

Programming Assignment 3:

Divide-and-Conquer

Revision: February 17, 2016

Introduction

In this programming assignment, you will be practicing implementing divide-and-conquer solutions.

Learning Outcomes

Upon completing this programming assignment you will be able to:

1. Apply the divide-and-conquer technique to solve various computational problems efficiently. This will usually require you to design an algorithm that solves a problem by splitting it into several disjoint subproblems, solving them recursively, and then combining their results to get an answer for the initial problem.
2. Implement the binary search algorithm that is used to search keys in sorted data.
3. Implement the quick sort algorithm which is one of the fastest and most commonly used in practice sorting algorithms.
4. Design and implement efficient divide-and-conquer algorithms for novel computational problems.

Passing Criteria: 2 out of 5

Passing this programming assignment requires passing at least 2 out of 5 code problems from this assignment. In turn, passing a code problem 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 Problem: Binary Search	3
2 Problem: Majority Element	4
3 Problem: Sorting: 3-Way Partition	6
4 Advanced Problem: Number of Inversions	7
5 Advanced Problem: Points and Segments	8

6	General Instructions and Recommendations on Solving Algorithmic Problems	10
6.1	Reading the Problem Statement	10
6.2	Designing an Algorithm	10
6.3	Implementing Your Algorithm	10
6.4	Compiling Your Program	10
6.5	Testing Your Program	11
6.6	Submitting Your Program to the Grading System	11
6.7	Debugging and Stress Testing Your Program	12
7	Frequently Asked Questions	13
7.1	I submit the program, but nothing happens. Why?	13
7.2	I submit the solution only for one problem, but all the problems in the assignment are graded. Why?	13
7.3	What are the possible grading outcomes, and how to read them?	13
7.4	How to understand why my program fails and to fix it?	14
7.5	Why do you hide the test on which my program fails?	14
7.6	My solution does not pass the tests? May I post it in the forum and ask for a help?	15

1 Problem: Binary Search

Problem Introduction

In this problem, you will implement the binary search algorithm that allows searching very efficiently (even huge) lists provided that the list is sorted.

Problem Description

Task. The goal in this code problem is to implement the binary search algorithm.

Input Format. The first line of the input contains an integer n and a sequence $a_0 < a_1 < \dots < a_{n-1}$ of n pairwise distinct positive integers in increasing order. The next line contains an integer k and k positive integers b_0, b_1, \dots, b_{k-1} .

Constraints. $1 \leq n, k \leq 10^5$; $1 \leq a_i \leq 10^9$ for all $0 \leq i < n$; $1 \leq b_j \leq 10^9$ for all $0 \leq j < k$;

Output Format. For all i from 0 to $k - 1$, output an index $0 \leq j \leq n - 1$ such that $a_j = b_i$ or -1 if there is no such index.

Time Limits. C: 2.0 sec, C++: 2.0 sec, Java: 3.0 sec, Python: 10.0 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
5 1 5 8 12 13
5 8 1 23 1 11
```

Output:

```
2 0 -1 0 -1
```

Starter Files

The starter files contain an implementation of the linear search algorithm that just scans an array to find a given key. Try submitting a starter file to the grader to ensure that linear search indeed takes too long.

What To Do

Implement the binary search algorithm and replace the call to the linear search with a call to the binary search algorithm.

2 Problem: Majority Element

Problem Introduction

An element of a sequence of length n is called a majority element if it appears in the sequence strictly more than $n/2$ times.

Problem Description

Task. The goal in this code problem is to check whether an input sequence contains a majority element.

Input Format. The first line contains an integer n , the next one contains a sequence of n integers a_0, a_1, \dots, a_{n-1} .

Constraints. $1 \leq n \leq 10^5$; $-10^9 \leq a_i \leq 10^9$ for all $0 \leq i < n$.

Output Format. Output 1 if the sequence contains a majority element and 0 otherwise.

Time Limits. C: 1 sec, C++: 1 sec, Java: 1.5 sec, Python: 5 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
5
2 3 9 2 2
```

Output:

```
1
```

Sample 2.

Input:

```
4
1 2 3 4
```

Output:

```
0
```

Sample 3.

Input:

```
4
1 2 3 1
```

Output:

```
0
```

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 Python; for C, you need to implement a solution from scratch).

What To Do

As you might have already guessed, this problem can be solved by the divide-and-conquer algorithm in time $O(n \log n)$. Indeed, if a sequence of length n contains a majority element, then the same element is also a majority element for one of its halves. Thus, to solve this problem you first split a given sequence into halves and make two recursive calls. Do you see how to combine the results of two recursive calls?

It is interesting to note that this problem can also be solved in $O(n)$ time by a more advanced (non-divide and conquer) algorithm that just scans the given sequence twice.

3 Problem: Sorting: 3-Way Partition

Problem Introduction

The goal in this problem is to redesign a given implementation of the randomized quick sort algorithm so that it works fast even on sequences containing many equal elements.

Problem Description

Task. To force the given implementation of the quick sort algorithm to efficiently process sequences with few unique elements, your goal is replace a 2-way partition with a 3-way partition. That is, your new partition procedure should partition the array into three parts: $< x$ part, $= x$ part, and $> x$ part.

Input Format. The first line of the input contains an integer n . The next line contains a sequence of n integers a_0, a_1, \dots, a_{n-1} .

Constraints. $1 \leq n \leq 10^5$; $1 \leq a_i \leq 10^9$ for all $0 \leq i < n$.

Output Format. Output this sequence sorted in non-decreasing order.

Time Limits. C: 2 sec, C++: 2 sec, Java: 3 sec, Python: 10 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
5
2 3 9 2 2
```

Output:

```
2 2 2 3 9
```

Starter Files

In the starter files, you are given an implementation of the randomized quick sort algorithm using a 2-way partition procedure. This procedure partitions the given array into two parts with respect to a pivot x : $\leq x$ part and $> x$ part. As discussed in the video lectures, such an implementation has $\Theta(n^2)$ running time on sequences containing a single unique element. Indeed, the partition procedure in this case splits the array into two parts, one of which is empty and the other one contains $n - 1$ elements. It spends cn time on this. The overall running time is then

$$cn + c(n-1) + c(n-2) + \dots = \Theta(n^2).$$

What To Do

Implement a 3-way partition procedure and then replace a call to the 2-way partition procedure by a call to the 3-way partition procedure.

4 Advanced Problem: Number of Inversions

(Recall that advanced problems are not covered in the video lectures and require additional ideas to be solved. We therefore strongly recommend you start solving these problems only when you are done with the basic problems.)

Problem Introduction

An inversion of a sequence a_0, a_1, \dots, a_{n-1} is a pair of indices $0 \leq i < j < n$ such that $a_i > a_j$. The number of inversions of a sequence in some sense measures how close the sequence is to being sorted. For example, a sorted (in non-descending order) sequence contains no inversions at all, while in a sequence sorted in descending order any two elements constitute an inversion (for a total of $n(n+1)/2$ inversions).

Problem Description

Task. The goal in this problem is to count the number of inversions of a given sequence.

Input Format. The first line contains an integer n , the next one contains a sequence of integers a_0, a_1, \dots, a_{n-1} .

Constraints. $1 \leq n \leq 10^5$, $1 \leq a_i \leq 10^9$ for all $0 \leq i < n$.

Output Format. Output the number of inversions in the sequence.

Time Limits. C: 3 sec, C++: 3 sec, Java: 4.5 sec, Python: 15 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
5
2 3 9 2 9
```

Output:

```
2
```

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 Python; for C, you need to implement a solution from scratch).

What To Do

This problem can be solved by modifying the merge sort algorithm. For this, we change both the **Merge** and **MergeSort** procedures as follows:

- **Merge**(B, C) returns the resulting sorted array and the number of pairs (b, c) such that $b \in B$, $c \in C$, and $b > c$;
- **MergeSort**(A) returns a sorted array A and the number of inversions in A .

5 Advanced Problem: Points and Segments

(Recall that advanced problems are not covered in the video lectures and require additional ideas to be solved. We therefore strongly recommend you start solving these problems only when you are done with the basic problems.)

Problem Introduction

The goal in this problem is given a set of segments on a line and a set of points on a line, to count, for each point, the number of segments which contain it.

Problem Description

Task. In this problem you are given a set of points on a line and a set of segments on a line. The goal is to compute, for each point, the number of segments that contain this point.

Input Format. The first line contains two non-negative integers s and p defining the number of segments and the number of points on a line, respectively. The next s lines contain two integers a_i, b_i defining the i -th segment $[a_i, b_i]$. The next line contains p integers defining points x_1, x_2, \dots, x_p .

Constraints. $1 \leq s, p \leq 50000$; $-10^8 \leq a_i \leq b_i \leq 10^8$ for all $0 \leq i < s$; $-10^8 \leq x_j \leq 10^8$ for all $0 \leq j < p$.

Output Format. Output p non-negative integers k_0, k_1, \dots, k_{p-1} where k_i is the number of segments which contain x_i . More formally,

$$k_i = |\{j: a_j \leq x_i \leq b_j\}|.$$

Time Limits. C: 3.0 sec, C++: 3.0 sec, Java: 4.5 sec, Python: 15.0 sec.

Memory Limit. 64Mb.

Sample 1.

Input:

```
2 3
0 5
7 10
1 6 11
```

Output:

```
1 0 0
```

Explanation:

Here, we have two segments and three points. The first point lies only in the first segment while the remaining two points are outside of all the given segments.

Sample 2.

Input:

```
1 3
-10 10
-100 100 0
```

Output:

```
0 0 1
```

Sample 3.

Input:


```
3 2
0 5
-3 2
7 10
1 6
```

Output:

```
2 0
```

Starter Files

The starter files for this problem contain an implementation of the following naive algorithm: for each point, scan the list of all segments to check how many of them contain this point. The running time of this algorithm is $O(sp)$ making it too slow to pass the given time limit.

What To Do

As you might have already guessed, your goal is to first sort the given segments somehow.

6 General Instructions and Recommendations on Solving Algorithmic Problems

Your main goal in an algorithmic problem is to implement a program that solves a given computational problem in just few seconds even on massive datasets. Your program should read a dataset from the standard input and write an answer to the standard output.

Below we provide general instructions and recommendations on solving such problems. Before reading them, go through readings and screencasts in the first module that show a step by step process of solving two algorithmic problems: [link](#).

6.1 Reading the Problem Statement

You start by reading the problem statement that contains the description of a particular computational task as well as time and memory limits your solution should fit in, and one or two sample tests. In some problems your goal is just to implement carefully an algorithm covered in the lectures, while in some other problems you first need to come up with an algorithm yourself.

6.2 Designing an Algorithm

If your goal is to design an algorithm yourself, one of the things it is important to realize is the expected running time of your algorithm. Usually, you can guess it from the problem statement (specifically, from the subsection called constraints) as follows. Modern computers perform roughly 10^8 – 10^9 operations per second. So, if the maximum size of a dataset in the problem description is $n = 10^5$, then most probably an algorithm with quadratic running time is not going to fit into time limit (since for $n = 10^5$, $n^2 = 10^{10}$) while a solution with running time $O(n \log n)$ will fit. However, an $O(n^2)$ solution will fit if n is up to $10^3 = 1000$, and if n is at most 100, even $O(n^3)$ solutions will fit. In some cases, the problem is so hard that we do not know a polynomial solution. But for n up to 18, a solution with $O(2^n n^2)$ running time will probably fit into the time limit.

To design an algorithm with the expected running time, you will of course need to use the ideas covered in the lectures. Also, make sure to carefully go through sample tests in the problem description.

6.3 Implementing Your Algorithm

When you have an algorithm in mind, you start implementing it. Currently, you can use the following programming languages to implement a solution to a problem: C, C++, Java, Python2, Python3. For all problems, we will be providing starter solutions for C++, Java, and Python3. If you are going to use one of these programming languages, use these starter files. If you are going to use C or Python2, you need to implement a solution from scratch.

6.4 Compiling Your Program

For solving programming assignments, you can use any of the following programming languages: C, C++, Java, Python2, or Python3. However, we will only be providing starter solution files for C++, Java, and Python3. The programming language of your submission is detected automatically, based on the extension of your submission.

Your solution will be compiled as follows. We recommend that when testing your solution locally, you use the same compiler flags for compiling. This will increase the chances that your program behaves in the same way on your machine and on the testing machine (note that a buggy program may behave differently when compiled by different compilers, or even by the same compiler with different flags).

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

```
gcc -pipe -O2 -std=c11
```

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

```
g++ -pipe -O2 -std=c++11
```

If your C/C++ compiler does not recognize `-std=c++11` 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`.

- Java (Open JDK 7). File extensions: `.java`. No flags:

```
javac
```

- 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: `.py2` or `.py` (a file ending in `.py` needs to have a first line which is a comment containing “python3”). No flags:

```
python3
```

6.5 Testing Your Program

When your program is ready, you start testing it. It makes sense to start with small datasets — for example, sample tests provided in the problem description. Ensure that your program produces a correct result.

You then proceed to checking how long does it take your program to process a massive dataset. For this, it makes sense to implement your algorithm as a function like `solve(dataset)` and then implement an additional procedure `generate()` that produces a large dataset. For example, if an input to a problem is a sequence of integers of length $1 \leq n \leq 10^5$, then generate a sequence of length exactly 10^5 , pass it to your `solve()` function, and ensure that the program outputs the result quickly.

Also, check the boundary values. Ensure that your program processes correctly sequences of size $n = 1, 2, 10^5$. If a sequence of integers from 0 to, say, 10^6 is given as an input, check how your program behaves when it is given a sequence $0, 0, \dots, 0$ or a sequence $10^6, 10^6, \dots, 10^6$. Check also on randomly generated data. For each such test check that you program produces a correct result (or at least a reasonably looking result).

In the end, we encourage you to stress test your program to make sure it passes in the system at the first attempt. See the readings and screencasts from the first week to learn about testing and stress testing: [link](#).

6.6 Submitting Your Program to the Grading System

When you are done with testing, you submit your program to the grading system. For this, you go the submission page, create a new submission, and upload a file with your program. The grading system then compiles your program (detecting the programming language based on your file extension, see Subsection 6.4) and runs it on a set of carefully constructed tests to check that your program always outputs a correct result and that it always fits into the given time and memory limits. The grading usually takes no more than a minute, but in rare cases when the servers are overloaded it might take longer. Please be patient. You can safely leave the page when your solution is uploaded.

As a result, you get a feedback message from the grading system. The feedback message that you will love to see is: **Good job!** This means that your program has passed all the tests. On the other hand, the three messages **Wrong answer**, **Time limit exceeded**, **Memory limit exceeded** notify you that your program failed due to one these three reasons. Note that the grader will not show you the actual test you program have failed on (though it does show you the test if your program have failed on one of the first few tests; this is done to help you to get the input/output format right).

6.7 Debugging and Stress Testing Your Program

If your program failed, you will need to debug it. Most probably, you didn't follow some of our suggestions from the section [6.5](#). See the readings and screencasts from the first week to learn about debugging your program: [link](#).

You are almost guaranteed to find a bug in your program using stress testing, because the way these programming assignments and tests for them are prepared follows the same process: small manual tests, tests for edge cases, tests for large numbers and integer overflow, big tests for time limit and memory limit checking, random test generation. Also, implementation of wrong solutions which we expect to see and stress testing against them to add tests specifically against those wrong solutions.

Go ahead, and we hope you pass the assignment soon!

7 Frequently Asked Questions

7.1 I submit the program, but nothing happens. Why?

You need to create submission and upload the file with your solution in one of the programming languages C, C++, Java, or Python (see Subsections 6.3 and 6.4). Make sure that after uploading the file with your solution you press on the blue “Submit” button in 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 is shown to you.

7.2 I submit the solution only for one problem, but all the problems in the assignment are graded. Why?

Each time you submit any solution, the last uploaded solution for each problem is tested. Don’t worry: this doesn’t affect your score even if the submissions for the other problems are wrong. As soon as you pass the sufficient number of problems in the assignment (see in the pdf with instructions), you pass the assignment. After that, you can improve your result if you successfully pass more problems from the assignment. We recommend working on one problem at a time, checking whether your solution for any given problem passes in the system as soon as you are confident in it. However, it is better to test it first, please refer to the reading about stress testing: [link](#).

7.3 What are the possible grading outcomes, and how to read them?

Your solution may either pass or not. To pass, it must work without crashing and return the correct answers on all the test cases we prepared for you, and do so under the time limit and memory limit constraints specified in the problem statement. If your solution passes, you get the corresponding feedback “Good job!” and get a point for the problem. If your solution fails, it can be because it crashes, returns wrong answer, works for too long or uses too much memory for some test case. The feedback will contain the number of the test case on which your solution fails 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 test number 1 to the test with the biggest number.

Here are the possible outcomes:

Good job! Hurrah! Your solution passed, and you get a point!

Wrong answer. Your solution has output incorrect answer for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won’t know the input, the output, and the correct answer. Check that you consider all the cases correctly, avoid integer overflow, output the required white space, 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. See this reading on testing: [link](#).

Time limit exceeded. Your solution worked longer than the allowed time limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1, you will also see the input data, the output of your program and the correct answer. Otherwise, you won’t know the input, the output and the correct answer. Check again that your algorithm has good enough running time estimate. Test your program locally on the test of maximum size allowed by the problem statement and see how long it works. Check that your program doesn’t wait for some input from the user which makes it to wait forever. See this reading on testing: [link](#).

Memory limit exceeded. Your solution used more than the allowed memory limit for some test case. If it is a sample test case from the problem statement, or if you are solving Programming Assignment 1,

you will also see the input data, the output of your program and the correct answer. Otherwise, you won't know the input, the output and the correct answer. Estimate the amount of memory that your program is going to use in the worst case and check that it is less than the memory limit. Check that you don't create too large arrays or data structures. 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 test of maximum size allowed by the problem statement and look at its memory consumption in the system.

Cannot check answer. Perhaps output format is wrong. This happens when you output something completely different than expected. For example, you are required to output word "Yes" or "No", but you output number 1 or 0, or vice versa. Or your program has empty output. Or your program outputs not only the correct answer, but also some additional information (this is not allowed, so please follow exactly the 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 division by zero, accessing memory outside of the array bounds, using uninitialized variables, too deep recursion that triggers stack overflow, sorting with 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 compilers and the same compiler options as we do. Try different testing techniques from this reading: [link](#).

Grading failed. Something very wrong happened with the system. Contact Coursera for help or write in the forums to let us know.

7.4 How to understand why my program fails and to fix it?

If your program works incorrectly, it gets a feedback from the grader. For the Programming Assignment 1, when your solution fails, you will see the input data, the correct answer and the output of your program in case it didn't crash, finished under the time limit and memory limit constraints. If the program crashed, worked too long or used too much memory, the system stops it, so you won't see the output of your program or will see just part of the whole output. We show you all this information so that you get used to the algorithmic problems in general and get some experience debugging your programs while knowing exactly on which tests they fail.

However, in the following Programming Assignments throughout the Specialization you will only get so much information for the test cases from the problem statement. For the next tests you will only get the result: passed, time limit exceeded, memory limit exceeded, wrong answer, wrong output format or some form of crash. We hide the test cases, because it is crucial for you to learn to test and fix your program even without knowing exactly the test on which it fails. In the real life, often there will be no or only partial information about the failure of your program or service. You will need to find the failing test case yourself. Stress testing is one powerful technique that allows you to do that. You should apply it after using the other testing techniques covered in this reading.

7.5 Why do you hide the test on which my program fails?

Often beginner programmers think by default that their programs work. Experienced programmers know, however, that their programs almost never work initially. Everyone who wants to become a better programmer needs to go through this realization.

When you are sure that your program works by default, you just throw a few random test cases against it, and if the answers look reasonable, you consider your work done. However, mostly this is not enough. To make one's programs work, one must test them really well. Sometimes, the programs still don't work although you tried really hard to test them, and you need to be both skilled and creative to fix your bugs. Solutions

to algorithmic problems are one of the hardest to implement correctly. That's why in this Specialization you will gain this important experience which will be invaluable in the future when you write programs which you really need to get right.

It is crucial for you to learn to test and fix your programs yourself. In the real life, often there will be no or only partial information about the failure of your program or service. Still, you will have to reproduce the failure to fix it (or just guess what it is, but that's rare, and you will still need to reproduce the failure to make sure you have really fixed it). When you solve algorithmic problems, it is very frequent to make subtle mistakes. That's why you should apply the testing techniques described in this reading to find the failing test case and fix your program.

7.6 My solution does not pass the tests? May I post it in the forum and ask for a help?

No, please do not post any solutions in 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). Recall the third item of the Coursera 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). This includes both solutions written by me, as well as any solutions provided by the course staff or others" ([link](#)).