Chapter 1

Intro, Quick Sort and BSP

By Sariel Har-Peled, November 13, 2008

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.

\[^{[1]}\]This 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/](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

**Definition 1.2.1 Random variable.**

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

**Definition 1.2.2 (Expectation.)** The expectation of a random variable $X$, is its average. Formally, the expectation of $X$ is

$$E[X] = \sum_x x \Pr[X = x].$$

**Definition 1.2.3 (Conditional Probability.)** The conditional probability of $X$ given $Y$, is the probability that $X = x$ given that $Y = y$. We denote this quantity by $\Pr[X = x \mid Y = y]$.

The conditional probability can be computed using the formula

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

For example, let role a dice. $Y$ would be true if the number we get is even, and $X$ would be the number we get. Then

$$\Pr[X = 2 \mid Y = true] = 1/3.$$
Definition 1.2.4 Two random variables $X$ and $Y$ are independent if $\Pr[X = x \mid Y = y] = \Pr[X = x]$, for all $x$.

Lemma 1.2.5 (Linearity of expectation.) Linearity of expectation is the property that for any two random variables $X$ and $Y$, we have $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$.

1.3 QuickSort

Let the input be a set $t_1, \ldots, t_n$ 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 $\mathbb{E}[X] = \Pr[X = 1] = \Pr[\mathcal{E}]$.

Let $S_1, \ldots, S_n$ 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

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

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 $S_i, \ldots, S_j$, 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,

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

Thus,

$$\sum_{i=1}^{n} \sum_{j>i} p_{ij} = \sum_{i=1}^{n} \sum_{j>i} \frac{2}{j - i + 1} = \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \leq 2 n \ln n \leq n + 2 n \ln n,$$

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

---

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

3
**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.

### 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 $P = \{s_1, \ldots, s_n\}$ 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 $\sigma$ of $1, \ldots, n$, 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 $E(s, t)$ denote the event that when inserted $s$ it had split $t$. In particular, let 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 index$(s, t) = \infty$.

We have that
\[
Pr[E(s, t)] = \frac{1}{1 + \text{index}(s, t)}.
\]

Let $X_{s,t}$ be the indicator variable that is 1 if $E(s, t)$ 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
\[
E[S] = E \left[ \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} X_{s_i, s_j} \right] = \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} E[X_{s_i, s_j}] = \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} Pr[E(s_i, s_j)]
\]
\[
= \sum_{i=1}^{n} \sum_{j=1, i \neq j}^{n} \frac{1}{1 + \text{index}(s_i, s_j)}
\]
\[
\leq \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{2}{1 + j} = 2nH_n.
\]

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 $\Omega \left( \frac{n \log n}{\log \log n} \right)$.

**Bibliography**