array based on their first occurrence (Hashing)

Given an unsorted integer array containing many duplicate elements, rearrange it such that the same element appears together and the relative order of the first occurrence of each element remains unchanged. For example,

Input: { 1, 2, 3, 1, 2, 1 }

Output: { 1, 1, 1, 2, 2, 3 }

Input: { 5, 4, 5, 5, 3, 1, 2, 2, 4 }

Output: { 5, 5, 5, 4, 4, 3, 1, 2, 2 }