LAB-PRAC-14_SORT-MISC

Predecessor and Successor (p1v1d1)

Predecessor and Successor [20 marks]

Problem Statement
The first line of the input will give you two integers n and q, separated by a space. n will be strictly positive but q may be negative or zero or positive. The second line will give you a list of n integers, separated by a space. The integers will be given in non-decreasing order.

In your output, you have to print, in two separate lines, the largest number in the list that is strictly smaller than q (if such a number does not exist, print -1), and the smallest number in the list that is strictly larger than q (if such a number does not exist, print -1).

Caution

1. Pay close attention to the problem statement, especially on the significance of the word "strictly".
2. The numbers in the list may repeat and the number q may itself appear in the list one or more times.
3. Try to use the binary search method to avoid searching all the elements of the list. Try to search for the two numbers you have to print, in O(log n) time rather than O(n) time.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE 1:
INPUT
4 6
3 4 9 11

OUTPUT:
4
9

Explanation: The query number is 6, the largest number smaller than 6 in the array is 4, while the smallest number greater than 6 in the array is 9.

EXAMPLE 2:
INPUT
5 12
3 4 9 11 12

OUTPUT:
11
-1

Explanation: There are no numbers strictly greater than 12 in the list. 11 is the largest of the numbers strictly smaller than 12.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
10 3 -20 -5 3 5 7 8 9 11 14 20	-5 5
5 -1 3 6 9 9 12	-1 3
10 7 -4 -4 -4 1 3 7 7 7 11 15	3 11
6 6 6 6 8 8 10 10	-1 8
6 10 6 6 8 8 10 10	8 -1
6 40 40 40 40 40 40 40	-1 -1

Insertion Sort (p1v2d1)

Insertion Sort [20 marks]

Problem Statement
There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One popular algorithm for sorting small to medium arrays is called Insertion Sort.

Insertion sort maintains the following invariant - at the end of the i-th iteration of the algorithm, the first i positions of the array must be sorted in non-decreasing order. If there are n elements in the array, then the algorithm runs for n iterations. Say after i iterations, the first i elements are sorted in non-decreasing order. Then in the (i+1)-th iteration, the element currently at index i in the array (i.e. the (i+1)-th element from the left in the array) is *inserted* into its appropriate position with respect to the first i elements, thus creating an array of (i+1) elements sorted in non-decreasing order and also fulfilling the promise of the invariant..

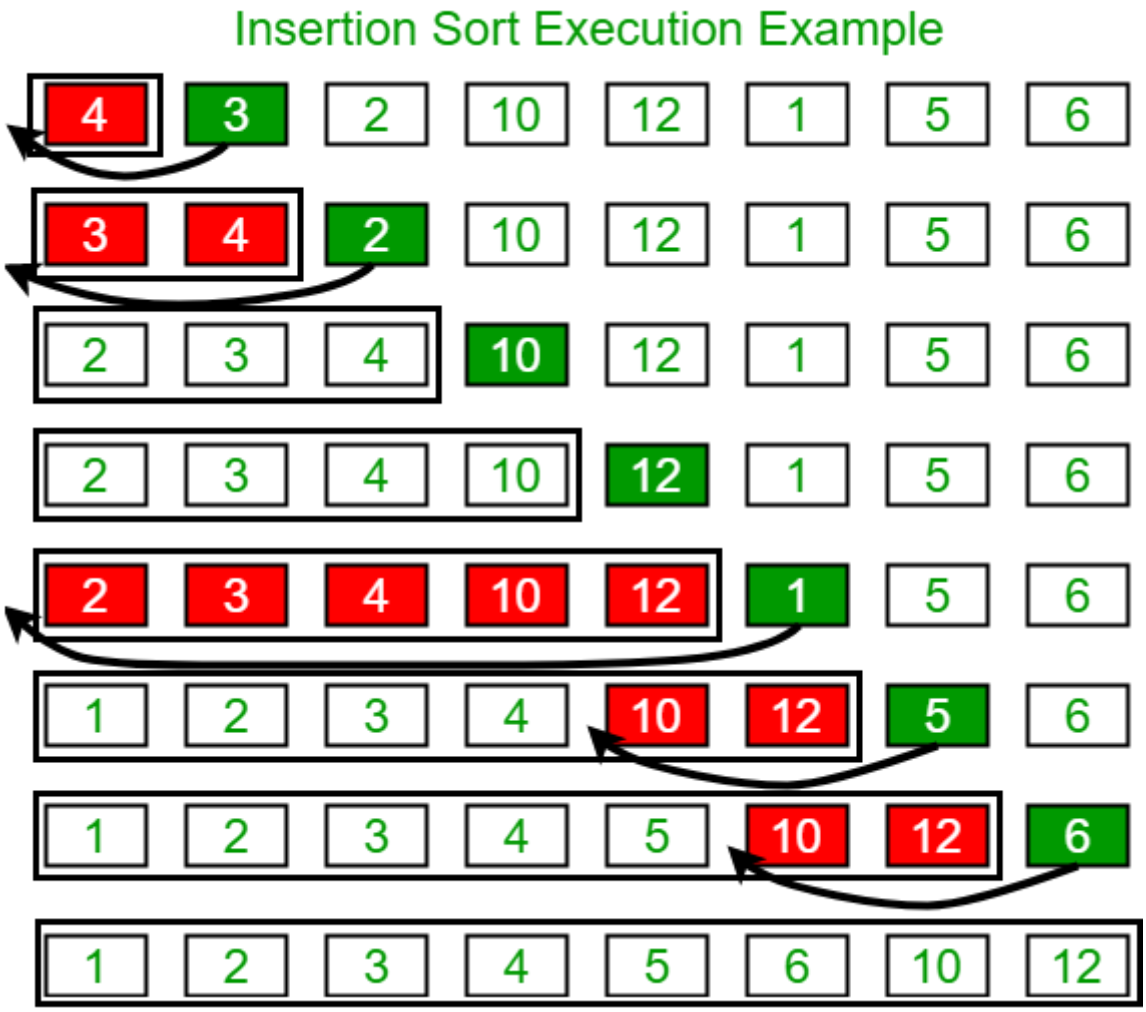
This is where the algorithm gets its name. At the i-th iteration, the i-th element of the array is inserted into its correct location with respect to the previous (i-1) elements to create a sorted array of i elements. The first line of the input will give you n, a strictly positive integer and the second line will give you n integers, separated by a space. Store these numbers in an array of size n.

In your output, you have to print the array after each iteration of insertion sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space.

However, there should be no space at the end of a line.

The image below shows you how insertion sort works. Notice that at each step, a larger and larger set of elements (indicated by the black box) gets sorted in non-decreasing order. Also notice that at some steps, nothing needs to be done but in other steps, the element just outside the box needs to be inserted in the correct position to expand the sorted box.

Image courtesy: geeksforgeeks.org (image modified for better clarity)



Caution

1. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order insertion sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
2. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
3. The number of elements n can be any strictly positive number. Your output must have exactly n lines.
4. As you can imagine, the first iteration of the algorithm has to do nothing since it only has to ensure that the array of the first 1 elements is sorted. However, a single element is always sorted so nothing needs to be done. However, you must still print the array after the first iteration.
5. Be careful about extra/missing lines and extra/missing spaces in your output. There should be no space at the end of any line in your output, nor should there be any extra newlines at the end of your output.

EXAMPLE:
INPUT

4
3 5 4 2

OUTPUT:
3 5 4 2
3 5 4 2
3 4 5 2
2 3 4 5

Explanation: At the end of iteration 1, the first 1 element of the array is correctly sorted. Since, it is a single element, it is sorted by default. At the end of the 2nd iteration, the first two elements are sorted. Since they were sorted already, nothing had to be done. After the third iteration, the first 3 elements must have to be in sorted order. However, this requires 4 to be inserted between 3 and 5 since that is its correct place in the sorted order of the first 3 elements in the array. In the final step, all the elements must be sorted, and hence, 2 is placed before 3.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 7 3 2 1 -100	7 3 2 1 -100 3 7 2 1 -100 2 3 7 1 -100 1 2 3 7 -100 -100 1 2 3 7
6 199 21 39 5 37 199	199 21 39 5 37 199 21 199 39 5 37 199 21 39 199 5 37 199 5 21 39 199 37 199 5 21 37 39 199 199 5 21 37 39 199 199
5 4 3 4 4 3	4 3 4 4 3 3 4 4 4 3 3 4 4 4 3 3 4 4 4 3 3 3 4 4 4
10 2 3 4 -2 -2 -2 1 1 1 11	2 3 4 -2 -2 -2 1 1 1 11 2 3 4 -2 -2 -2 1 1 1 11 2 3 4 -2 -2 -2 1 1 1 11 -2 2 3 4 -2 -2 1 1 1 11 -2 -2 2 3 4 -2 1 1 1 11 -2 -2 -2 2 3 4 1 1 1 11 -2 -2 -2 1 2 3 4 1 1 11 -2 -2 -2 1 1 2 3 4 1 11

	-2 -2 -2 1 1 1 2 3 4 11 -2 -2 -2 1 1 1 2 3 4 11
8 9 6 8 9 6 8 9 7	9 6 8 9 6 8 9 7 6 9 8 9 6 8 9 7 6 8 9 9 6 8 9 7 6 8 9 9 6 8 9 7 6 6 8 9 9 8 9 7 6 6 8 8 9 9 9 7 6 6 8 8 9 9 9 7 6 6 7 8 8 9 9 9
1 -1	-1

Link a List (p1v3d1)

Link a List [20 marks]

Problem Statement
The first line of the input will give you n, a strictly positive integer. The next line will give you n integers, separated by a space. These are the occupants of a linked list we are going to construct. However, the linked list does not contain these elements in this order. Store these n integers in an array arr.

After this, in the next line, we will give you the index of the head of the linked list in the array arr. After this, there will be n-1 more lines which will contain pairs of numbers of the form a b (i.e. the two numbers will be separated by a space). This will indicate that the element at index a in the array arr points to the element at index b in the array arr.

Create a linked list with n nodes with the elements in the given order. Please refer to the hints to see an easy way to do so. In your output, in the first line, print the linked list by printing elements from the head to tail, two elements separated by a single space. There should be no spaces at the end of the line. Then reverse the linked list i.e. all links are reversed and the old tail becomes the new head and the old head becomes the new tail. Print the reverse linked list on the second line of the output.

Caution

1. n may be any strictly positive integer, even 1 in which the linked list will have a single element.
2. There are two lines of output. There should be no extra spaces at the end of any of the lines.

HINTS: Following these steps might make your life easier:

1. Create a structure node to store an integer and a pointer.
2. Make an array of n nodes to store the n integers as given as well as the pointers to the next element.
3. Store the location of the head node
4. For each pair (a b) given as input, make the node at index a in the array point to the node at index b in the array
5. Write a function to print a linked list.
6. Write a function to reverse the linked list.

Note that we are asking you to use a static data structure like an array, to store a linked list, only to make this

problem less complicated. What you have created is not a very efficient dynamic data structure.

EXAMPLE:
INPUT
5
1 2 3 4 5
2
1 0
3 4
0 3
2 1

OUTPUT:
3 2 1 4 5
5 4 1 2 3

Explanation: The head is index 2 in the array i.e. the element 3. The links in the linked list are shown below
2 => 1
4 => 5
1 => 4
3 => 2

Thus, the linked list is 3 => 2 => 1 => 4 => 5 with 5 as the tail. The reversed list is
5 => 4 => 1 => 2 => 3 with 5 as the head and 3 as the tail.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 1 2 3 4 5 2 1 0 3 4 0 3 2 1	3 2 1 4 5 5 4 1 2 3
4 23 39 5 37 0 0 1 1 2 2 3	23 39 5 37 37 5 39 23

1 42 0	42 42
8 2 2 2 2 2 2 2 2 2 0 1 5 6 4 5 2 0 3 4 6 7 1 3	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
7 1 2 3 4 5 6 7 6 3 2 5 4 4 3 2 1 1 0 6 5	7 6 5 4 3 2 1 1 2 3 4 5 6 7
2 4 3 1 1 0	3 4 4 3

The United Sums of Arrays (p2v1d1)

The United Sums of Arrays [20 marks]

Problem Statement
In the first line of the input we will give you three integers m, n, q. m and n will be strictly positive and will denote the size of two arrays A and B. q will be an integer that may be negative or positive or zero. In the next line we will give you m integers that constitute the array A. In the next line we will give you n integers that constitute the array B. Both arrays will be sorted in non-decreasing order.

You have to tell us, for every element of the array A[i], how many elements of B exist, such that A[i] + B[j] = q. Your output should contain m lines, each line telling us, for one value of i, how many elements of B exist which, when added to A[i], give q. There should be no spaces in your output.

Caution

1. Try to use the binary search method to avoid searching all pairs elements from the two arrays. Try to search for a matching element in array B corresponding to every element in array A. Your solution should work in O(m * log n) time rather than O(m * n) time.
2. Integers may repeat in the arrays A and B. The arrays may contain negative integers or zero values as well.
3. The lengths of the arrays, i.e. m and n may be any strictly positive integer, even 1.
4. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
4 4 12
1 4 4 6
6 8 8 12

OUTPUT:
0
2
2
1

Explanation: A[0] = 1 does not add with anything in array B to give 12. A[1] = 4 adds with B[1] = B[2] = 8 to give 12. A[2] = 4 also adds with B[1] = B[2] = 8 to give 12. A[3] = 6 adds with B[0] = 6 to give 12.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
4 4 12 1 4 4 6 6 8 8 12	0 2 2 1
2 4 14 11 12 2 2 3 3	2 2
4 6 10 5 5 5 5 5 5 5 5 5 5	6 6 6 6
5 5 -11 -8 -8 -5 -5 -5 -6 -6 -6 -3 -3	2 2 3 3 3
1 1 0 -1 1	1
4 1 0 3 3 3 4	1 1

-3	1 0

Bubble Sort (p2v2d1)

Bubble Sort [20 marks]

Problem Statement
There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One simple algorithm is called Bubble Sort. As the name suggests, bubble sort causes elements to "bubble" up to their correct position in the array.

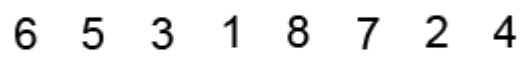
Bubble Sort maintains the following invariant - after the i-th iteration of the algorithm is over, the i-th largest element in the array must reach its correct location in the array. Thus, if the array has n elements in total, the i-th largest element would be found in the index n - i i.e. the largest element would be found at index n-1 after iteration 1, the 2nd largest element would be found in index n - 2 after iteration 2 and so on. Thus, if there are n elements in the array, bubble sort would run for n-1 iterations since after the (n-1)-th iteration, all elements would be in their correct position.

The way bubble sort accomplishes the above is actually very cute - it goes from left to right, comparing adjacent elements. If it finds an element that is strictly larger than the element to its immediate right (i.e. the elements are out of order), it swaps the two elements. Verify that if we do this on an array from left to right, we would end up transporting the largest element of the array to the last position in the array. The first line of the input will give you n, a strictly positive integer greater than 1 (i.e. n can be 2 or larger) and the second line will give you n integers, separated by a space. Store these numbers in an array of size n.

In your output, you have to print the array after each iteration of bubble sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space. However, there should be no space at the end of a line.

The animation below shows you how bubble sort works. Notice that at step i, the i-th largest element reaches its correct location in the array.

Image courtesy: wikipedia.org



Caution

1. Bubble sort goes over the array from left to right, starting from the index 0, comparing adjacent locations and swapping them if they are out of order, in every iteration. However, you should be able to deduce that it does not need to go all the way till the last index in later iterations.

2. There may be iterations where nothing needs to be done. However you still have to print the array after that iteration. Your output must contain n-1 lines.
3. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order bubble sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
4. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
5. The number of elements n can be any strictly positive number greater than 1. Your output must have exactly n-1 lines.
6. Be careful about extra/missing lines and extra/missing spaces in your output. There should be no space at the end of any line in your output, nor should there be any extra newlines at the end of your output.

EXAMPLE:
INPUT
4
3 5 4 2

OUTPUT:
3 4 2 5
3 2 4 5
2 3 4 5

Explanation: Initial state of the array is 3 5 4 2
Iteration 1:

1. 3 and 5 get compared. They are in correct order so no change
2. 5 and 4 get compared. They are out of order so they get swapped. 3 4 5 2
3. 5 and 2 get compared. They are out of order so they get swapped. 3 4 2 5

State of array after iteration 1: 3 4 2 5
Iteration 2:

1. 3 and 4 get compared. They are in correct order so no change
2. 4 and 2 get compared. They are out of order so they get swapped. 3 2 4 5
3. 4 and 5 get compared. They are in correct order so no change

State of array after iteration 2: 3 2 4 5
Iteration 3:

1. 3 and 2 get compared. They are out of order so they get swapped. 2 3 4 5
2. 3 and 4 get compared. They are in correct order so no change
3. 4 and 5 get compared. They are in correct order so no change

State of array after iteration 2: 2 3 4 5

Notice that after iteration 1, 5 (the largest element in the array) reaches its correct position in the sorted array. After iteration 2, 4 (the second largest element in the array) reaches its correct position in the sorted array and so on.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 7 3 6 10 -100	3 6 7 -100 10 3 6 -100 7 10 3 -100 6 7 10 -100 3 6 7 10
7 6 7 6 6 7 1 2	6 6 6 7 1 2 7 6 6 6 1 2 7 7 6 6 1 2 6 7 7 6 1 2 6 6 7 7 1 2 6 6 6 7 7 1 2 6 6 6 7 7
10 0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
10 9 8 7 6 5 4 3 2 1 0	8 7 6 5 4 3 2 1 0 9 7 6 5 4 3 2 1 0 8 9 6 5 4 3 2 1 0 7 8 9 5 4 3 2 1 0 6 7 8 9 4 3 2 1 0 5 6 7 8 9 3 2 1 0 4 5 6 7 8 9 2 1 0 3 4 5 6 7 8 9 1 0 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
9 2 3 4 -2 -2 -2 1 1 1	2 3 -2 -2 -2 1 1 1 4 2 -2 -2 -2 1 1 1 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4
2 5 4	4 5

Pretty Queues Revisited (p2v3d1)

Pretty Queues Revisited [20 marks]

Problem Statement
We have implemented queues in the previous weeks and used them to schedule malloc and free requests. In this problem, we will re-implement queues, but using linked lists. A queue is, as you may recall, a data structure which allows entry at the end and exit at the beginning (you may see Week 10 Tuesday question 1 problem statement from the practice problems in the folder LAB-PRAC-10_MAT-FUN if you have forgotten or have not written down details of queues in your notebook).

The queue is a FIFO data structure since the first element to enter the queue is also the first one to exit the queue. We will implement a queue using a linked lists where elements will be inserted (i.e. enqueued) at the tail of the linked list and removed (i.e dequeued) from the head of the linked list.

The first line of the input will give you n, a strictly positive integer. The next line will give you n integers, separated by a space. These are the occupants of a linked list we are going to construct. However, the linked list does not contain these elements in this order. Store these n integers in an array arr.

After this, in the next line, we will give you the index of the head of the linked list in the array arr. After this, there will be n-1 more lines which will contain pairs of numbers of the form a b (i.e. the two numbers will be separated by a space). This will indicate that the element at index a in the array arr points to the element at index b in the array arr. Create a linked list with n nodes with the elements in the given order. Please refer to the hints to see an easy way to do so. After this, on the next line, we will give you a strictly positive number t, indicating the number of operations we wish to perform on this queue. In the next t lines we will give you pairs of integers which are to be interpreted as follows

1. If the line contains 1 x where x is an integer (negative, positive or zero), insert the element at the end of the queue, i.e. insert x at the tail of the linked list and then print the updated queue (i.e. the updated linked list) on a new line
2. If the line contains 2 0, then dequeue the element at the front of the queue i.e. delete the head of the linked list. Print the value of the deleted element on a new line and then print the update queue (i.e. the updated linked list) on a new line.

In your output, in the first line, print the linked list by printing elements from the head to tail, two elements separated by a single space. There should be no spaces at the end of the line. The, for every operation we ask, perform that operation and print the required outputs. For insertion operations, simply print the updated linked list on a new line. For deletion operations, first print the deleted element on a new line and then print the updated linked list on a new line.

Caution

1. We assure you that n + t will be at most 100.
2. We assure you that we will never as you to dequeue from an empty queue, i.e. delete an element from an empty linked list. We additionally promise you that the queue will never become empty so that you do not have to worry about printing empty lists.
3. n may be any strictly positive integer, even 1 in which the linked list will have a single element.
4. Elements may repeat in the initial list we give you, as well as the same integer may get enqueued again in the operations. An integer that has been dequeued (from the head) may be asked to be enqueued again (at the tail) in a subsequent operation.
5. The integers in the list, as well as those being inserted (enqueued) may be positive, negative or zero.
6. There should be no extra spaces at the end of any of the lines.

HINTS: Following these steps might make your life easier:

1. Create a structure node to store an integer and a pointer.
2. Make an array of 100 nodes to store the n integers as given as well as the pointers to the next element. This array will also store any new elements that are enqueued into the queue. As we have promised, n + t will always be less than or equal to 100.
3. Store the location of the head node as well as the tail node. It will help you maintain the queue easily. You may maintain location of the head and the tail nodes as global variables to help make your code easier. However, it is not very good programming practice.
4. For each pair (a b) given as input, make the node at index a in the array point to the node at index b in the array
5. Write a function to print a linked list.
6. If asked to insert an element into the list, the tail will get updated since elements get enqueued at the tail of a queue. You may store the newly inserted elements in the array itself. Since n + t is never more than 100, you will never run out of space.
7. If asked to delete an element , the head will get updated since elements get dequeued from the head of a queue.

Note that we are asking you to use a static data structure like an array, to store a linked list, only to make this problem less complicated. What you have created is not a very efficient dynamic data structure.

EXAMPLE:
INPUT
5
1 2 3 4 5
2
1 0
3 4
0 3
2 1
3
1 6
1 7
2 0

OUTPUT:
3 2 1 4 5
3 2 1 4 5 6
3 2 1 4 5 6 7
3
2 1 4 5 6 7

Explanation: The head is index 2 in the array i.e. the element 3. The links in the linked list are shown below
2 => 1
4 => 5
1 => 4
3 => 2

Thus, the linked list is 3 => 2 => 1 => 4 => 5 with 3 as the head and 5 as the tail. Three operations are to be performed on the linked list.

1. First we enqueue 6 at the tail of the linked list. The list looks like 3 => 2 => 1 => 4 => 5 => 6 after this step.
2. We then enqueue 7 at the tail of the linked list. The list looks like 3 => 2 => 1 => 4 => 5 => 6 => 7 after this step.
3. We then dequeue an element from the head of the linked list. That element is 3 and the new list is 2 => 1 => 4 => 5 => 6 => 7 after this step.

Note that in case of deletion, we first print the deleted (dequeued) element on a new line and then on a new line, print the new state of the linked list.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 1 2 3 4 5 2 1 0 3 4 0 3 2 1 3 1 6 1 7 2 0	 3 2 1 4 5 3 2 1 4 5 6 3 2 1 4 5 6 7 3 2 1 4 5 6 7
4 23 39 5 37 0 0 1 1 2 2 3 5 1 1 1 2 2 0 2 0 2 0	 23 39 5 37 23 39 5 37 1 23 39 5 37 1 2 23 39 5 37 1 2 39 5 37 1 2 5 37 1 2
1 5 0 4 1 5 1 5	5 5 5 5 5 5 5 5 5

2 0 2 0	5 5
3 1 2 3 2 2 1 1 0 2 1 3 2 0	3 2 1 3 2 1 3 3 2 1 3
2 4 3 1 1 0 5 1 5 1 6 1 7 1 -1 1 -2	3 4 3 4 5 3 4 5 6 3 4 5 6 7 3 4 5 6 7 -1 3 4 5 6 7 -1 -2
7 1 2 3 4 5 6 7 6 3 2 5 4 4 3 2 1 1 0 6 5 4 1 0 2 0 2 0 1 9	7 6 5 4 3 2 1 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 6 5 4 3 2 1 0 5 4 3 2 1 0 9

Just About Sorted (p3v1d1)

Just About Sorted [20 marks]

Problem Statement
Mr C took a lot of pain to sort an array in non-decreasing order. However, one of the mischievous clones came and messed up the array by rotating the array right by a few elements. For example, if the nice original sorted array was
1 2 3 4 5
then rotating it right by one location would give the array
5 1 2 3 4
Rotating the original sorted array two locations to the right would give the array
4 5 1 2 3
Rotating the original sorted array four locations to the right would give the array
2 3 4 5 1
We will not consider rotations by five locations since that would give back the original array. Thus, if the array has n elements, then the clone could have rotated it right by 0 locations (i.e. no change), 1 location, 2

locations, ... n-1 locations.

The first line of the input will give you n, the number of elements in the array. In the next line we will give you n integers, separated by a space. These will constitute the array which was sorted and then rotated to the right by a certain number of locations. In the third line of the input we will give you a query integer q.

In the first line of the output, print by how many locations to the right did the clone rotate the original sorted array. Remember that if there are n elements in the array, then this number can be between 0 and n-1 (both included). In the second line of the output print the smallest index at which the query element occurs, if it does occur in the given (possibly rotated) array, else print -1 to indicate that the query element is not present in the given array at all.

You would see that both the above problems are very easy to solve using O(n) time. The challenge is to solve these using O(log n) time using techniques we learnt in binary search (i.e. divide and conquer).

Caution

1. You have to print the location of the query element in the array that is given to you, i.e. the array that was possibly rotated after being sorted. Do not give the location of the query element in the original sorted array.
2. If the query element appears twice in the given (possibly rotated) array, print the smallest index at which it occurs.
3. The array elements may be positive, negative or zero. The query element may also be positive, negative or zero.
4. The array may contain repeated elements. However, we promise you that there will be at least two distinct elements in the array i.e. the original sorted array will never look like 6 6 6 6 6 6 6 since rotations do not make sense in such an array. However, the original sorted array may look like 6 6 6 6 6 7.
5. Be careful about extra/missing lines and extra/missing spaces in your output.

EXAMPLE:
INPUT
5
3 4 5 1 2
5

OUTPUT:
3
2

Explanation: The original sorted array was 1 2 3 4 5 and it was rotated 3 locations to the right to get 3 4 5 1 2. The element 5 is present in the given array but at index 2.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 3 4 5 1 2 7	3 -1
6 5 6 1 2 2 4 2	2 3
10 2 3 4 5 6 -3 -2 -1 0 1 12	5 -1
8 2 2 2 2 2 2 2 1 1	7 7
9 1 1 1 1 1 1 1 2 3 2	0 7
10 4 5 5 1 1 2 2 3 3 4 4	3 0

Brick Sort (p3v2d1)

Brick Sort [20 marks]

Problem Statement
There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One simple algorithm is called Brick Sort. The algorithm got its name since it sorts alternating elements of the array in subsequent phases, so that the whole arrangement (see figure below) looks like bricks have been laid down. This variant was originally designed for parallel processing and may work very well on parallel computing architectures such as GPUs.

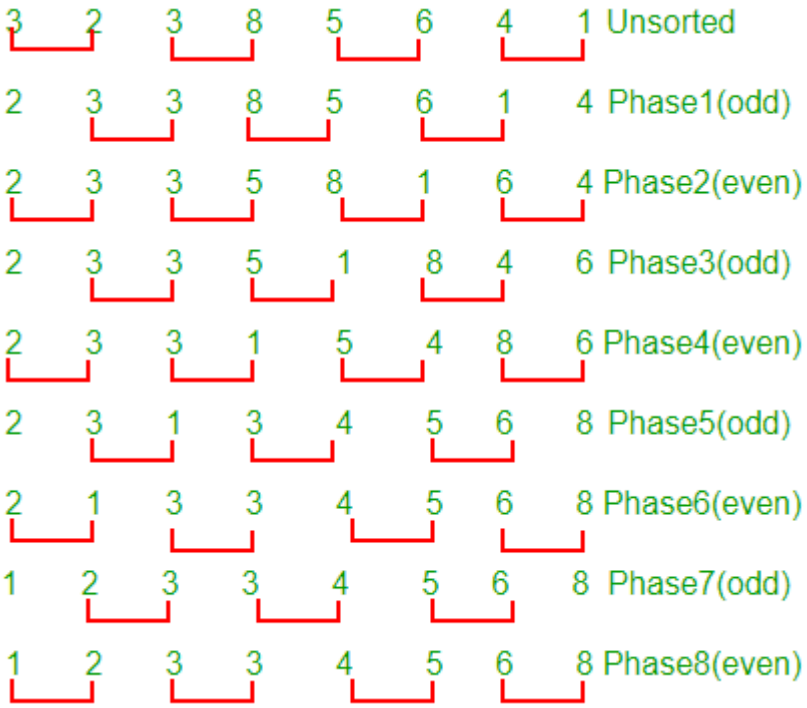
Brick sort proceeds in n phases. In the first phase, even indices of the array get compared to the element to their immediate right and if the pair is out of order (i.e. the element at the even index is strictly greater than the element to its immediate right), then the pair is swapped. In the second phase, odd indices of the array get compared (and swapped if out of order) to the element to their immediate right. The third phase again considers the even indices and so on. If the array has n elements, then the algorithm works for n phases. The array is guaranteed to be sorted at the end of the n-th phase.

The first line of the input will give you n, a strictly positive integer and the second line will give you n integers, separated by a space. Store these numbers in an array of size n. In your output, you have to print the array after each phase of brick sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space. However, there should be no space at the end of a line. The image below shows you how brick sort works. Also notice that at some iterations (e.g. phase 7), nothing needs to be done.

Now, we agree that at first it is not entirely clear why this algorithm should even be correct (i.e. why should it completely sort the array in n phases). The proof of correctness of this algorithm is a bit involved and is based on a powerful meta-theorem by Donald Knuth called the 0-1 principle. Check it out if you are

interested.

Image courtesy: geeksforgeeks.org (image modified for better clarity)



Caution

1. If the array has an odd number of elements, say 5, then the index 4 element will have no element to its right in the phases when even elements are being compared to the element to their immediate right (for example the first phase). So nothing will need to be done. However, the index 4 element will get compared to the index 3 element in the 2nd pass when odd index elements are being compared to their immediate neighbors to the right.
2. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order insertion sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
3. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.
4. The number of elements n can be any strictly positive number, even 1. Your output must have exactly n lines.
5. Some phases may not require any work. However, you must still print the array after those phases. Your output must have exactly n lines.
6. Be careful about extra/missing lines and extra/missing spaces in your output. There should be no space at the end of any line in your output, nor should there be any extra newlines at the end of your output.

EXAMPLE:
INPUT
4
3 4 1 2

OUTPUT:
3 4 1 2
3 1 4 2
1 3 2 4
1 2 3 4

Explanation:
First phase
3 4 1 2 (Checking for swap between 3 & 4 - nothing to be done)
3 4 1 2 (Checking for swap between 1 & 2 - nothing to be done)
No other even index elements left.

Second phase
3 1 4 2 (Checking for swap between 4 & 1 - swap them)
No other odd index elements left.

Third phase
1 3 4 2 (Checking for swap between 3 & 1 - swap them)
1 3 2 4 (Checking for swap between 4 & 2 - swap them)
No other even index elements left.

Fourth phase
1 2 3 4 (Checking for swap between 3 & 2 - swap them)
No other odd index elements left.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 5 3 6 11 7	3 5 6 11 7 3 5 6 7 11 3 5 6 7 11 3 5 6 7 11 3 5 6 7 11
9 9 8 7 6 5 4 3 2 1	8 9 6 7 4 5 2 3 1 8 6 9 4 7 2 5 1 3 6 8 4 9 2 7 1 5 3 6 4 8 2 9 1 7 3 5 4 6 2 8 1 9 3 7 5 4 2 6 1 8 3 9 5 7 2 4 1 6 3 8 5 9 7 2 1 4 3 6 5 8 7 9 1 2 3 4 5 6 7 8 9
4 23 39 5 37	23 39 5 37 23 5 39 37 5 23 37 39 5 23 37 39
8 2 3 4 -2 -2 -5 1 -11	2 3 -2 4 -5 -2 -11 1 2 -2 3 -5 4 -11 -2 1

	-2 2 -5 3 -11 4 -2 1 -2 -5 2 -11 3 -2 4 1 -5 -2 -11 2 -2 3 1 4 -5 -11 -2 -2 2 1 3 4 -11 -5 -2 -2 1 2 3 4 -11 -5 -2 -2 1 2 3 4
8 9 6 8 9 11 8 9 7	6 9 8 9 8 11 7 9 6 8 9 8 9 7 11 9 6 8 8 9 7 9 9 11 6 8 8 7 9 9 9 11 6 8 7 8 9 9 9 11 6 7 8 8 9 9 9 11 6 7 8 8 9 9 9 11 6 7 8 8 9 9 9 11
2 4 3	3 4 3 4

All My Descendants (p3v3d1)

All My Descendants [20 marks]

Problem Statement
We have read about binary trees in the tutorial. These are data structures where all nodes, except the root, have exactly one parent, and all nodes may have a left child and a right child. Nodes that have no children are called leaves. In this problem, we will construct a binary tree using the concepts we learnt while studying linked lists.

The first line of the input will give you n, a strictly positive integer. This will be the number of nodes in the tree. The next line will give you n integers, separated by a space. These are the occupants of a binary tree we are going to construct. However, the binary tree does not contain these elements in this order. Store these n integers in an array arr (please refer to the hint below to see how to do so).

After this, in the next line, we will give you the index of the root of the binary tree in the array arr. After this, there will be n lines telling you which are the children of these n nodes. Each line will contain a triplet of numbers a b c (i.e. the three numbers will be separated by a space). This will indicate that the element at index a in the array arr has the element at index b in the array arr, as its left child, and the element at index c in the array arr, as its right child. If a node has no left child, b will be -1. If a node has no right child, c will be -1. The last line of the input will give you the index q of an element in the array arr. Call the node corresponding to q as the query node. q will be between 0 and n-1 (both included).

Create a binary tree with n nodes with the elements in the given arrangement. Please refer to the hints to see an easy way to do so. In the first line of the input, print the value of the number stored at the query node. In the next line, print the value of the number stored in the left child of the query node (if there is no left child print -1). In the next line, print the value of the number stored in the right child of the query node (if there is no right child print -1). In the fourth and final line of the output, print the sum of values stored at q and all its descendants (i.e. the sum of value stored at the query node + sum of values stored in its children, however many they may be + sum of values stored in is grandchildren, however many they may be + values stored at great-grand children, and so on).

P.S. The name of this problem is derived from that of a television serial called "All My Children" that used to air in the US. The serial ran for a ridiculous 41 years, spanning multiple resets and relaunches, and finally ended in 2011

Caution

1. A node may have a left child but no right child. A node may have a right child but no left child. Leaves have neither a left child nor a right child.
2. If the query node is a leaf, then the sum will simply be the number stored at the node itself. Print -1 for values stored in the left and right children since there are no left and right children for leaves.
3. The numbers being stored in the nodes of the tree may be positive, negative or zero. Numbers may repeat in multiple nodes of the tree. Remember, this is just a binary tree, not a binary search tree. So there is no restriction on which number may be stored in which node.
4. n may be any strictly positive integer, even 1 in which the tree will have a single node - the root.
5. There are four lines of output. There should be no extra spaces at the end of any of the lines.

HINTS: Following these steps might make your life easier:

1. Create a structure node to store an integer and a left pointer and a right pointer.
2. Make an array of n nodes to store the n integers as given as well as the pointers to the left and the right children.
3. Store the location of the root node
4. For each triplet (a b c) given as input, make the node at index a in the array point to the node at index b in the array as its left child and the point to the node at index c in the array as its right child
5. Try to write a recursive function to print the sum of all descendants of a node.

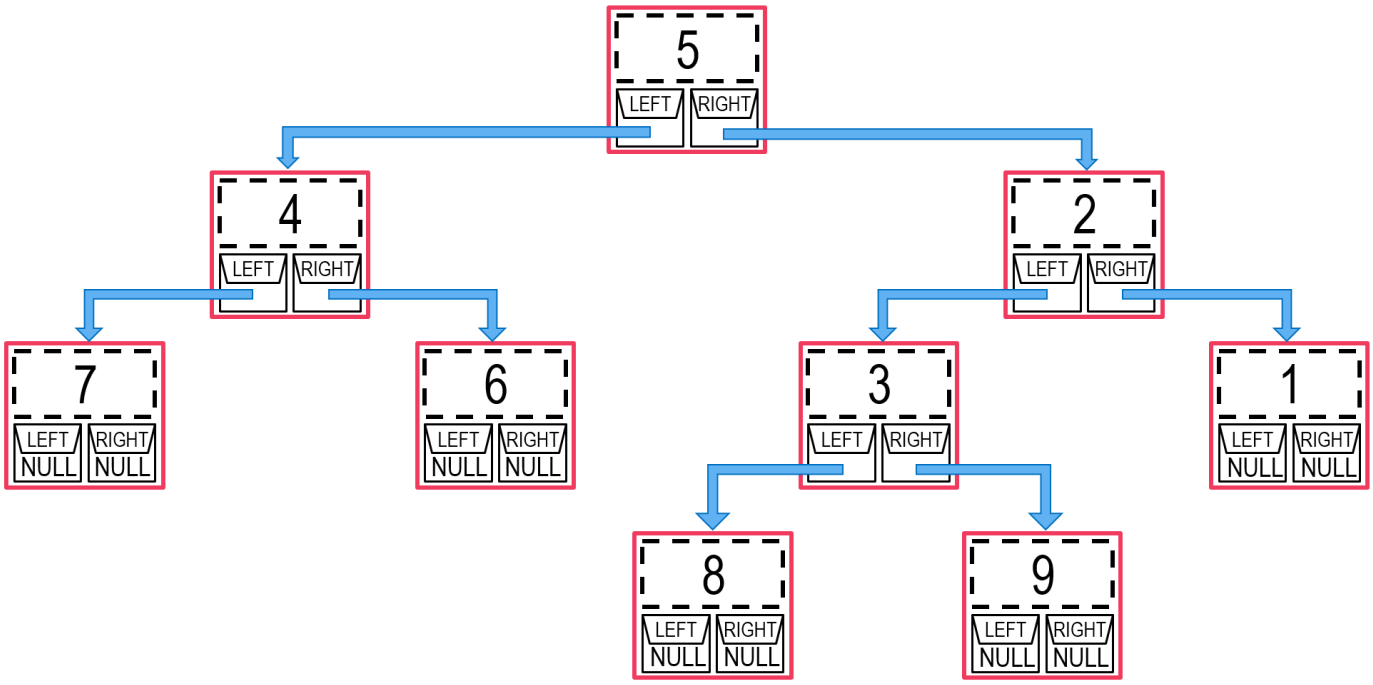
Note that we are asking you to use a static data structure like an array, to store a binary tree, which is a dynamic data structure, only to make this problem less complicated. What you have created is not a very efficient dynamic data structure.

EXAMPLE:
INPUT
9
1 2 3 4 5 6 7 8 9
4
4 3 1
2 7 8
7 â€“1 â€“1
3 6 5
5 â€“1 â€“1
6 â€“1 â€“1
8 â€“1 â€“1
1 2 0
0 â€“1 â€“1
3

OUTPUT:
4
7
6
17

Explanation: See the following image to see what this tree looks like. The node at index 3 in the array is 4 and its children are 7 and 6. The sum of all nodes which are descendants of 4 (including 4 itself) is 17.

For example, had q been 1 i.e. the query node been the node that stores the number 2, then the answer would have been
2
3
1
23



Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are four lines in your output. The first three lines carry 10% weightage and the last line carries 70% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
9 1 2 3 4 5 6 7 8 9 4 4 3 1 2 7 8 7 -1 -1 3 6 5 5 -1 -1 6 -1 -1 8 -1 -1 1 2 0 0 -1 -1 1	2 3 1 23

9 1 2 3 4 5 6 7 8 9 4 4 3 1 2 7 8 7 -1 -1 3 6 5 5 -1 -1 6 -1 -1 8 -1 -1 1 2 0 0 -1 -1 4	5 4 2 45
1 5 0 0 -1 -1 0	5 -1 -1 5
10 0 1 2 3 4 5 6 7 8 9 5 0 -1 -1 2 -1 -1 1 0 2 4 -1 -1 3 1 4 8 -1 -1 9 8 -1 7 6 9 6 -1 -1 5 3 7 2	2 -1 -1 2
10 10 11 12 13 14 15 16 17 18 19 5 0 -1 -1 2 -1 -1 1 0 2 4 -1 -1 3 1 4 8 -1 -1 9 8 -1 7 6 9 6 -1 -1 5 3 7 9	19 18 -1 37
10 100 11 12 13 14 15 16 17 18 19 5 0 -1 -1 2 -1 -1 1 0 2 4 -1 -1 3 1 4 8 -1 -1 9 8 -1	17 16 19 70

7 6 9 6 -1 -1 5 3 7 7	

Mr C likes a Majority (p4v1d1)

Mr C likes a Majority [20 marks]

Problem Statement
In the first line of the input we will give you n, a strictly positive odd number. In the second line we will give you n integers separated by a space. The numbers will be sorted in non-decreasing order. Store these numbers in an array called arr.

In the first line of the output, you have to tell us if there is any number in the array that appears a majority number of times i.e. ceil(n/2) times or more (recall that n will always be odd). If that is the case, print "YES" (without quotes) in the first line of the output. In the second line of the output, print the number of times the majority number appears in the array. If no number appears a majority number of times, print "NO" (without quotes) in the first line of the output, followed by the number of times the first element (i.e. arr[0]) appears in the array in the second line.

You would see that both the above problems are very easy to solve using O(n) time. The challenge is to solve these using O(log n) time using techniques we learnt in binary search (i.e. divide and conquer).

Caution

1. n may be any strictly positive odd number, even 1.
2. The numbers we give you may be negative, positive, or zero. The same number may repeat in the array. However, the array will always be sorted in non-decreasing order.
3. There should be no spaces in your output. There are two lines in your output.

EXAMPLE 1:
INPUT
7
1 1 2 2 3 3 3

OUTPUT:
NO
2

Explanation: Since n = 7, an element needs to occur 4 or more times to be in majority (ceil(7/2) = 4). No such element exists. The first element of the array appears 2 times.

EXAMPLE 2:
INPUT
9
1 3 4 4 4 4 4 4 8

OUTPUT:

YES
6

Explanation: Since n = 9, an element needs to occur 5 or more times to be in majority (ceil(9/2) = 5). The element 4 does appear 6 times.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are two lines in your output. Printing each line correctly, in the correct order, carries 50% weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
7 1 1 2 2 3 3 3	NO 2
9 1 3 4 4 4 4 4 4 8	YES 6
11 11 11 11 11 11 11 11 11 11 11 11	YES 11
13 -1 2 5 6 6 6 7 7 7 7 7 7 7	YES 7
9 1 2 3 4 5 6 7 8 9	NO 1
15 1 1 1 1 1 1 1 1 2 3 4 5 6 7 8	YES 8

Cocktail Sort (p4v2d1)

Cocktail Sort [20 marks]

Problem Statement
There are several sorting algorithms that have been developed over the years. We saw some during the lectures and will explore others here. One simple algorithm is called Cocktail Sort and the reason it is called this name is due to a back and forth behavior it has which is reminiscent of the shaking motion bartenders perform when preparing cocktails at a bar. The Cocktail Sort is a close cousin of another simple sorting algorithm known as Bubble Sort. As the name suggests, bubble sort causes elements to "bubble" up to their correct position in the array.

Cocktail Sort maintains the following invariant. After the first iteration, the largest element in the array should be in the last position of the array (as it should be in a non-decreasing sorted order). After the second iteration, the smallest element in the array must be in the first position of the array (as it should be in a non-

decreasing sorted order). After the third iteration, the second-largest element in the array should be in the second-last position of the array (as it should be in a non-decreasing sorted order). After the fourth iteration, the second-smallest element in the array must be in the second position of the array (as it should be in a non-decreasing sorted order) - this process repeats for a total of n iterations after which the array must get sorted.

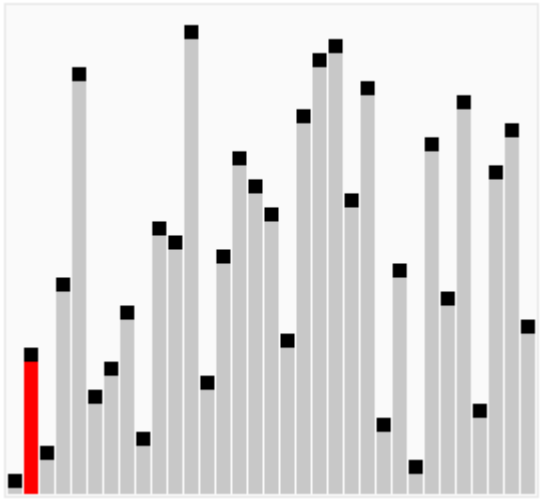
The way cocktail sort accomplishes the above is actually very cute. In odd iterations i.e. 1st iteration, 3rd iteration etc, - it goes from left to right, comparing adjacent elements. If it finds an element that is strictly larger than the element to its immediate right (i.e. the elements are out of order), it swaps the two elements. In even iterations i.e. 2nd iteration, 4th iteration etc, - it goes from right to left, comparing adjacent elements. If it finds an element that is strictly larger than the element to its immediate right (i.e. the elements are out of order), it swaps the two elements.

Verify that if we do this, then in the first iteration, we would end up transporting the largest element of the array to the last position in the array. Also, in the second iteration, we would end up transporting the smallest element of the array to the first position in the array.

The first line of the input will give you n, a strictly positive integer and the second line will give you n integers, separated by a space. Store these numbers in an array of size n. In your output, you have to print the array after each iteration of cocktail sort, on a separate line. Print the array by printing each number in the array from left to right with two numbers separated by a single space. However, there should be no space at the end of a line.

The animation below shows you how cocktail sort works. Notice that first the algorithm does a left to right sweep in the first iteration, then a right to left sweep in the second iteration and then a left to right sweep again in the third iteration and so on, for n iterations.

Image courtesy: wikipedia.org



Caution

1. Cocktail sort goes over the array from left to right, starting from the index 0, comparing adjacent locations and swapping them if they are out of order, then going from right to left and doing the same thing and repeating this over and over again. However, you should be able to deduce that it does not need to go all the way till the left most and right most indices in later iterations.
2. There may be iterations where nothing needs to be done. However you still have to print the array after that iteration. Your output must contain n lines.
3. Please do not try to cheat by using library functions like qsort(). These will not sort the array in the order bubble sort will and hence you will not get partial marks for printing the intermediate steps of the algorithm.
4. The n numbers we give you may be positive, negative or zero. The same number may occur twice in the list too.

5. The number of elements n can be any strictly positive number, even 1. Your output must have exactly n lines.
6. Be careful about extra/missing lines and extra/missing spaces in your output. There should be no space at the end of any line in your output, nor should there be any extra newlines at the end of your output.

EXAMPLE:
INPUT
4
3 4 1 2

OUTPUT:
3 1 2 4
1 3 2 4
1 2 3 4
1 2 3 4

Explanation: Initial state of the array is 3 4 1 2
Iteration 1:

1. 3 and 4 get compared. They are in correct order so no change
2. 4 and 1 get compared. They are out of order so they get swapped. 3 1 4 2
3. 4 and 2 get compared. They are out of order so they get swapped. 3 1 2 4

State of array after iteration 1: 3 1 2 4
Iteration 2:

1. 2 and 4 get compared. They are in correct order so no change
2. 1 and 2 get compared. They are in correct order so no change
3. 3 and 1 get compared. They are out of order so they get swapped. 1 3 2 4

State of array after iteration 2: 1 3 2 4
Iteration 3:

1. 1 and 3 get compared. They are in correct order so no change
2. 3 and 2 get compared. They are out of order so they get swapped. 1 2 3 4
3. 3 and 4 get compared. They are in correct order so no change

State of array after iteration 3: 1 2 3 4
Iteration 4:

1. 1 and 2 get compared. They are in correct order so no change
2. 2 and 3 get compared. They are in correct order so no change
3. 3 and 4 get compared. They are in correct order so no change

State of array after iteration 4: 1 2 3 4

Notice that after iteration 1, 4 (the largest element in the array) reaches its correct position in the sorted array. After iteration 2, 1 (the smallest element in the array) reaches its correct position in the sorted array and so

on.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
5 7 3 6 10 -100	3 6 7 -100 10 -100 3 6 7 10 -100 3 6 7 10 -100 3 6 7 10 -100 3 6 7 10
7 6 7 6 6 7 1 2	6 6 6 7 1 2 7 1 6 6 6 7 2 7 1 6 6 6 2 7 7 1 2 6 6 6 7 7 1 2 6 6 6 7 7 1 2 6 6 6 7 7 1 2 6 6 6 7 7
10 0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
10 9 8 7 6 5 4 3 2 1 0	8 7 6 5 4 3 2 1 0 9 0 8 7 6 5 4 3 2 1 9 0 7 6 5 4 3 2 1 8 9 0 1 7 6 5 4 3 2 8 9 0 1 6 5 4 3 2 7 8 9 0 1 2 6 5 4 3 7 8 9 0 1 2 5 4 3 6 7 8 9 0 1 2 3 5 4 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
9 2 3 4 -2 -2 -2 1 1 1	2 3 -2 -2 -2 1 1 1 4 -2 2 3 -2 -2 1 1 1 4 -2 2 -2 -2 1 1 1 3 4 -2 -2 2 -2 1 1 1 3 4 -2 -2 -2 1 1 1 2 3 4

	-2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4 -2 -2 -2 1 1 1 2 3 4
1 7	7

All My Descendants - Part II (p4v3d1)

All My Descendants - Part II [20 marks]

Problem Statement
We have read about binary trees in the tutorial. These are data structures where all nodes, except the root, have exactly one parent, and all nodes may have a left child and a right child. Nodes that have no children are called leaves. In this problem, we will construct a binary tree using the concepts we learnt while studying linked lists.

The first line of the input will give you n, a strictly positive integer. This will be the number of nodes in the tree. The next line will give you n integers, separated by a space. These are the occupants of a binary tree we are going to construct. However, the binary tree does not contain these elements in this order. Store these n integers in an array arr (please refer to the hint below to see how to do so).

After this, in the next line, we will give you the index of the root of the binary tree in the array arr. After this, there will be n lines telling you which are the children of these n nodes. Each line will contain a triplet of numbers a b c (i.e. the three numbers will be separated by a space). This will indicate that the element at index a in the array arr has the element at index b in the array arr, as its left child, and the element at index c in the array arr, as its right child. If a node has no left child, b will be -1. If a node has no right child, c will be -1.

Create a binary tree with n nodes with the elements in the given arrangement. Please refer to the hints to see an easy way to do so. In your output, you need to print the nodes of the tree in what is known as an *in-order traversal*. Traversal in a linked list is simple - just for from left to right!. However, in a tree, it is not clear in which order should the nodes be visited. In-order traversal is just one example of many possible ways to do so. The in-order traversal of a binary tree is recursively defined as follows.

Suppose we have a binary tree with a root which has a left child and a right child (see example below). Notice that the left child and the right child can have their own descendants. The left child and all its descendants are collectively called the left subtree of the root node. The right child and all its descendants are collectively called the right subtree of the root node. Note that the left and right subtrees are binary trees themselves.

The in-order traversal of a binary tree is defined as the in-order traversal of the left subtree of the root followed by the value stored at the root itself, followed by the in-order traversal of the right subtree of the root. The in-order traversal of a tree whose root does not have a left child is simply the value at the root followed by the in-order traversal of the right subtree of the root. The in-order traversal of a tree whose root does not have a right child is simply the in-order traversal of the left subtree of the root followed by the value at the root. The in-order traversal of a binary tree whose root has no children is simply the value at the root itself.

In your output you have to print the in-order traversal of the tree we have given you. Print each element on a different line. Be careful of the order in which you print elements. You will get partial marks only if you print the correct element at its correct location. There should be no spaces in your output in any line. Note that this definition is inherently recursive so you should use recursion to solve this problem.

P.S. The name of this problem is derived from that of a television serial called "All My Children" that used to air in the US. The serial ran for a ridiculous 41 years, spanning multiple resets and relaunches, and finally ended in 2011

Caution

1. A node may have a left child but no right child. A node may have a right child but no left child. Leaves have neither a left child nor a right child.
2. The numbers being stored in the nodes of the tree may be positive, negative or zero. Numbers may repeat in multiple nodes of the tree. Remember, this is just a binary tree, not a binary search tree. So there is no restriction on which number may be stored in which node.
3. n may be any strictly positive integer, even 1 in which the tree will have a single node - the root.
4. We will not penalize you for extra newlines at the end of your output. However, there should be no extra spaces at the end of any of the lines in your output.

HINTS: Following these steps might make your life easier:

1. Create a structure node to store an integer and a left pointer and a right pointer.
2. Make an array of n nodes to store the n integers as given as well as the pointers to the left and the right children.
3. Store the location of the root node
4. For each triplet (a b c) given as input, make the node at index a in the array point to the node at index b in the array as its left child and the point to the node at index c in the array as its right child
5. Try to write a recursive function to print the in-order traversal of the tree.

Note that we are asking you to use a static data structure like an array, to store a binary tree, which is a dynamic data structure, only to make this problem less complicated. What you have created is not a very efficient dynamic data structure.

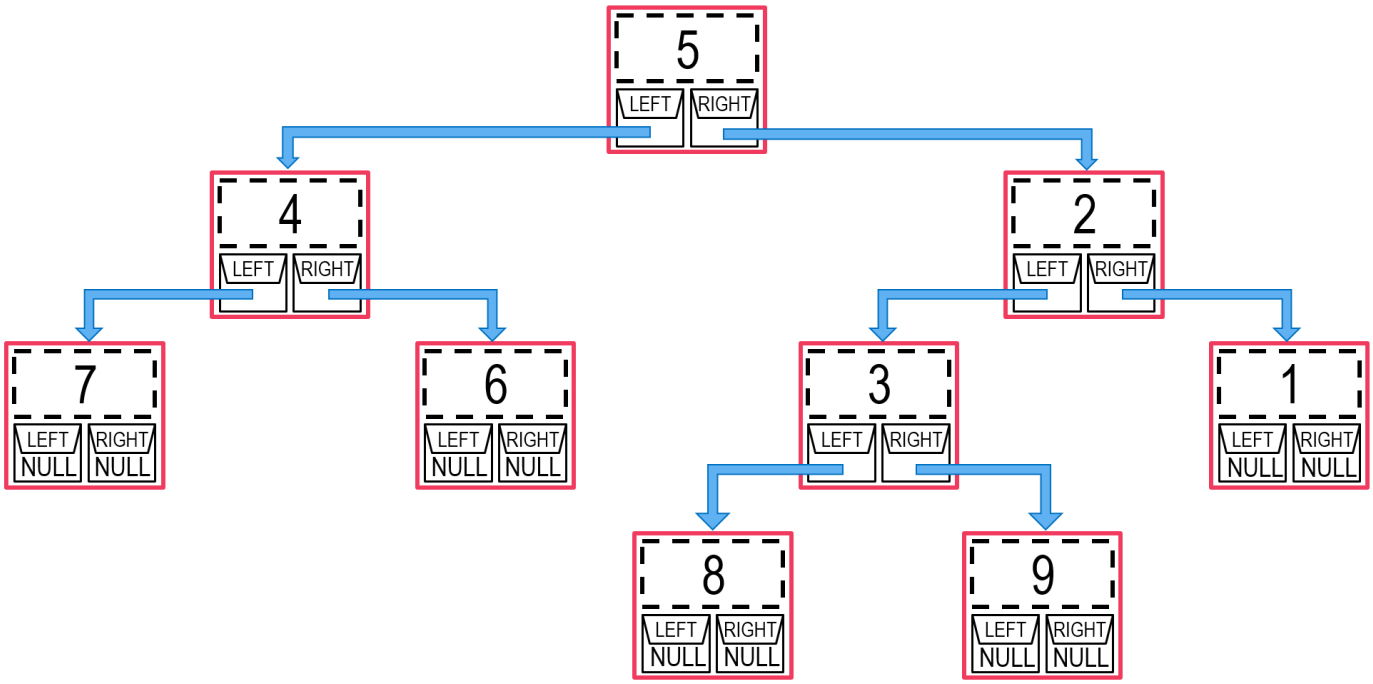
EXAMPLE:
INPUT
9
1 2 3 4 5 6 7 8 9
4
4 3 1
2 7 8
7 -1 -1
3 6 5
5 -1 -1
6 -1 -1
8 -1 -1
1 2 0
0 -1 -1

OUTPUT:
7
4
6
5
8
3

9
2
1

Explanation: See the following image to see what this tree looks like.

1. The in-order traversal of the left subtree of the root is 7 4 6 since the left subtree has 4 as the root which has its left child as 7 and right child as 6. Thus, the left subtree and the right subree of the node 4 have only node each.
2. The in-order traversal of the right subtree of the root is 8 3 9 2 1 since the right subtree has 2 as the root which has its left child as 3 and right child as 1. Thus, the left subtree of the node 2 has the in-order traversal 8 3 9 and the right subree of the node 2 has only one node.



Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. All lines carry equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

All Test Cases (Visible + Hidden)

Input	Output
9 1 2 3 4 5 6 7 8 9 4 4 3 1 2 7 8	7 4 6 5 8

7 -1 -1 3 6 5 5 -1 -1 6 -1 -1 8 -1 -1 1 2 0 0 -1 -1	3 9 2 1
3 0 1 2 1 0 -1 -1 2 -1 -1 1 0 2	0 1 2
1 5 0 0 -1 -1	5
10 100 11 12 13 14 15 16 17 18 19 5 0 -1 -1 2 -1 -1 1 0 2 4 -1 -1 3 1 4 8 -1 -1 9 8 -1 7 6 9 6 -1 -1 5 3 7	100 11 12 13 14 15 16 17 18 19
4 2 2 2 2 1 1 0 2 0 -1 -1 2 -1 3 3 -1 -1	2 2 2 2
4 10 21 32 43 3 2 -1 1 3 -1 2 1 -1 0 0 -1 -1	43 32 21 10