Programming Assignment 4: Binary Search Trees

Revision: August 10, 2019

Introduction

In this programming assignment, you will practice implementing binary search trees including balanced ones and using them to solve algorithmic problems. In some cases you will just implement an algorithm from the lectures, while in others you will need to invent an algorithm to solve the given problem using hashing.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply binary search trees to solve the given algorithmic problems.
2. Implement in-order, pre-order and post-order traversal of a binary tree.
3. Implement a data structure to compute range sums.
4. Implement a data structure that can store strings and quickly cut parts and patch them back.

Passing Criteria: 3 out of 5

Passing this programming assignment requires passing at least 3 out of 5 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Binary tree traversals

Problem Introduction

In this problem you will implement in-order, pre-order and post-order traversals of a binary tree. These traversals were defined in the week 1 lecture on tree traversals, but it is very useful to practice implementing them to understand binary search trees better.

Problem Description

Task. You are given a rooted binary tree. Build and output its in-order, pre-order and post-order traversals.

Input Format. The first line contains the number of vertices n. The vertices of the tree are numbered from 0 to $n - 1$. Vertex 0 is the root.

The next n lines contain information about vertices 0, 1, ..., $n-1$ in order. Each of these lines contains three integers key_i, $left_i$ and $right_i$ — key_i is the key of the i-th vertex, $left_i$ is the index of the left child of the i-th vertex, and $right_i$ is the index of the right child of the i-th vertex. If i doesn't have left or right child (or both), the corresponding $left_i$ or $right_i$ (or both) will be equal to -1.

Constraints. $1 \le n \le 10^5$; $0 \le key_i \le 10^9$; $-1 \le left_i, right_i \le n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex.

Output Format. Print three lines. The first line should contain the keys of the vertices in the in-order traversal of the tree. The second line should contain the keys of the vertices in the pre-order traversal of the tree. The third line should contain the keys of the vertices in the post-order traversal of the tree.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	12	6	1.5	2	6	6	12

Memory Limit. 512MB.

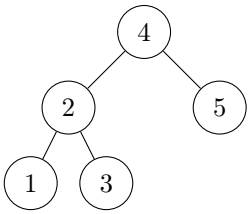
Sample 1.

Input:

```
5
4 1 2
2 3 4
5 -1 -1
1 -1 -1
3 -1 -1
```

Output:

```
1 2 3 4 5
4 2 1 3 5
1 3 2 5 4
```



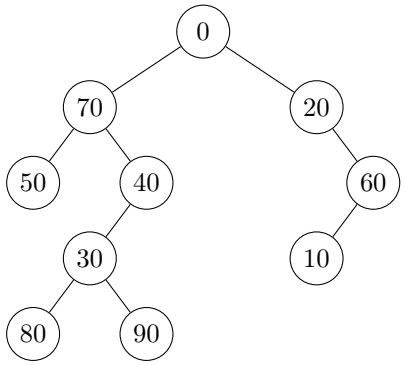
Sample 2.

Input:

```
10
0 7 2
10 -1 -1
20 -1 6
30 8 9
40 3 -1
50 -1 -1
60 1 -1
70 5 4
80 -1 -1
90 -1 -1
```

Output:

```
50 70 80 30 90 40 0 20 10 60
0 70 50 40 30 80 90 20 60 10
50 80 90 30 40 70 10 60 20 0
```



Starter Files

There are starter solutions only for C++, Java and Python3, and if you use other languages, you need to implement solution from scratch. Starter solutions read the input, define the methods to compute different traversals of the binary tree and write the output. You need to implement the traversal methods.

What to Do

Implement the traversal algorithms from the lectures. Note that the tree can be very deep in this problem, so you should be careful to avoid stack overflow problems if you're using recursion, and definitely test your solution on a tree with the maximum possible height.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

2 Is it a binary search tree?

Problem Introduction

In this problem you are going to test whether a binary search tree data structure from some programming language library was implemented correctly. There is already a program that plays with this data structure by inserting, removing, searching integers in the data structure and outputs the state of the internal binary tree after each operation. Now you need to test whether the given binary tree is indeed a correct binary search tree. In other words, you want to ensure that you can search for integers in this binary tree using binary search through the tree, and you will always get correct result: if the integer is in the tree, you will find it, otherwise you will not.

Problem Description

Task. You are given a binary tree with integers as its keys. You need to test whether it is a correct binary search tree. The definition of the binary search tree is the following: for any node of the tree, if its key is x, then for any node in its left subtree its key must be strictly less than x, and for any node in its right subtree its key must be strictly greater than x. In other words, smaller elements are to the left, and bigger elements are to the right. You need to check whether the given binary tree structure satisfies this condition. You are guaranteed that the input contains a valid binary tree. That is, it is a tree, and each node has at most two children.

Input Format. The first line contains the number of vertices n. The vertices of the tree are numbered from 0 to $n - 1$. Vertex 0 is the root.

The next n lines contain information about vertices 0, 1, ..., $n-1$ in order. Each of these lines contains three integers key_i, $left_i$ and $right_i$ — key_i is the key of the i-th vertex, $left_i$ is the index of the left child of the i-th vertex, and $right_i$ is the index of the right child of the i-th vertex. If i doesn't have left or right child (or both), the corresponding $left_i$ or $right_i$ (or both) will be equal to -1.

Constraints. $0 \le n \le 10^5$; $-2^{31} < key_i < 2^{31} - 1$; $-1 \le left_i, right_i \le n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex. All keys in the input will be different.

Output Format. If the given binary tree is a correct binary search tree (see the definition in the problem description), output one word "CORRECT" (without quotes). Otherwise, output one word "INCORRECT" (without quotes).

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

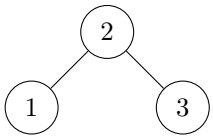
Sample 1.

Input:

```
3
2 1 2
1 -1 -1
3 -1 -1
```

Output:

```
CORRECT
```



Left child of the root has key 1, right child of the root has key 3, root has key 2, so everything to the left is smaller, everything to the right is bigger.

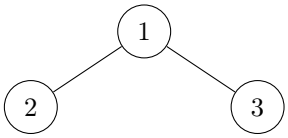
Sample 2.

Input:

```
3
1 1 2
2 -1 -1
3 -1 -1
```

Output:

```
INCORRECT
```



The left child of the root must have smaller key than the root.

Sample 3.

Input:

```
0
```

Output:

```
CORRECT
```

Empty tree is considered correct.

Sample 4.

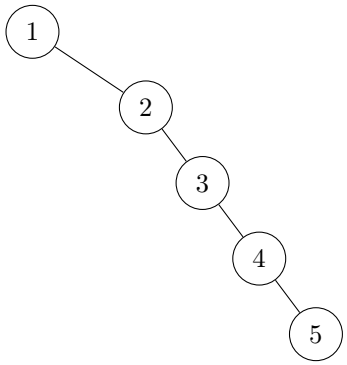
Input:

```
5
1 -1 1
2 -1 2
3 -1 3
4 -1 4
5 -1 -1
```

Output:

```
CORRECT
```

Explanation:



The tree doesn't have to be balanced. We only need to test whether it is a correct binary search tree, which the tree in this example is.

Sample 5.

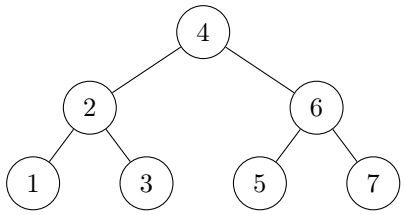
Input:

```
7
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
```

Output:

```
CORRECT
```

Explanation:



This is a full binary tree, and the property of the binary search tree is satisfied in every node.

Sample 6.

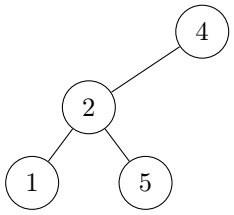
Input:

```
4
4 1 -1
2 2 3
1 -1 -1
5 -1 -1
```

Output:

```
INCORRECT
```

Explanation:



Node 5 is in the left subtree of the root, but it is bigger than the key 4 in the root.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `is_bst`

What to Do

Testing the binary search tree condition for each node and every other node in its subtree will be too slow. You should come up with a faster algorithm.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

3 Is it a binary search tree? Hard version.

Problem Introduction

In this problem you are going to solve the same problem as the previous one, but for a more general case, when binary search tree may contain equal keys.

Problem Description

Task. You are given a binary tree with integers as its keys. You need to test whether it is a correct binary search tree. Note that there can be duplicate integers in the tree, and this is allowed. The definition of the binary search tree in such case is the following: for any node of the tree, if its key is x, then for any node in its left subtree its key must be strictly less than x, and for any node in its right subtree its key must be greater than **or equal** to x. In other words, smaller elements are to the left, bigger elements are to the right, and duplicates are always to the right. You need to check whether the given binary tree structure satisfies this condition. You are guaranteed that the input contains a valid binary tree. That is, it is a tree, and each node has at most two children.

Input Format. The first line contains the number of vertices n. The vertices of the tree are numbered from 0 to $n - 1$. Vertex 0 is the root.

The next n lines contain information about vertices 0, 1, ..., $n-1$ in order. Each of these lines contains three integers key_i, $left_i$ and $right_i$ — key_i is the key of the i-th vertex, $left_i$ is the index of the left child of the i-th vertex, and $right_i$ is the index of the right child of the i-th vertex. If i doesn't have left or right child (or both), the corresponding $left_i$ or $right_i$ (or both) will be equal to -1.

Constraints. $0 \le n \le 10^5$; $-2^{31} \le key_i \le 2^{31} - 1$; $-1 \le left_i, right_i \le n - 1$. It is guaranteed that the input represents a valid binary tree. In particular, if $left_i \ne -1$ and $right_i \ne -1$, then $left_i \ne right_i$. Also, a vertex cannot be a child of two different vertices. Also, each vertex is a descendant of the root vertex. Note that the minimum and the maximum possible values of the 32-bit integer type are allowed to be keys in the tree — beware of integer overflow!

Output Format. If the given binary tree is a correct binary search tree (see the definition in the problem description), output one word "CORRECT" (without quotes). Otherwise, output one word "INCORRECT" (without quotes).

Time Limits.

language	`C`	`C++`	`Java`	`Python`	`C#`	`Haskell`	`JavaScript`	`Ruby`	`Scala`
time (sec)	2	2	3	10	3	4	10	10	6

Memory Limit. 512MB.

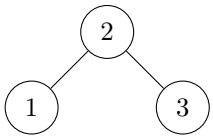
Sample 1.

Input:

```
3
2 1 2
1 -1 -1
3 -1 -1
```

Output:

```
CORRECT
```



Left child of the root has key 1, right child of the root has key 3, root has key 2, so everything to the left is smaller, everything to the right is bigger.

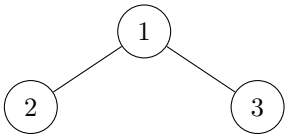
Sample 2.

Input:

```
3
1 1 2
2 -1 -1
3 -1 -1
```

Output:

```
INCORRECT
```



The left child of the root must have smaller key than the root.

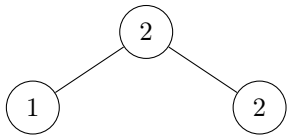
Sample 3.

Input:

```
3
2 1 2
1 -1 -1
2 -1 -1
```

Output:

```
CORRECT
```



Duplicate keys are allowed, and they should always be in the right subtree of the first duplicated element.

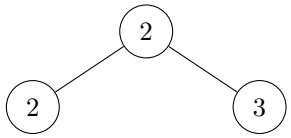
Sample 4.

Input:

```
3
2 1 2
2 -1 -1
3 -1 -1
```

Output:

```
INCORRECT
```



The key of the left child of the root must be strictly smaller than the key of the root.

Sample 5.

Input:

```
0
```

Output:

```
CORRECT
```

Empty tree is considered correct.

Sample 6.

Input:

```
1
2147483647 -1 -1
```

Output:

```
CORRECT
```

Explanation:

The maximum possible value of the 32-bit integer type is allowed as key in the tree.

Sample 7.

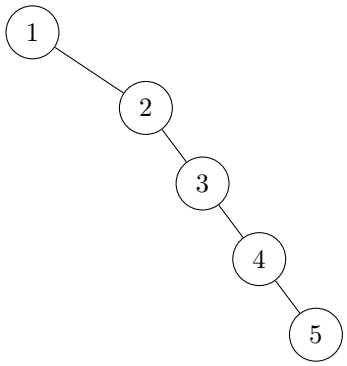
Input:

```
5
1 -1 1
2 -1 2
3 -1 3
4 -1 4
5 -1 -1
```

Output:

```
CORRECT
```

Explanation:



The tree doesn't have to be balanced. We only need to test whether it is a correct binary search tree, which the tree in this example is.

Sample 8.

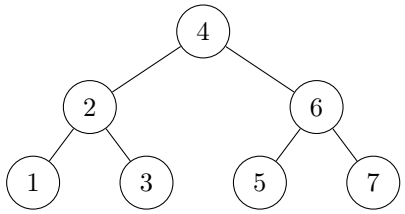
Input:

```
7
4 1 2
2 3 4
6 5 6
1 -1 -1
3 -1 -1
5 -1 -1
7 -1 -1
```

Output:

```
CORRECT
```

Explanation:



This is a full binary tree, and the property of the binary search tree is satisfied in every node.

Starter Files

The starter solutions for this problem read the input data from the standard input, pass it to a blank procedure, and then write the result to the standard output. You are supposed to implement your algorithm in this blank procedure if you are using `C++`, `Java`, or `Python3`. For other programming languages, you need to implement a solution from scratch. Filename: `is_bst_hard`

What to Do

Try to adapt the algorithm from the previous problem to the case when duplicate keys are allowed, and beware of integer overflow!

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

4 Set with range sums

Problem Introduction

In this problem, your goal is to implement a data structure to store a set of integers and quickly compute range sums.

Problem Description

Task. Implement a data structure that stores a set S of integers with the following allowed operations:

- **add**(i) — add integer i into the set S (if it was there already, the set doesn't change).
- **del**(i) — remove integer i from the set S (if there was no such element, nothing happens).
- **find**(i) — check whether i is in the set S or not.
- **sum**(l, r) — output the sum of all elements v in S such that $l \le v \le r$.

Input Format. Initially the set S is empty. The first line contains n — the number of operations. The next n lines contain operations. Each operation is one of the following:

- "+ i" — which means **add** some integer (not i, see below) to S,
- "- i" — which means **del** some integer (not i, see below)from S,
- "? i" — which means **find** some integer (not i, see below)in S,
- "s l r" — which means compute the **sum** of all elements of S within some range of values (not from l to r, see below).

However, to make sure that your solution can work in an online fashion, each request will actually depend on the result of the last **sum** request. Denote $M = 1\ 000\ 000\ 001$. At any moment, let x be the result of the last **sum** operation, or just 0 if there were no **sum** operations before. Then

- "+ i" means **add**$((i + x) \bmod M)$,
- "- i" means **del**$((i + x) \bmod M)$,
- "? i" means **find**$((i + x) \bmod M)$,
- "s l r" means **sum**$((l + x) \bmod M, (r + x) \bmod M)$.

Constraints. $1 \le n \le 100\ 000$; $0 \le i \le 10^9$.

Output Format. For each find request, just output "Found" or "Not found" (without quotes; note that the first letter is capital) depending on whether $(i + x) \bmod M$ is in S or not. For each **sum** query, output the sum of all the values v in S such that $((l+x) \bmod M) \le v \le ((r+x) \bmod M)$ (it is guaranteed that in all the tests $((l + x) \bmod M) \le ((r + x) \bmod M)$), where x is the result of the last **sum** operation or 0 if there was no previous **sum** operation.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	1	1	4	120	1.5	2	120	120	4

Memory Limit. 512MB.

Sample 1.

Input:

```
15
? 1
+ 1
? 1
+ 2
s 1 2
+ 1000000000
? 1000000000
- 1000000000
? 1000000000
s 999999999 1000000000
- 2
? 2
- 0
+ 9
s 0 9
```

Output:

```
Not found
Found
3
Found
Not found
1
Not found
10
```

Explanation:
For the first 5 queries, $x = 0$. For the next 5 queries, $x = 3$. For the next 5 queries, $x = 1$. The actual list of operations is:
find(1)
add(1)
find(1)
add(2)
sum$(1, 2) \to 3$
add(2)
find$(2) \to$ Found
del(2)
find$(2) \to$ Not found
sum$(1, 2) \to 1$
del(3)
find$(3) \to$ Not found
del(1)
add(10)
sum$(1, 10) \to 10$

Adding the same element twice doesn't change the set. Attempts to remove an element which is not in the set are ignored.

Sample 2.

Input:

```
5
? 0
+ 0
? 0
- 0
? 0
```

Output:

```
Not found
Found
Not found
```

First, 0 is not in the set. Then it is added to the set. Then it is removed from the set.

Sample 3.

Input:

```
5
+ 491572259
? 491572259
? 899375874
s 310971296 877523306
+ 352411209
```

Output:

```
Found
Not found
491572259
```

Explanation:
First, 491572259 is added to the set, then it is found there. Number 899375874 is not in the set. The only number in the set is now 491572259, and it is in the range between 310971296 and 877523306, so the sum of all numbers in this range is equal to 491572259.

Starter Files

The starter solutions in C++, Java and Python3 read the input, write the output, fully implement splay tree and show how to use its methods to solve this problem, but don't solve the whole problem. You need to finish the implementation. If you use other languages, you need to implement a solution from scratch.

Note that we strongly encourage you to use stress testing, max tests, testing for min and max values of each parameter according to the restrictions section and other testing techniques and advanced advice from this reading. If you're stuck for a long time, you can read ths forum thread to find out what other learners struggled with, how did they overcome their troubles and what tests did they come up with. If you're still stuck, you can read the hints in the next What to Do section mentioning some of the common problems and how to test for them, resolve some of them. Finally, if none of this worked, we included some of the trickier test cases in the starter files for this problem, so you can use them to debug your program if it fails on one of those tests in the grader. However, you will learn more if you pass this problem without looking at those test cases in the starter files.

What to Do

Use splay tree to efficiently store the set, add, delete and find elements. For each node in the tree, store additionally the sum of all the elements in the subtree of this node. Don't forget to update this sum each

time the tree changes. Use split operation to cut ranges from the tree. To get the sum of all the needed elements after split, just look at the sum stored in the root of the splitted tree. Don't forget to merge the trees back after the **sum** operation.

Some hints based on the problems some learners encountered with their solutions:

- Use the sum attribute to keep updated the sum in the subtree, don't compute the sum from scratch each time, otherwise it will work too slow.
- Don't forget to do splay after each operation with a splay tree.
- Don't forget to splay the node which was accessed last during the find operation.
- Don't forget to update the root variable after each operation with the tree, because splay operation changes root, but it doesn't change where your root variable is pointing in some of the starters.
- Don't forget to merge back after splitting the tree.
- When you detach a node from its parent, don't forget to detach pointers from both ends.
- Don't forget to update all the pointers correctly when merging the trees together.
- Test sum operation when there are no elements within the range.
- Test sum operation when all the elements are within the range.
- Beware of integer overflow.
- Don't forget to check for null when erasing.
- Test: Try adding nodes in the tree in such an order that the tree becomes very unbalanced. Play with this visualization to find out how to do it. Create a very big unbalanced tree. Then try searching for an element that is not in the tree many times.
- Test: add some elements and then remove all the elements from the tree.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

5 Rope

Problem Introduction

In this problem you will implement Rope — data structure that can store a string and efficiently cut a part (a substring) of this string and insert it in a different position. This data structure can be enhanced to become persistent — that is, to allow access to the previous versions of the string. These properties make it a suitable choice for storing the text in text editors.

This is a very advanced problem, harder than all the previous advanced problems in this course. Don't be upset if it doesn't crack. Congratulations to all the learners who are able to successfully pass this problem!

Problem Description

Task. You are given a string S and you have to process n queries. Each query is described by three integers i, j, k and means to cut substring $S[i..j]$ (i and j are 0-based) from the string and then insert it after the k-th symbol of the remaining string (if the symbols are numbered from 1). If $k = 0$, $S[i..j]$ is inserted in the beginning. See the examples for further clarification.

Input Format. The first line contains the initial string S.
The second line contains the number of queries q.
Next q lines contain triples of integers i, j, k.

Constraints. S contains only lowercase english letters. $1 \le |S| \le 300\,000$; $1 \le q \le 100\,000$; $0 \le i \le j \le n - 1$; $0 \le k \le n - (j - i + 1)$.

Output Format. Output the string after all q queries.

Time Limits.

language	C	C++	Java	Python	C#	Haskell	JavaScript	Ruby	Scala
time (sec)	3	3	6	120	4.5	6	120	120	12

Memory Limit. 512MB.

Sample 1.

Input:

```
hlelowrold
2
1 1 2
6 6 7
```

Output:

```
helloworld
```

Explanation:
hlelowrold → *hellowrold* → *helloworld*
When $i = j = 1$, $S[i..j] = l$, and it is inserted after the 2-nd symbol of the remaining string *helowrold*, which gives *hellowrold*. Then $i = j = 6$, so $S[i..j] = r$, and it is inserted after the 7-th symbol of the remaining string *hellowold*, which gives *helloworld*.

Sample 2.

Input:

```
abcdef
2
0 1 1
4 5 0
```

Output:

```
efcabd
```

$abcdef \to cabdef \to efcabd$

Starter Files

The starter solutions for C++ and Java in this problem read the input, implement a naive algorithm to cut and paste substrings and write the output. The starter solution for Python3 just reads the input and writes the output. You need to implement a data structure to make the operations with string very fast. If you use other languages, you need to implement the solution from scratch.

What to Do

Use splay tree to store the string. Use the split and merge methods of the splay tree to cut and paste substrings. Think what should be stored as the key in the splay tree. Try to find analogies with the ideas from this lecture.

Need Help?

Ask a question or see the questions asked by other learners at this forum thread.

6 Appendix

6.1 Compiler Flags

C (`gcc 5.2.1`). File extensions: `.c`. Flags:

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (`g++ 5.2.1`). File extensions: `.cc`, `.cpp`. Flags:

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

If your `C/C++` compiler does not recognize `-std=c++14` flag, try replacing it with `-std=c++0x` flag or compiling without this flag at all (all starter solutions can be compiled without it). On Linux and MacOS, you most probably have the required compiler. On Windows, you may use your favorite compiler or install, e.g., `cygwin`.

C# (`mono 3.2.8`). File extensions: `.cs`. Flags:

```
mcs
```

Go (`golang 1.12`). File extensions: `.go`. Flags

```
go
```

Haskell (`ghc 7.8.4`). File extensions: `.hs`. Flags:

```
ghc -O2
```

Java (`Open JDK 8`). File extensions: `.java`. Flags:

```
javac -encoding UTF-8
java -Xmx1024m
```

JavaScript (`Node v10.15.3`). File extensions: `.js`. No flags:

```
nodejs
```

Kotlin (`Kotlin 1.2.21`). File extensions: `.kt`. Flags:

```
kotlinc
java -Xmx1024m
```

Python 2 (`CPython 2.7`). File extensions: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python2"). No flags:

```
python2
```

Python 3 (`CPython 3.4`). File extensions: `.py3` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing "python3"). No flags:

```
python3
```

Ruby (`Ruby 2.1.5`). File extensions: `.rb`.

```
ruby
```

Rust (`Rust 1.28.0`). File extensions: `.rs`.

```
rustc
```

Scala (`Scala 2.11.6`). File extensions: `.scala`.

```
scalac
```

6.2 Frequently Asked Questions

Why My Submission Is Not Graded?

You need to create a submission and upload the *source file* (rather than the executable file) of your solution. Make sure that after uploading the file with your solution you press the blue "Submit" button at the bottom. After that, the grading starts, and the submission being graded is enclosed in an orange rectangle. After the testing is finished, the rectangle disappears, and the results of the testing of all problems are shown.

What Are the Possible Grading Outcomes?

There are only two outcomes: "pass" or "no pass." To pass, your program must return a correct answer on all the test cases we prepared for you, and do so under the time and memory constraints specified in the problem statement. If your solution passes, you get the corresponding feedback "Good job!" and get a point for the problem. Your solution fails if it either crashes, returns an incorrect answer, works for too long, or uses too much memory for some test case. The feedback will contain the index of the first test case on which your solution failed and the total number of test cases in the system. The tests for the problem are numbered from 1 to the total number of test cases for the problem, and the program is always tested on all the tests in the order from the first test to the test with the largest number.

Here are the possible outcomes:

- `Good job!` Hurrah! Your solution passed, and you get a point!
- `Wrong answer.` Your solution outputs incorrect answer for some test case. Check that you consider all the cases correctly, avoid integer overflow, output the required white spaces, output the floating point numbers with the required precision, don't output anything in addition to what you are asked to output in the output specification of the problem statement.
- `Time limit exceeded.` Your solution worked longer than the allowed time limit for some test case. Check again the running time of your implementation. Test your program locally on the test of maximum size specified in the problem statement and check how long it works. Check that your program doesn't wait for some input from the user which makes it to wait forever.
- `Memory limit exceeded.` Your solution used more than the allowed memory limit for some test case. Estimate the amount of memory that your program is going to use in the worst case and check that it does not exceed the memory limit. Check that your data structures fit into the memory limit. Check that you don't create large arrays or lists or vectors consisting of empty arrays or empty strings, since those in some cases still eat up memory. Test your program locally on the tests of maximum size specified in the problem statement and look at its memory consumption in the system.
- `Cannot check answer. Perhaps the output format is wrong.` This happens when you output something different than expected. For example, when you are required to output either "Yes" or "No", but instead output 1 or 0. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (please follow the exact output format specified in the problem statement). Maybe your program doesn't output anything, because it crashes.
- `Unknown signal 6 (or 7, or 8, or 11, or some other).` This happens when your program crashes. It can be because of a division by zero, accessing memory outside of the array bounds, using uninitialized variables, overly deep recursion that triggers a stack overflow, sorting with a contradictory comparator, removing elements from an empty data structure, trying to allocate too much memory, and many other reasons. Look at your code and think about all those possibilities. Make sure that you use the same compiler and the same compiler flags as we do.
- `Internal error: exception...` Most probably, you submitted a compiled program instead of a source code.
- `Grading failed.` Something wrong happened with the system. Report this through Coursera or edX Help Center.

May I Post My Solution at the Forum?

Please do not post any solutions at the forum or anywhere on the web, even if a solution does not pass the tests (as in this case you are still revealing parts of a correct solution). Our students follow the Honor Code: "I will not make solutions to homework, quizzes, exams, projects, and other assignments available to anyone else (except to the extent an assignment explicitly permits sharing solutions)."

Do I Learn by Trying to Fix My Solution?

My implementation always fails in the grader, though I already tested and stress tested it a lot. Would not it be better if you gave me a solution to this problem or at least the test cases that you use? I will then be able to fix my code and will learn how to avoid making mistakes. Otherwise, I do not feel that I learn anything from solving this problem. I am just stuck.

First of all, learning from your mistakes is one of the best ways to learn.

The process of trying to invent new test cases that might fail your program is difficult but is often enlightening. Thinking about properties of your program makes you understand what happens inside your program and in the general algorithm you're studying much more.

Also, it is important to be able to find a bug in your implementation without knowing a test case and without having a reference solution, just like in real life. Assume that you designed an application and an annoyed user reports that it crashed. Most probably, the user will not tell you the exact sequence of operations that led to a crash. Moreover, there will be no reference application. Hence, it is important to learn how to find a bug in your implementation yourself, without a magic oracle giving you either a test case that your program fails or a reference solution. We encourage you to use programming assignments in this class as a way of practicing this important skill.

If you have already tested your program on all corner cases you can imagine, constructed a set of manual test cases, applied stress testing, etc, but your program still fails, try to ask for help on the forum. We encourage you to do this by first explaining what kind of corner cases you have already considered (it may happen that by writing such a post you will realize that you missed some corner cases!), and only afterwards asking other learners to give you more ideas for tests cases.