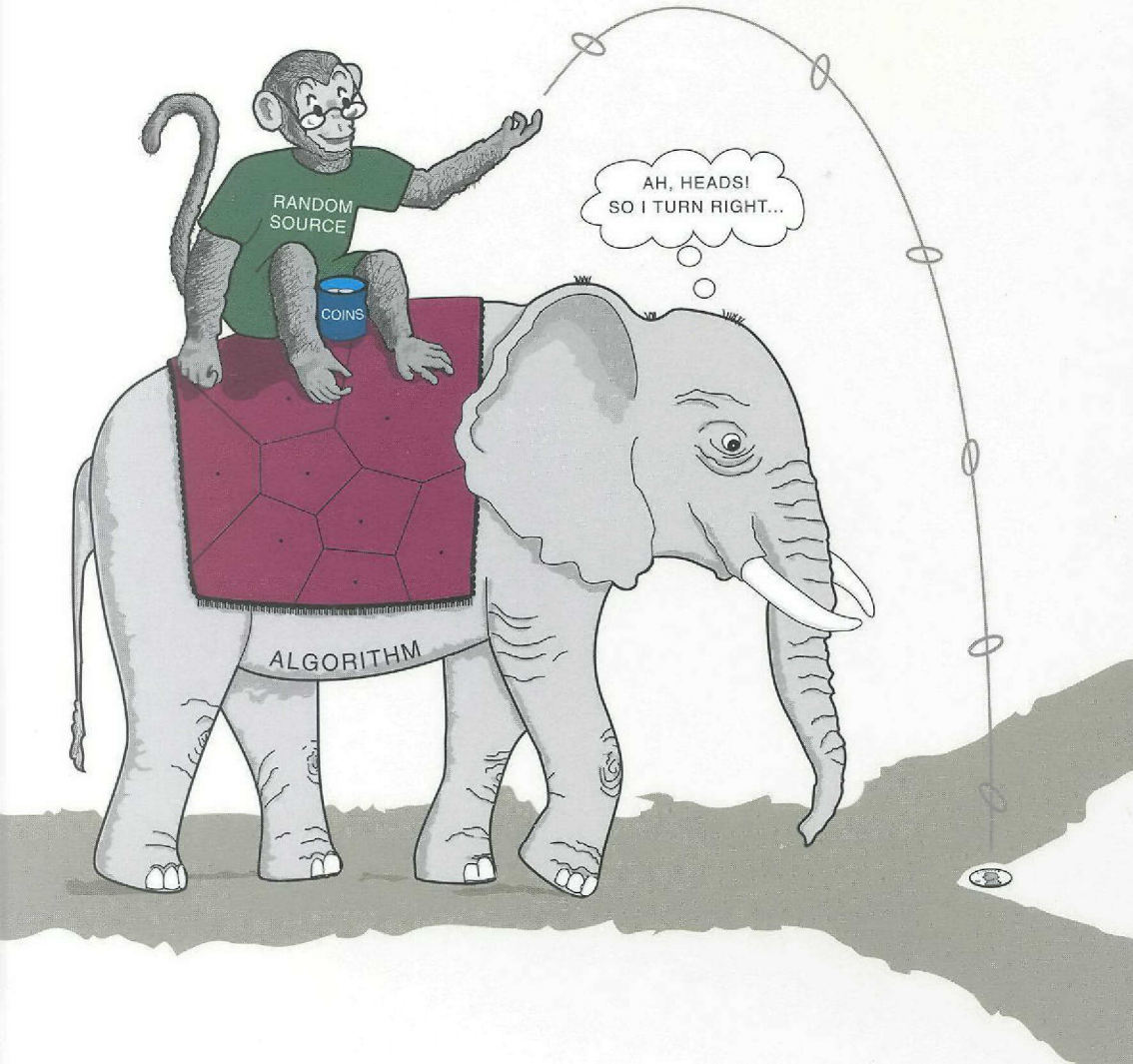
# Computational Geometry

## An Introduction Through Randomized Algorithms

Ketan Mulmuley



# Computational Geometry

## An Introduction Through Randomized Algorithms

Ketan Mulmuley
*The University of Chicago*

PRENTICE HALL, Englewood Cliffs, NJ 07632

Library of Congress Cataloging-in-Publication Data

Mulmuley, Ketan.
*Computational geometry: an introduction through randomized* algorithms / Ketan Mulmuley.
p. cm.
Includes bibliographical references and index.
ISBN 0-13-336363-5
1. Geometry--Data processing. 2. Algorithms. I. Title.
QA448.M85 1994
516'.13--dc20 93-3138
CIP

Acquisitions editor: **BILL ZOBRIST**
Production editor: **JOE SCORDATO**
*Copy editor:* **PETER J. ZURITA**
Prepress buyer: **LINDA BEHRENS**
Manufacturing buyer: **DAVID DICKEY**
Editorial assistant: **DANIELLE ROBINSON**



Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-336363-5

Prentice-Hall International (UK) Limited, London
Prentice-Hall of Australia Pty. Limited, Sydney
Prentice-Hall Canada Inc., Toronto
Prentice-Hall Hispanoamericana, S.A., Mexico
Prentice-Hall of India Private Limited, New Delhi
Prentice-Hall of Japan, Inc., Tokyo
Simon & Schuster Asia Pte. Ltd., Singapore
Editora Prentice-Hall do Brasil, Ltda., Rio de Janeiro

For Sanchit

# Contents

# Preface

This book is based on lectures given to graduate students at the University of Chicago. It is intended to provide a rapid and concise introduction to computational geometry. No prior familiarity with computational geometry is assumed. A modest undergraduate background in computer science or a related field should suffice.

My goal is to describe some basic problems in computational geometry and the simplest known algorithms for them. It so happens that several of these algorithms are randomized. That is why we have chosen randomized methods to provide an introduction to computational geometry. There is another feature of randomized methods that makes them ideal for this task: They are all based on a few basic principles, which can be applied systematically to a large number of apparently dissimilar problems. Thus, it becomes possible to provide through randomized algorithms a unified, broad perspective of computational geometry. I have tried to give an account that brings out this simplicity and unity of randomized algorithms and also their depth.

Randomization entered computational geometry with full force only in the 80s. Before that the algorithms in computational geometry were mostly deterministic. We do cover some of the very basic, early deterministic algorithms. Their study will provide the reader an historical perspective and familiarity with some deterministic paradigms that every student of computational geometry must know. It will also provide the reader an opportunity to study the relationship between these deterministic algorithms and their randomized counterparts. This relationship is quite akin to the relationship between quick-sort and deterministic sorting algorithms: Randomization yields simplicity and efficiency at the cost of losing determinism.

But randomization does more than just provide simpler alternatives to the deterministic algorithms. Its role in the later phase of computational geometry was pivotal. For several problems, there are no deterministic algorithms that match the performance of randomized algorithms. For several others, the only known deterministic algorithms are based on a technique called derandomization. Later in the book, we study the basic principles underlying

this technique. It consists of an ingenious simulation of a randomized algorithm in a deterministic fashion. It has made the study of randomized algorithms even more important.

Of course, there are also numerous problems in computational geometry for which randomization offers no help. We have not touched upon these problems in this book. Our goal is to provide a cohesive and unified account rather than a comprehensive one. Even the field of randomized algorithms has become so vast that it is impossible to cover it comprehensively within a book of this size. Finally, the choice of topics is subjective. But I hope that the book will provide the reader with a glimpse of the exciting developments in computational geometry. Once the vistas of this terrain are roughly illuminated, it is hoped that the reader will be provided with the perspective necessary to delve more deeply into the subfields of his or her choice.

## Organization of the book

Figure 0.1 shows logical dependence among the various chapters.

The book is organized in two parts—basics and applications. In the first part, we describe the basic principles that underlie the randomized algorithms in this book. These principles are illustrated with the help of simple two-dimensional geometric problems. The first part uses only elementary planar geometry. It can also be easily read by those who are mainly interested in randomized algorithms rather than in computational geometry.

In the applications part of the book, we apply the basic principles developed in the first part to several higher-dimensional problems. Here the geometry becomes more interesting. The chapters in this part are independent of each other. Thus, one can freely select and read the chapters of interest.

If the reader wishes, Chapter 2 can be read after reading just the introduction of Chapter 1. But the rest of Chapter 1 should be read before proceeding further. Chapters 1, 2, 3, and 5 should be read before proceeding to the applications part. Chapter 4 is optional, and can be skipped in the first reading after reading its introduction. This requires skipping Section 7.5 and some exercises that depend on it. The other dynamic algorithms in the book are based on the principles described in Chapter 5.

It should be noted that all randomized algorithms in this book are simple, without any exception. But sometimes their analysis is not so simple. If you are not theoretically inclined, you may skip the probabilistic analysis—which is always separate—on the first reading. But, eventually, we hope, you will wish to know why these algorithms work so well and hence turn to their analysis.

## Prerequisites

This book is meant to be understandable to those with only a modest undergraduate background in computer science or a related field. It is assumed that the reader is familiar with elementary data structures such as lists, trees, graphs, and arrays. Familiarity with at least one nontrivial data structure, such as a balanced search tree, will help. But strictly speaking even that is not required, because Chapter 1 gives a complete description of randomized search trees. The algorithms in this book are described in plain English. It is assumed that the reader can convert them into a suitable programming language, if necessary.

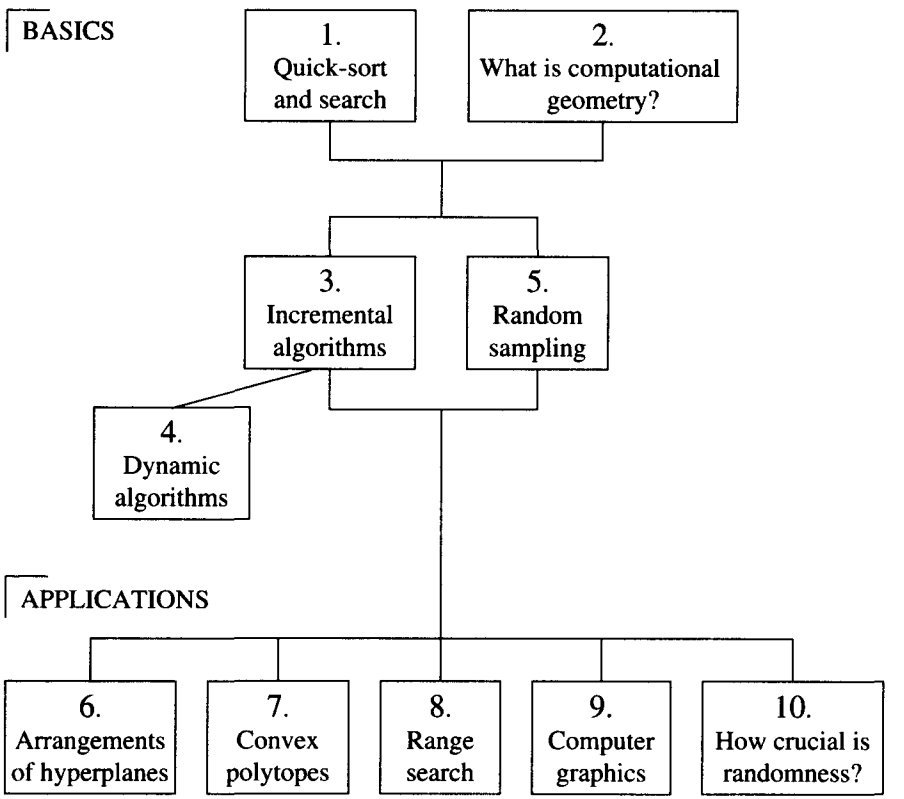



Figure 0.1: Logical dependence among the chapters.

It is also assumed that the reader is familiar with elementary notions of probability theory—random variables, their expectation, conditional expectation, and so on. We do not need anything deep from probability theory. The deeper aspects of randomized, geometric algorithms are generally geometric rather than probabilistic.

The first part of the book requires only planar geometry. The second part assumes nodding acquaintance with elementary notions in Euclidean geometry: closed set, open set, boundary of a set, linear function, and so on. Otherwise, it is meant to be self-contained.

## To the teacher

The book can be used for a course on computational geometry to beginning graduate students. Such a course may begin with some basic problems and their deterministic solutions (Chapter 2) and then shift to randomized algorithms as the problems get more complex.

On the deterministic side, the book contains only some very basic algorithms, which can be used for the first course in computational geometry. In two dimensions, we cover roughly as much ground as in, say, the book by Preparata and Shamos [185], but in higher dimensions, we go little further to address some additional deterministic techniques: incremental construction (Section 6.1), parametric search (Section 8.7), dynamization of decomposable search problems (Section 8.6), and derandomization (Section 10.2). Also, the algorithms for orthogonal intersection search and dynamic planar point location in Chapter 8 are almost deterministic. The only difference is that I have substituted the weight-balanced trees in their solutions with skip lists. So, if you wish, you can cover the deterministic versions, too, after covering the simpler randomized versions.

On the side of randomized algorithms, the book contains more than what can be reasonably covered in one semester course. The choice of material would depend on the inclination of the students. In a leisurely course to students who are not theoretically inclined, you may wish to cover only the basic algorithms in the beginnings of Chapters 3 and 5. This assumes very little mathematical sophistication. In an intensive course to theoretically inclined graduate students, you can cover most of Part I, but perhaps skip Chapter 4, and cover selected topics in Part II. Since the chapters in Part II are independent, the choice is flexible.

The book can also be used as a supplement for a course on randomized algorithms.

## Outline

Here is a brief outline of the book, chapter by chapter.

In Chapter 1, we begin with the simplest and the most well-known randomized algorithm: quick-sort. It is of special interest to us, because the randomized methods in computational geometry can be viewed as higher-dimensional generalizations of quick-sort. Quick-sort is known to be one of the simplest and the most efficient general-purpose algorithms for sorting. This indicates why its higher-dimensional analogs can be expected to be simple and efficient in practice. It is our hope that this book can help a practitioner get acquainted with some theoretical principles that may turn out to be useful in practice. With that in mind, we have chosen quick-sort as a starting point. Here, we analyze and interpret quick-sort in several ways, some of which are standard and some are not. These various interpretations are systematically extended later in the book to attack a large number of problems.

Chapter 2 gives a snapshot of computational geometry. It describes several of the basic motivating problems in the field. These problems are tackled in a unified fashion later in the book. We also cover here some early deterministic algorithms for very basic two-dimensional problems. This should provide the reader familiarity with some simple deterministic design principles and an opportunity to compare these deterministic algorithms with their randomized counterparts studied later.

Chapter 3 deals with randomized incremental algorithms in computational geometry. They solve a problem by adding the objects in the input, one at a time, in random order. This is exactly how quick-sort proceeds, if it is viewed appropriately. We describe this paradigm in general form and then illustrate it with several simple problems dealing with planar graphs, Voronoi diagrams, and so on.

Chapter 4 deals with dynamic problems. In a dynamic setting, the user is allowed to add or delete an object in the input in an on-line fashion. We develop some general principles and demonstrate these on the same two-dimensional problems considered in Chapter 3.

Chapter 5 is central to the book. It describes the principle of random sampling. This can be thought of as an extension of the randomized divide-and-conquer view of quick-sort. We describe the general principles and then illustrate them on the two-dimensional problems considered in the earlier chapters.

Chapter 6 deals with arrangements of hyperplanes. Arrangements are important in computational geometry because, among all geometric configurations, they have perhaps the simplest combinatorial structure. Thus, they

serve as a nice test-bed for the algorithmic principles developed earlier in the book.

Chapter 7 deals with convex polytopes, convex hulls, Voronoi diagrams, and related problems. Convex polytopes and Voronoi diagrams come up in many fields—signal processing, operations research, physics, and so on. There is no need to dwell on their importance here.

Chapter 8 deals with range searching problems. Range searching is an important theme that encompasses a large number of problems in computational geometry. These problems have the following form: We are given a set of geometric objects. The goal is to build a data structure so that, given a query region, one can quickly report or count the input objects that it intersects. Chapter 8 deals with several important problems of this form.

Chapter 9 deals with the applications of randomized methods to some basic problems in computer graphics.

Finally, Chapter 10 studies how crucial the randomness is for the performance of the algorithms studied earlier in the book. We shall see that the number of random bits used by most randomized incremental algorithms can be made logarithmic in the input size without changing their expected performance by more than a constant factor. Several of the algorithms based on random sampling can be made completely deterministic by "derandomizing" them. A comprehensive account of such deterministic algorithms is outside the scope of this book. However, we describe the basic principles underlying this technique.

There are several exercises throughout the book. The reader is encouraged to solve as many as possible. Some simple exercises are used in the book. These are mostly routine. Quite often, we defer the best known solution to a given problem to the exercises, if this solution is too technical. The exercises marked with * are difficult—the difficulty increases with the number of stars. The exercises marked with † are unsolved at the time of this writing. Some sections in the book are starred. They can be skipped on the first reading.

The bibliography at the end is far from being comprehensive. We have mainly confined ourselves to the references that bear directly on the methods covered in this book. For the related topics not covered in this book, we have only tried to provide a few references that can be used as a starting point; in addition, we also suggest for this purpose [63, 174, 180, 199, 231].

As for the exercises in the book, some of them are new and some are just routine or standard. For the remaining exercises, I have provided explicit references unless they directly continue or extend the material covered in the sections containing them (this should be apparent), in which case, the reference for the containing section is meant to be applicable for the exercise, too, unless mentioned otherwise.

## Acknowledgments

I am very grateful to the Computer Science Department in the University of Chicago for providing a stimulating atmosphere for the work on this book. I wish to thank the numerous researchers and graduate students who shared their insights with me. I am especially grateful to the graduate students who took the courses based on this book and provided numerous criticisms, in particular, Stephen Crocker, L. Satyanarayana, Sundar Vishwanathan, and Victoria Zanko. Sundar provided a great help in the preparation of the appendix. I also wish to thank Bernard Chazelle, Alan Frieze, Jirka Matoušek, Joe Mitchell and Rajeev Motwani for their helpful criticisms. The work on this book was supported by a Packard Fellowship. I wish to thank the Packard Foundation for its generous support. It is a pleasure to thank Adam Harris who did the artwork, including the art on the cover, and provided every kind of support during the typesetting of this book. I wish to thank Sam Rebelsky for customizing LaTeX, Bill Zobrist and Joe Scordato of Prentice Hall for their enthusiasm and cooperation. Finally, I wish to thank my wife Manju for her endurance while the book was being written. I should also acknowledge the contribution of my son Sanchit; he insisted on having his favorite animals on the cover.

Ketan Mulmuley

# Notation

| | |
|---|---|
| $f(n) = O(g(n))$ | $f(n) < cg(n)$, for some constant $c > 0$. |
| $f(n) = \Omega(g(n))$ | $f(n) > cg(n)$, for some constant $c > 0$. |
| $f(n) \approx g(n)$ | Asymptotic equality, i.e., $f(n) = O(g(n))$ and $g(n) = O(f(n))$. |
| $f(n) = \tilde{O}(g(n))$ | $f(n) = O(g(n))$ with high probability. This means, for some constant $c > 0$, $f(n) < cg(n)$ with probability $1 - 1/p(n)$, where $p(n)$ is a polynomial whose degree depends on $c$, and this degree can be made arbitrarily high by choosing $c$ large enough. Thus, $f(n) > cg(n)$ with probability $1/p(n)$, which is minuscule. |
| $X \subseteq Y$ | $X$ is a subset of $Y$. |
| $X \setminus Y$ | The relative complement of $Y$ in $X$. |
| $R$ | The real line. (On a few occasions, $R$ has a different meaning, but this will be clear from the context.) |
| $R^d$ | The $d$-fold product of $R$, i.e., the $d$-dimensional Euclidean space. |
| $\|X\|$ | The size of $X$. If $X$ is an ordinary set, then this is just its cardinality. If $X$ is a geometric partition (complex), this is the total number of its faces of all dimensions (cf. Chapter 2). Finally, if $X$ is a real number, $\|X\|$ is its absolute value. |
| $\lceil x \rceil$ | The smallest integer greater than $x$ (the ceiling function). |
| $\lfloor x \rfloor$ | The largest integer smaller than $x$ (the floor function). |
| $E[X]$ | Expected value of the random variable $X$. |
| $E[X \mid ..]$ | Expected value of $X$ subject to the specified conditions. |
| prob{..} | Probability of the specified event. |
| $\partial Z$ | Boundary of the set $Z$. |
| min{$\cdots$} | The minimum element in the set. |
| max{$\cdots$} | The maximum element in the set. |
| $[x]_d$ | The falling factorial $x(x-1)\cdots(x-d+1)$. |
| polylog($n$) | $\log^a n$, for a fixed constant $a > 0$. |

We often refer to a number which is bounded by a constant as simply a bounded number. By a random sample of a set, we mean its random subset of the specified size.

# Part I

# Basics

# Chapter 1

# Quick-sort and search

In this book, we view computational geometry as a study of sorting and searching problems in higher dimensions. In a sense, these problems are just generalizations of the ubiquitous problem in computer science—how to search and sort lists of elements drawn from an ordered universe. This latter problem can be seen as the simplest one-dimensional form of sorting and searching. This becomes clear if we reformulate it in a geometric language as follows. We are given a set $N$ of $n$ points on the real line $R$. The goal is to sort them by their coordinates. This is equivalent to (Figure 1.1):

**The sorting problem:** Find the partition $H(N)$ of $R$ formed by the given set of points $N$.

The partition $H(N)$ is formally specified by the points in $N$, the resulting (open) intervals within the line $R$—such as $J$ in Figure 1.1—and adjacencies among the points and the intervals. A geometric formulation of the associated search problem is the following:

**The search problem:** Associate a search structure $\tilde{H}(N)$ with $H(N)$ so that, given any point $q \in R$, one can locate the interval in $H(N)$ containing $q$ quickly, i.e., in logarithmic time.

In a dynamic variant of the search problem, the set $N$ can be changed in an on-line fashion by addition or deletion of a point. We are required to update $\tilde{H}(N)$ quickly, i.e., in logarithmic time, during each such update.

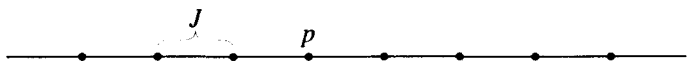



Figure 1.1: $H(N)$.

In higher dimensions, the elements of the set $N$ are not points any more, but, rather, hyperplanes or sites or polygons and so on, depending on the problem under consideration.

The simplest methods for sorting and searching linear lists are randomized. It is no surprise that the simplest known methods for several higher-dimensional sorting and searching problems are also randomized. These methods can be thought of as generalizations of the randomized methods for sorting and searching linear lists. Hence, we begin with a review of these latter methods. Actually, our goal is more than just a review—we want to *reformulate* these methods in a geometric language and analyze them in ways that can be generalized to higher dimensions. A great insight in higher-dimensional problems is obtained by just redoing the one-dimensional case.

*Remark.* In the above geometric formulation of sorting and searching in lists, we assumed that the points in $N$ were points on the line. Strictly speaking, this is not necessary. The elements in $N$ can be arbitrary, as long as they can be linearly ordered and the comparison between any two elements in $N$ can be carried out in constant time. All methods described in this chapter can be translated to this more general setting trivially.

## 1.1 Quick-sort

A simple randomized method for sorting a list of points on the real line $R$ is *quick-sort*. It is based on the randomized divide-and-conquer paradigm. Let $N$ be the given set of $n$ points in $R$. Pick a random point $S \in N$. It divides $R$ into two halves. Let $N_1, N_2 \subseteq N$ be the subsets of points contained in these two halves. Sort $N_1$ and $N_2$ recursively.

Intuitively, we should expect the sizes of $N_1$ and $N_2$ to be roughly equal to $n/2$. Hence, the expected depth of recursion is $O(\log n)$. This means the expected running time of the algorithm should be $O(n \log n)$. We shall give a rigorous justification for this in a moment.

One issue remains to be addressed here. In the division step, how do we choose a random point in $N$? In practice, this has to be done with the help of a random number generator. But the numbers generated by these so-called random number generators are not truly random. They only "appear" random. Fortunately, we shall see in Chapter 10 that all randomized algorithms in this book work very well even when the source of randomness in not perfectly random, but only "pseudo-random". Until then, we shall assume, for the sake of simplicity, that all our algorithms have ability to make perfectly random choices whenever necessary.

## 1.2 Another view of quick-sort

If we unwind the recursion in the definition of quick-sort, we get its so-called randomized incremental version. Though this version is equivalent to the version in the previous section, in higher dimensions, these two paradigms lead to markedly different algorithms.

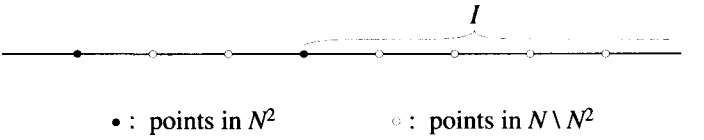



Figure 1.2: $H(N^2)$.

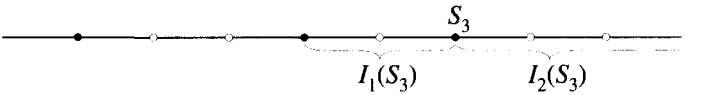



Figure 1.3: Addition of the third point.

The randomized incremental version of quick-sort works as follows. It constructs the required partition of the line incrementally by adding the points in $N$, one at a time, in random order. In other words, at any given time we choose a random point from the set of unadded points and add the chosen point to the existing partition. Let us elaborate this idea fully. Let $N^i$ be the set of the first $i$ added points. Let $H(N^i)$ be the partition of $R$ formed by $N^i$. Starting with $H(N^0)$, the empty partition of $R$, we construct a sequence of partitions

$$H(N^0), H(N^1), H(N^2), \ldots, H(N^n) = H(N).$$

At the $i$th stage of the algorithm, we also maintain, for each interval $I \in H(N^i)$, its *conflict list* $L(I)$. This is defined to be an *unordered* list of the points in $N \setminus N^i$ contained in $I$ (Figure 1.2). Conversely, with each point in $N \setminus N^i$, we maintain a pointer to the conflicting interval in $H(N^i)$ containing it.

Addition of a randomly chosen point $S = S_{i+1}$ in $N \setminus N^i$ consists in splitting the interval $I$ in $H(N^i)$ containing $S$, together with its conflict list.

Let $I_1(S)$ and $I_2(S)$ denote the intervals in $H(N^{i+1})$ adjacent to $S$ (Figure 1.3). Let $l(I_1(S))$ and $l(I_2(S))$ denote their *conflict sizes*, i.e., the sizes of their conflict lists $L(I_1(S))$ and $L(I_2(S))$. It is easily seen that:

**Fact 1.2.1** *The cost of adding $S$ is proportional to $l(I_1(S)) + l(I_2(S))$, ignoring an additive $O(1)$ term.*