

Chapter 1

Intro, Quick Sort and BSP

By Sarel Har-Peled, May 29, 2013^①

Finally: It was stated at the outset, that this system would not be here, and at once, perfected. You cannot but plainly see that I have kept my word. But I now leave my cetological System standing thus unfinished, even as the great Cathedral of Cologne was left, with the crane still standing upon the top of the uncompleted tower. For small erections may be finished by their first architects; grand ones, true ones, ever leave the copestone to posterity. God keep me from ever completing anything. This whole book is but a draft - nay, but the draft of a draft. Oh, Time, Strength, Cash, and Patience!

– Herman Melville, Moby Dick.

1.1 General Introduction

Administrivia.

- prerequisites: algorithms course, ability to do proofs
- homework weekly (first next week)
- books.

Randomized algorithms are algorithms that makes random decision during their execution. Specifically, they are allowed to use variables that their value is take from some random distribution. It is not immediately clear why adding the ability to consult with randomness would help an algorithm. But it turns out that the benefits are quite substantial:

Best. There are cases were only randomized algorithm is known or possible, especially for games. For example, consider the 3 coins example.

Speed. In some cases randomized algorithms are considerably faster than any deterministic algorithm.

^①This work is licensed under the Creative Commons Attribution-Noncommercial 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Simplicity. Even if a randomized algorithm is not faster, often it is considerably simpler than its deterministic counterpart.

Derandomization. Some deterministic algorithms arise from derandomizing the randomized algorithms, and this is the only algorithm we know for these problems (i.e., discrepancy).

Adversary arguments and lower bounds. The standard worst case analysis relies on the idea that the adversary can select the input on which the algorithm performs worst. Inherently, the adversary is more powerful than the algorithm, since the algorithm is completely predictable. By using a randomized algorithm, we can make the algorithm unpredictable and break the adversary lower bound.

1.1.1 Randomized vs average-case analysis

Randomized algorithms are not the same as *average-case analysis*. In average case analysis, one assumes

- Probabilistic analysis assuming random input
- randomized algorithms do not assume random inputs
- so analyses are more applicable

1.2 Basic probability

Here we recall some definitions about probability. The reader already familiar with these definitions can happily skip this section.

1.2.1 Formal basic definitions: Sample space, σ -algebra, and probability

Here we formally define some basic notions in probability. The reader familiar with these concepts can safely skip this part.

A *sample space* Ω is a set of all possible outcomes of an experiment. We also have a set of events \mathcal{F} , where every member of \mathcal{F} is a subset of Ω . Formally, we will require that \mathcal{F} is a σ -algebra.

Definition 1.2.1. A set \mathcal{F} of subsets of Ω is a *σ -algebra* if:

- (i) \mathcal{F} is not empty,
 - (ii) if $X \in \mathcal{F}$ then $\bar{X} = \Omega \setminus X \in \mathcal{F}$, and
 - (iii) if $X, Y \in \mathcal{F}$ then $X \cup Y \in \mathcal{F}$.
- (More generally, we will require that if $X_i \in \mathcal{F}$, for $i \in \mathbb{Z}$, then $\cup_i X_i \in \mathcal{F}$.)

We will refer to a member of \mathcal{F} as being an *event*.

As a concrete example, if we are rolling a dice, then $\Omega = \{1, 2, 3, 4, 5, 6\}$ and \mathcal{F} would be the power set of all possible subsets of Ω .

Definition 1.2.2. A *probability measure* is a mapping $\mathbf{Pr} : \mathcal{F} \rightarrow [0, 1]$ assigning *probabilities* to events. The function \mathbf{Pr} needs to have the following properties:

- (i) (additive) for $X, Y \in \mathcal{F}$ disjoint sets, we have that $\mathbf{Pr}[X \cup Y] = \mathbf{Pr}[X] + \mathbf{Pr}[Y]$, and
- (ii) $\mathbf{Pr}[\Omega] = 1$.

Definition 1.2.3. A *probability space* is a triple $(\Omega, \mathcal{F}, \mathbf{Pr})$, where Ω is a sample space, \mathcal{F} is a σ -algebra defined over Ω , and \mathbf{Pr} is a probability measure.

Definition 1.2.4. A *random variable* f is a mapping from Ω into some set \mathcal{G} . We will require that the probability of the random variable to take on any value in a given subset of values is well defined. Formally, we will require that for any subset $U \subseteq \mathcal{G}$, we have that $f^{-1}(U) \in \mathcal{F}$. That is, $\mathbf{Pr}[f \in U] = \mathbf{Pr}[f^{-1}(U)]$ is defined.

Going back to the dice example, the number on the top of the dice when we roll it is a random variable. Similarly, let X be one if the number rolled is larger than 3, and zero otherwise. Clearly X is a random variable.

We denote the *probability* of a random variable X to get the value x , by $\mathbf{Pr}[X = x]$ (or sometime $\mathbf{Pr}[x]$, if we are really lazy).

1.2.2 Expectation and conditional probability

Definition 1.2.5 (Expectation.). The expectation of a random variable X , is its average. Formally, the *expectation* of X is

$$\mathbf{E}[X] = \sum_x x \mathbf{Pr}[X = x].$$

Definition 1.2.6 (Conditional Probability.). The *conditional probability* of X given Y , is the probability that $X = x$ given that $Y = y$. We denote this quantity by $\mathbf{Pr}[X = x \mid Y = y]$.

The conditional probability can be computed using the formula

$$\mathbf{Pr}[X = x \mid Y = y] = \frac{\mathbf{Pr}[(X = x) \cap (Y = y)]}{\mathbf{Pr}[Y = y]}.$$

For example, let us roll a dice and let X be the number we got. Let Y be the random variable that is true if the number we get is even. Then, we have that

$$\mathbf{Pr}[X = 2 \mid Y = \text{true}] = 1/3.$$

Definition 1.2.7. Two random variables X and Y are *independent* if $\mathbf{Pr}[X = x \mid Y = y] = \mathbf{Pr}[X = x]$, for all x .

Lemma 1.2.8 (Linearity of expectation.). *Linearity of expectation* is the property that for any two random variables X and Y , we have $\mathbf{E}[X + Y] = \mathbf{E}[X] + \mathbf{E}[Y]$.

1.3 QuickSort

Let the input be a set t_1, \dots, t_n of n items to be sorted. We remind the reader, that the **QuickSort** algorithm randomly pick a pivot element (uniformly), splits the input into two subarrays of all the elements smaller than the pivot, and all the elements larger than the pivot, and then it recurses on these two subarrays (the pivot is not included in these two subproblems). Here we will show that the expected running time of **QuickSort** is $O(n \log n)$.

Definition 1.3.1. For an event \mathcal{E} , let X be a random variable which is 1 if \mathcal{E} occurred and 0 otherwise. The random variable X is an *indicator variable*.

Observation 1.3.2. For an indicator variable X of an event \mathcal{E} , we have

$$\mathbf{E}[X] = \Pr[X = 1] = \Pr[\mathcal{E}].$$

Let S_1, \dots, S_n be the elements in their sorted order (i.e., the output order). Let $X_{ij} = 1$ be the indicator variable which is one iff **QuickSort** compares S_i to S_j , let p_{ij} denote the probability that this happens. Clearly, the number of comparisons performed by the algorithm is $C = \sum_{i < j} X_{ij}$. By linearity of expectations, we have

$$\mathbf{E}[C] = \sum_{i < j} \mathbf{E}[X_{ij}] = \sum_{i < j} p_{ij}.$$

We want to bound p_{ij} , the probability that the S_i is compared to S_j . Consider the last recursive call involving both S_i and S_j . Clearly, the pivot at this step must be one of S_i, \dots, S_j , all equally likely. Indeed, S_i and S_j were separated in the next recursive call.

Observe, that S_i and S_j get compared if and only if pivot is S_i or S_j . Thus, the probability for that is $2/(j - i + 1)$. Indeed,

$$p_{ij} = \Pr[S_i \text{ or } S_j \text{ picked} \mid \text{picked pivot from } S_i, \dots, S_j] = \frac{2}{j - i + 1}.$$

Thus,

$$\begin{aligned} \sum_{i=1}^n \sum_{j>i} p_{ij} &= \sum_{i=1}^n \sum_{j>i} 2/(j - i + 1) = \sum_{i=1}^n \sum_{k=1}^{n-i+1} \frac{2}{k} \leq 2 \sum_{i=1}^n \sum_{k=1}^n \frac{1}{k} \\ &\leq 2nH_n \leq n + 2n \ln n, \end{aligned}$$

where H_n is the *harmonic number*^② $H_n = \sum_{i=1}^n \frac{1}{i}$. We thus proved the following result.

Lemma 1.3.3. **QuickSort** performs in expectation at most $n + 2n \ln n$ comparisons, when sorting n elements.

Note, that this holds for all inputs. No assumption on the input is made. Similar bounds holds not only in expectation, but also with high probability.

This raises the question, of how does the algorithm pick a random element? We assume we have access to a random source that can get us number between 1 and n uniformly.

Note, that the algorithm always works, but it might take quadratic time in the worst case.

^②Using integration to bound summation, we have $H_n \leq 1 + \int_{x=1}^n \frac{1}{x} dx \leq 1 + \ln n$. Similarly, $H_n \geq \int_{x=1}^n \frac{1}{x} dx = \ln n$.

1.4 Binary space partition (BSP)

Let assume that we would like to render an image of a three dimensional scene on the computer screen. The input is in general a collection of polygons in three dimensions. The *painter* algorithm, render the scene by drawing things from back to front; and let front stuff overwrite what was painted before.

The problem is that it is not always possible to order the objects in three dimensions. This ordering might have cycles. So, one possible solution is to build a *binary space partition*. We build a binary tree. In the root, we place a polygon P . Let h be the plane containing P . Next, we partition the input polygons into two sets, depending on which side of h they fall into. We recursively construct a BSP for each set, and we hang it from the root node. If a polygon intersects h then we cut it into two polygons as split by h . We continue the construction recursively on the objects on one side of h , and the objects on the other side. What we get, is a binary tree that splits space into cells, and furthermore, one can use the painter algorithm on these objects. The natural question is how big is the resulting partition.

We will study the easiest case, of disjoint segments in the plane.

1.4.1 BSP for disjoint segments

Let $P = \{s_1, \dots, s_n\}$ be n disjoint segments in the plane. We will build the BSP by using the lines defined by these segments. This kind of BSP is called *autopartition*.

To recap, the BSP is a binary tree, at every internal node we store a segment of P , where the line associated with it splits its region into its two children. Finally, each leaf of the BSP stores a single segment. A *fragment* is just going to be a subsegment formed by this splitting. Clearly, every internal node, stores a fragment that defines its split. As such, the size of the BSP is proportional to the number of fragments generated when building the BSP.

One application of such a BSP is ray shooting - given a ray you would like to determine what is the first segment it hits. Start from the root, figure out which child contains the apex of the ray, and first (recursively) compute the first segment stored in this child that the ray intersect. Contain into the second child only if the first subtree does not contain any segment that intersect the ray.

1.4.1.1 The algorithm

We pick a random permutation σ of $1, \dots, n$, and in the i th step we insert $s_{\sigma(i)}$ splitting all the cells that s_i intersects.

Observe, that if s_i crosses a cell completely, it just splits it into two and no new fragments are created. As such, the bad case is when a segment s is being inserted, and its line intersect some other segment t .

So, let $\mathcal{E}(s, t)$ denote the event that when inserted s it had split t . In particular, let $\text{index}(s, t)$ denote the number of segments on the line of s between s (closer) endpoint and t (including t). If the line of s does not intersect t , then $\text{index}(s, t) = \infty$.

We have that

$$\Pr[\mathcal{E}(s, t)] = \frac{1}{1 + \text{index}(s, t)}.$$

Let $X_{s,t}$ be the indicator variable that is 1 if $\mathcal{E}(s, t)$ happens. We have that

$$S = \text{number of fragments} = \sum_{i=1}^n \sum_{j=1, i \neq j}^n X_{s_i, s_j}.$$

As such, by linearity of expectations, we have

$$\begin{aligned} \mathbf{E}[S] &= \mathbf{E}\left[\sum_{i=1}^n \sum_{j=1, i \neq j}^n X_{s_i, s_j}\right] = \sum_{i=1}^n \sum_{j=1, i \neq j}^n \mathbf{E}[X_{s_i, s_j}] = \sum_{i=1}^n \sum_{j=1, i \neq j}^n \Pr[\mathcal{E}(s_i, s_j)] \\ &= \sum_{i=1}^n \sum_{j=1, i \neq j}^n \frac{1}{1 + \text{index}(s_i, s_j)} \\ &\leq \sum_{i=1}^n \sum_{j=1}^n \frac{2}{1+j} = 2nH_n. \end{aligned}$$

Since the size of the BSP is proportional to the number of fragments created, we have the following result.

Theorem 1.4.1. *Given n disjoint segments in the plane, one can build a BSP for them of size $O(n \log n)$.*

Csaba Tóth [Tót03] showed that BSP for segments in the plane, in the worst case, has complexity $\Omega\left(n \frac{\log n}{\log \log n}\right)$.

1.5 Extra: QuickSelect running time

We remind the reader that **QuickSelect** receives an array $t[1 \dots n]$ of n real numbers, and a number k , and returns the element of rank k in the sorted order of the elements of t . We can of course, use **QuickSort**, and just return the k th element in the sorted array, but a more efficient algorithm, would be to modify **QuickSelect**, so that it recurses on the subproblem that contains the element we are interested in. Formally, **QuickSelect** chooses a random pivot, splits the array according to the pivot. This implies that we now know the rank of the pivot, and if its equal to \bar{m} , we return it. Otherwise, we recurse on the subproblem containing the required element (modifying \bar{m} as we go down the recursion. Namely, **QuickSelect** is a modification of **QuickSort** performing only a single recursive call (instead of two).

AS before, to bound the expected running time, we will bound the expected number of comparisons. As before, let S_1, \dots, S_n be the elements of t in their sorted order. Now, for $i < j$, let X_{ij} be the indicator variable that is one if S_i is being compared to S_j during the execution of **QuickSelect**. There are several possibilities to consider:

- (i) If $i < j < \bar{m}$: Here, S_i is being compared to S_j , if and only if the first pivot in the range

S_i, \dots, S_k is either S_i or S_j . The probability for that is $2/(k - i + 1)$. As such, we have that

$$\begin{aligned}\alpha_1 &= \mathbf{E} \left[\sum_{i < j < \bar{m}} X_{ij} \right] = \mathbf{E} \left[\sum_{i=1}^{med-2} \sum_{j=i+1}^{\bar{m}-1} X_{ij} \right] = \sum_{i=1}^{med-2} \sum_{j=i+1}^{\bar{m}-1} 2/(\bar{m} - i + 1) \\ &= \sum_{i=1}^{med-2} 2(\bar{m} - i - 1)/(\bar{m} - i + 1) \leq 2(\bar{m} - 2).\end{aligned}$$

(ii) If $\bar{m} < i < j$: Using the same analysis as above, we have that $\Pr[X_{ij} = 1] = 2/(j - \bar{m} + 1)$. As such,

$$\alpha_2 = \mathbf{E} \left[\sum_{j=\bar{m}+1}^n \sum_{i=\bar{m}+1}^{j-1} X_{ij} \right] = \sum_{j=\bar{m}+1}^n \sum_{i=\bar{m}+1}^{j-1} \frac{2}{j - \bar{m} + 1} = \sum_{j=\bar{m}+1}^n \frac{2(j - \bar{m} - 1)}{j - \bar{m} + 1} \leq 2(n - \bar{m}).$$

(iii) $i < \bar{m} < j$: Here, we compare S_i to S_j if and only if the first indicator in the range S_i, \dots, S_j is either S_i or S_j . As such, $\mathbf{E}[X_{ij}] = \Pr[X_{ij} = 1] = 2/(j - i + 1)$. As such, we have

$$\alpha_3 = \mathbf{E} \left[\sum_{i=1}^{\bar{m}-1} \sum_{j=\bar{m}+1}^n X_{ij} \right] = \sum_{i=1}^{\bar{m}-1} \sum_{j=\bar{m}+1}^n \frac{2}{j - i + 1}.$$

Observe, that for a fixed $\Delta = j - i + 1$, we are going to handle the gap Δ in the above summation, at most $\Delta - 2$ times. As such, $\alpha_3 \leq \sum_{\Delta=3}^n 2(\Delta - 2)/\Delta \leq 2n$.

(iv) $i = \bar{m}$. We have $\alpha_4 = \sum_{j=\bar{m}+1}^n \mathbf{E}[X_{ij}] = \sum_{j=\bar{m}+1}^n \frac{2}{j - \bar{m} + 1} = \ln n + 1$.

(v) $j = \bar{m}$. We have $\alpha_5 = \sum_{i=1}^{\bar{m}-1} \mathbf{E}[X_{ij}] = \sum_{i=1}^{\bar{m}-1} \frac{2}{\bar{m} - i + 1} \leq \ln \bar{m} + 1$.

Thus, the expected number of comparisons performed by **QuickSelect** is bounded by

$$\sum_i \alpha_i \leq 2(\bar{m} - 2) + 2(n - \bar{m}) + 2n + \ln n + 1 + \ln \bar{m} = 4n - 2 + \ln n + \ln \bar{m}.$$

Theorem 1.5.1. *In expectation, **QuickSelect** performs at most $4n - 2 + \ln n + \ln \bar{m}$ comparisons, when selecting the \bar{m} th element out of n elements.*

A different approach can reduce the number of comparisons (in expectation) to $1.5n + o(n)$. More on that later in the course.

Bibliography

[Tóth03] C. D. Tóth. A note on binary plane partitions. *Discrete Comput. Geom.*, 30(1):3–16, 2003.