Programming Assignment 4: Divide-and-Conquer

Revision: May 28, 2020

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. Design and implement efficient algorithms for the following computational problems:
   (a) searching a sorted data for a key;
   (b) finding a majority element in a data;
   (c) improving the quick sort algorithm;
   (d) checking how close a data is to being sorted;
   (e) organizing a lottery;
   (f) finding the closest pair of points.

Passing Criteria: 3 out of 6

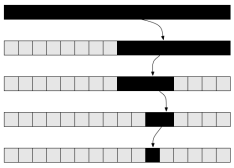
Passing this programming assignment requires passing at least 3 out of 6 programming challenges from this assignment. In turn, passing a programming challenge requires implementing a solution that passes all the tests for this problem in the grader and does so under the time and memory limits specified in the problem statement.

Contents

1 Binary 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 $a_0 < a_1 < \ldots < a_{n-1}$ of n pairwise distinct positive integers in increasing order. The next line contains an integer k and k positive integers $b_0, b_1, \ldots, b_{k-1}$.

Constraints. $1 \le k \le 10^5$; $1 \le n \le 3 \cdot 10^4$; $1 \le a_i \le 10^9$ for all $0 \le i < n$; $1 \le b_j \le 10^9$ for all $0 \le j < k$;

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

Sample 1.

Input:

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

Output:

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

In this sample, we are given an increasing sequence $a_0 = 1$, $a_1 = 5$, $a_2 = 8$, $a_3 = 12$, $a_4 = 13$ of length five and five keys to search: $8, 1, 23, 1, 11$. We see that $a_2 = 8$ and $a_0 = 1$, but the keys 23 and 11 do not appear in the sequence a. For this reason, we output a sequence $2, 0, -1, 0, -1$.

Need Help?

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

2 Majority Element

Problem Introduction

Majority rule is a decision rule that selects the alternative which has a majority, that is, more than half the votes.
Given a sequence of elements $a_1, a_2, \ldots, a_n$, you would like to check whether it contains an element that appears more than $n/2$ times. A naive way to do this is the following.

```
MajorityElement(a_1, a_2, ..., a_n):
for i from 1 to n:
  currentElement ← a_i
  count ← 0
  for j from 1 to n:
    if a_j = currentElement:
      count ← count + 1
  if count > n/2:
    return a_i
return "no majority element"
```

The running time of this algorithm is quadratic. Your goal is to use the divide-and-conquer technique to design an $O(n \log n)$ algorithm.

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 non-negative integers $a_0, a_1, \ldots, a_{n-1}$.

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

Output Format. Output 1 if the sequence contains an element that appears strictly more than $n/2$ times, and 0 otherwise.

Sample 1.

Input:

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

Output:

```
1
```

2 is the majority element.

Sample 2.

Input:

```
4
1 2 3 4
```

Output:

```
0
```

There is no majority element in this sequence.

Sample 3.

Input:

```
4
1 2 3 1
```

Output:

```
0
```

This sequence also does not have a majority element (note that the element 1 appears twice and hence is not a majority element).

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.

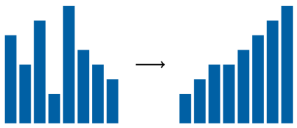
Need Help?

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

3 Improving Quick Sort

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 $a_0, a_1, \ldots, a_{n-1}$.

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

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

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: $\le x$ 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

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

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.

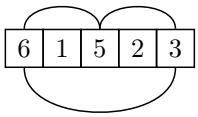
Need Help?

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

4 Number of Inversions

Problem Introduction

An inversion of a sequence $a_0, a_1, \ldots, a_{n-1}$ is a pair of indices $0 \le i < j < n$ 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 $a_0, a_1, \ldots, a_{n-1}$.

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

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

Sample 1.

Input:

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

Output:

```
2
```

The two inversions here are $(1, 3)$ $(a_1 = 3 > 2 = a_3)$ and $(2, 3)$ $(a_2 = 9 > 2 = a_3)$.

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.

Need Help?

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

5 Organizing a Lottery

Problem Introduction

You are organizing an online lottery. To participate, a person bets on a single integer. You then draw several ranges of consecutive integers at random. A participant's payoff then is proportional to the number of ranges that contain the participant's number minus the number of ranges that does not contain it. You need an efficient algorithm for computing the payoffs for all participants. A naive way to do this is to simply scan, for all participants, the list of all ranges. However, you lottery is very popular: you have thousands of participants and thousands of ranges. For this reason, you cannot afford a slow naive algorithm.

Problem Description

Task. 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 $x_1, x_2, \ldots, x_p$.

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

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

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

Sample 1.

Input:

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

Output:

```
1 0 0
```

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
```

Need Help?

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

Solution

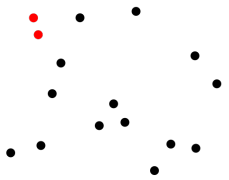
A detailed solution (with Python code) for this challenge is covered in the companion MOOCBook. We strongly encourage you to do your best to solve the challenge yourself before looking into the book! There are at least three good reasons for this.

- By solving this challenge, you practice solving algorithmic problems similar to those given at technical interviews.
- The satisfaction and self confidence that you get when passing the grader is priceless =)
- Even if you fail to pass the grader yourself, the time will not be lost as you will better understand the solution from the book and better appreciate the beauty of the underlying ideas.

6 Closest Points

Problem Introduction

In this problem, your goal is to find the closest pair of points among the given n points. This is a basic primitive in computational geometry having applications in, for example, graphics, computer vision, traffic-control systems.



Problem Description

Task. Given n points on a plane, find the smallest distance between a pair of two (different) points. Recall that the distance between points (x_1, y_1) and (x_2, y_2) is equal to $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Input Format. The first line contains the number n of points. Each of the following n lines defines a point (x_i, y_i).

Constraints. $2 \le n \le 10^5$; $-10^9 \le x_i, y_i \le 10^9$ are integers.

Output Format. Output the minimum distance. The absolute value of the difference between the answer of your program and the optimal value should be at most 10^{-3}. To ensure this, output your answer with at least four digits after the decimal point (otherwise your answer, while being computed correctly, can turn out to be wrong because of rounding issues).

Sample 1.

Input:

```
2
0 0
3 4
```

Output:

```
5.0
```

There are only two points here. The distance between them is 5.

Sample 2.

Input:

```
4
7 7
1 100
4 8
7 7
```

Output:

```
0.0
```

There are two coinciding points among the four given points. Thus, the minimum distance is zero.

Sample 3.

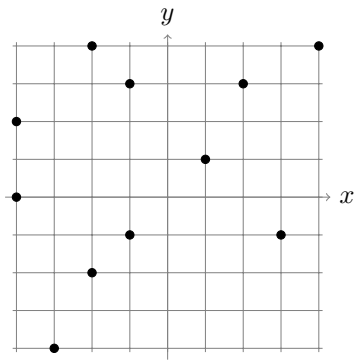
Input:

```
11
4 4
-2 -2
-3 -4
-1 3
2 3
-4 0
1 1
-1 -1
3 -1
-4 2
-2 4
```

Output:

```
1.414213
```

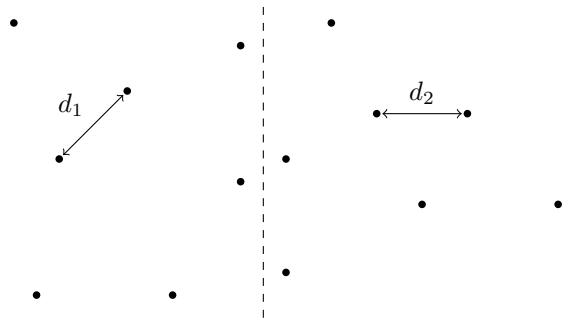
The smallest distance is $\sqrt{2}$. There are two pairs of points at this distance: $(-1,-1)$ and $(-2,-2)$; $(-2,4)$ and $(-1,3)$.



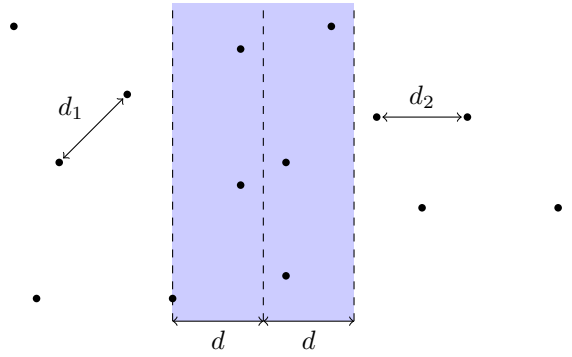
What To Do

This computational geometry problem has many applications in computer graphics and vision. A naive algorithm with quadratic running time iterates through all pairs of points to find the closest pair. Your goal is to design an $O(n \log n)$ time divide and conquer algorithm.

To solve this problem in time $O(n \log n)$, let's first split the given n points by an appropriately chosen vertical line into two halves S_1 and S_2 of size $\frac{n}{2}$ (assume for simplicity that all x-coordinates of the input points are different). By making two recursive calls for the sets S_1 and S_2, we find the minimum distances d_1 and d_2 in these subsets. Let $d = \min\{d_1, d_2\}$.

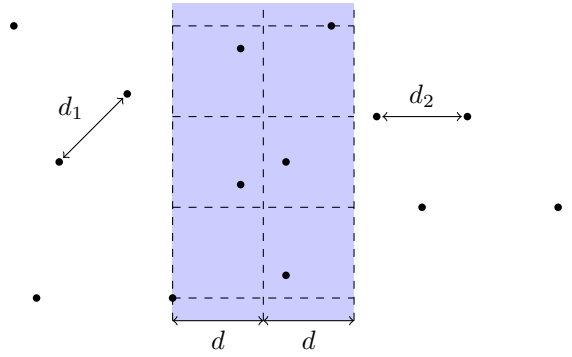


It remains to check whether there exist points $p_1 \in S_1$ and $p_2 \in S_2$ such that the distance between them is smaller than d. We cannot afford to check all possible such pairs since there are $\frac{n}{2} \cdot \frac{n}{2} = \Theta(n^2)$ of them. To check this faster, we first discard all points from S_1 and S_2 whose x-distance to the middle line is greater than d. That is, we focus on the following strip:



Stop and think: Why can we narrow the search to this strip? Now, let's sort the points of the strip by their y-coordinates and denote the resulting sorted list by $P = [p_1, \ldots, p_k]$. It turns out that if $|i - j| > 7$, then the distance between points p_i and p_j is greater than d for sure. This follows from the Exercise Break below.

Exercise break: Partition the strip into $d \times d$ squares as shown below and show that each such square contains at most four input points.



This results in the following algorithm. We first sort the given n points by their x-coordinates and then split the resulting sorted list into two halves S_1 and S_2 of size $\frac{n}{2}$. By making a recursive call for each of the sets S_1 and S_2, we find the minimum distances d_1 and d_2 in them. Let $d = \min\{d_1, d_2\}$. However, we are not done yet as we also need to find the minimum distance between points from different sets (i.e, a point from S_1 and a point from S_2) and check whether it is smaller than d. To perform such a check, we filter the initial point set and keep only those points whose x-distance to the middle line does not exceed d. Afterwards, we sort the set of points in the resulting strip by their y-coordinates and scan the resulting list of points. For each point, we compute its distance to the seven subsequent points in this list and compute d', the minimum distance that we encountered during this scan. Afterwards, we return $\min\{d, d'\}$.

The running time of the algorithm satisfies the recurrence relation

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n \log n)\,.$$

The $O(n \log n)$ term comes from sorting the points in the strip by their y-coordinates at every iteration.

Exercise break: Prove that $T(n) = O(n \log^2 n)$ by analyzing the recursion tree of the algorithm.

Exercise break: Show how to bring the running time down to $O(n \log n)$ by avoiding sorting at each recursive call.

Need Help?

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

7 Appendix

7.1 Compiler Flags

C (`gcc 7.4.0`). File extensions: `.c`. Flags:

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

C++ (`g++ 7.4.0`). 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 4.6.2`). File extensions: `.cs`. Flags:

```
mcs
```

Go (`golang 1.13.4`). File extensions: `.go`. Flags

```
go
```

Haskell (`ghc 8.0.2`). File extensions: `.hs`. Flags:

```
ghc -O2
```

Java (`OpenJDK 1.8.0_232`). File extensions: `.java`. Flags:

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

JavaScript (`NodeJS 12.14.0`). File extensions: `.js`. No flags:

```
nodejs
```

Kotlin (`Kotlin 1.3.50`). File extensions: `.kt`. Flags:

```
kotlinc
java -Xmx1024m
```

Python (`CPython 3.6.9`). File extensions: `.py`. No flags:

```
python3
```

Ruby (`Ruby 2.5.1p57`). File extensions: `.rb`.

```
ruby
```

Rust (`Rust 1.37.0`). File extensions: `.rs`.

```
rustc
```

Scala (`Scala 2.12.10`). File extensions: `.scala`.

```
scalac
```

7.2 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.