Chapter 1

Randomized Quicksort

In contrast to the deterministic variant, randomized quicksort chooses the pivot element uniformly at random, i.e., every element has the same probability to become the pivot. We will discuss the motivation for this random choice in the following analysis.

**Input:** a set $S$ of $n$ different numbers, stored in an array  
**Output:** the numbers from $S$ in increasing order

**Algorithm** randQS($S$)

1. Choose $y$ from $S$ uniformly at random;
2. $S_1 := \{x \in S \mid x < y\};$ $S_2 := \{x \in S \mid x > y\};$
3. If $S_1 \neq \emptyset$ then randQS($S_1$);
4. output $y$;
5. If $S_2 \neq \emptyset$ then randQS($S_2$);

We measure the running time of the algorithm by the number of comparisons. How many comparisons does the algorithm? - Clearly, this number depends on the outcome of the random experiments that determine the pivot elements on the different levels of recursion.
1.1 The worst case

Suppose the input $S$ is stored in an array in sorted form already and the random experiments come out in such a way that always the first element is chosen as pivot (as in many implementations of deterministic quicksort).

**Example:**

<table>
<thead>
<tr>
<th>execution</th>
<th>#comp.</th>
<th>probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>6</td>
<td>1/7</td>
</tr>
<tr>
<td>2 3 4 5 6 7</td>
<td>5</td>
<td>1/6</td>
</tr>
<tr>
<td>3 4 5 6 7</td>
<td>4</td>
<td>1/5</td>
</tr>
<tr>
<td>6 7</td>
<td>1</td>
<td>1/2</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

In this case, the number of comparisons is

$$\sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} = \Theta(n^2) .$$

However, the probability that the pivot is always the first element is only

$$\prod_{i=1}^{n} \frac{1}{i} = \frac{1}{n!} .$$

Thus, this particular bad event is extremely unlikely.

Observe that there are many other events that lead to a bad performance. The following analysis will show, however, that the overwhelming number of possible outcomes of the random experiments for choosing the pivots lead to a good performance, regardless of the input.
1.2 Probabilistic running time analysis

We calculate the expected number of comparisons. Define $s_i := i$-th smallest element $(1 \leq i \leq n)$. We introduce binary random variables

$$X_{i,j} := \begin{cases} 1 & \text{if algorithm compares } s_i \text{ and } s_j, \\ 0 & \text{otherwise,} \end{cases}$$

for $1 \leq i < j \leq n$. In this way, the total number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j},$$

and the expected number of comparisons is

$$\mathbb{E} \left[ \sum_{i=1}^{n} \sum_{j=i+1}^{n} X_{i,j} \right] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{E} [X_{i,j}]$$

by linearity of expectation (see Appendix A.12). Now define

$$p_{i,j} := \Pr [X_{i,j} = 1]$$

then

$$\mathbb{E} [X_{i,j}] := 1 \cdot p_{i,j} + 0 \cdot (1 - p_{i,j}) = p_{i,j}.$$ 

Thus, the expected number of comparisons is

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{i,j}.$$ 

It remains to determine the values $p_{i,j}$, for $1 \leq i < j \leq n$. For this purpose, we consider the recursion tree $T$ of the algorithm. The root of $T$ represents the initial call $\text{randQS}(S)$, the root of the left and the right subtree represent the recursive calls $\text{randQS}(S_i)$ and $\text{randQS}(S_2)$, respectively, and so on recursively so that the leaf nodes represent calls to $\text{randQS}$ for single elements. For each node in $T$, let us assign the pivot of the corresponding call to this node.
Example:

<table>
<thead>
<tr>
<th>Execution</th>
<th>Recursion Tree T</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 4 2 1 5 7 6</td>
<td>3</td>
</tr>
<tr>
<td>2 1 3 4 5 7 6</td>
<td>1</td>
</tr>
<tr>
<td>1 2 3 4 5 7 6</td>
<td>5</td>
</tr>
<tr>
<td>1 2 3 4 5 6 7</td>
<td>2 4 6</td>
</tr>
</tbody>
</table>

Define $\pi := \text{permutation of level-wise traversal of } T$. (In the example, $\pi = 3152467$.)

The following two lemmas enable us to determine the exact value of $p_{i,j}$.

**Lemma 1.1** There is a comparison between $s_i$ and $s_j$ ($i < j$) if and only if either $s_i$ or $s_j$ occurs earlier in $\pi$ than any other element in the set $S_{i,j} := \{s_i, s_{i+1}, \ldots, s_j\}$.

**Proof.** Let $s^*$ denote the element from $S_{i,j}$ that occurs first in $\pi$. Then the set $S_{i,j}$ is completely contained in the subtree rooted at $s^*$ because none of the pivots preceding $s^*$ in $\pi$ splits this set.

Suppose $s^* \in \{s_i, s_j\}$. If $s^* = s_i$ then $s_j$ is compared with $s_i$ at the time when $s_i$ is pivot and $s_j$ is assigned to the right subtree below $s_i$. Analogously, if $s^* = s_j$ then $s_i$ is compared with $s_j$ at the time when $s_j$ is pivot and $s_i$ is assigned to the left subtree below $s_j$. Thus, $s_i$ and $s_j$ are compared if one of them is the first element from $S_{i,j}$ that becomes the pivot.
Now suppose \( s^* \not\in \{s_i, s_j\} \). In this case \( s_i \) and \( s_j \) are compared with \( s^* \) at the time when \( s^* \) is pivot; \( s_i \) is assigned to the left subtree below \( s^* \) and \( s_j \) to the right subtree. Thus, \( s_i \) and \( s_j \) become pivot elements after \( s^* \) and they are not compared with each other because they reside in different subtrees at the time when they become pivots.

\[ \Box \]

**Lemma 1.2** The probability that either \( s_i \) or \( s_j \) is the element from \( S_{i,j} \) that occurs first in \( \pi \) is exactly \( \frac{2}{j-i+1} \).

**Proof.** Each element in \( S_{i,j} \) is equally likely to be the element from \( S_{i,j} \) occurring first because pivots are chosen uniformly at random. Thus each element has probability

\[
\frac{1}{|S_{i,j}|} = \frac{1}{j-i+1} .
\]

to occur first. Hence, the probability that either \( s_i \) or \( s_j \) are the first elements is \( \frac{2}{j-i+1} \).

\[ \Box \]

Putting together Lemma 1.1 and 1.2 yields

\[
p_{i,j} = \frac{2}{j-i+1} .
\]

Consequently, the expected number of comparisons is

\[
\sum_{i=1}^{n} \sum_{j=i+1}^{n} p_{i,j} = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \leq 2 \sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} .
\]

It follows that the expected number of comparisons is bounded above by \( 2nH_n \), where \( H_n \) is the \( n \)-th Harmonic number, defined by

\[
H_n := \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1) .
\]

This implies the following theorem.
Theorem 1.3 The expected number of comparisons in an execution of randomized quicksort is at most $2nH_n = O(n \log n)$.

1.3 Remarks

- Deterministic quicksort has average running time $O(n \log n)$. However, the algorithm performs poorly on some particular inputs. (In a naive implementation, the worst running time occurs when the input is sorted.)

- In contrast, the expected running time of randomized quicksort for every input is $O(n \log n)$.

- Moreover, not only the expected running time is small. One can also show that it is very unlikely that randomized quicksort needs much more than $2nH_n$ comparisons.