Analysis & design of Algorithms Probabilistic Analysis & Quicksort

Angel Napa

May 2024

1 Probability Review

A *random variable* X is a function $X : S \to \mathbb{R}$. This function associates a real number with each possible outcome of an experiment.

For example, if the experiment is rolling a die, we have $S = \{1, 2, 3, 4, 5, 6\}$.

A random variable could be $X(1) = 1, X(2) = 2, X(3) = 3, X(4) = 4, X(5) = 5$, $X(6) = 6$ which represents the *number rolled on the die*.

Another random variable could be $X(1) = 0, X(2) = 0, X(3) = 0, X(4) = 1$, $X(5)= 1, X(6)= 1$ which answers the question of whether the *number rolled on the die is greater than 3*.

For example, if the experiment is rolling two dice, we have $S = \{1, 2, 3, 4, 5, 6\} \times \{1, 2, 3, 4, 5, 6\} = \{11, 12, 13, \dots, 46, 56, 66\}$.

A random variable could be $X(ij) = i + j$ which represents the *total result of both dice*.

Another random variable could be $X(ij) = \max\{i, j\}$ which represents the *maximum of the two values shown on the dice*.

Notation: The set

$$\{s \in S : X(s) = x\}$$

is called an *event* and is denoted by $X = x$.

For example, in the last example (rolling two dice and seeing the maximum) we have that $X = 3$ denotes the event $\{13, 23, 33, 31, 32\}$.

We then define the probability of an event as follows.

$$Pr\{X = x\} = \frac{|X = x|}{|S|}$$

In the example, $Pr\{X = 3\} == \frac{|\{13,23,33,32,31\}|}{36} = \frac{5}{36}$. In words, this means that the probability of rolling two dice and the maximum result being exactly 3 is $\frac{5}{36}$.

We define the *probability function* of the random variable X as

$$f(x) = Pr\{X = x\}.$$

It is known, by the axioms of probability, that $f(x) \geq 0$ and that $\sum_x f(x) = 1$. In the example, we have that $f(1) + f(2) + f(3) + f(4) + f(5) + f(6) = 1$ (verify manually).

The *expected value (mean, average)* of a random variable X is

$$E[X] = \sum_x x f(x).$$

In the previous example, we have

$$\begin{aligned}
E[X] &= \sum_x x \cdot f(x) \\
&= 1 \cdot f(1) + 2 \cdot f(2) + 3 \cdot f(3) + 4 \cdot f(4) + 5 \cdot f(5) + 6 \cdot f(6) \\
&= 1 \cdot Pr\{11\} + 2 \cdot Pr\{12, 21, 22\} + 3 \cdot Pr\{13, 23, 33, 32, 31\} \\
&+ 4 \cdot Pr\{14, 24, 34, 44, 43, 42, 41\} + 5 \cdot Pr\{15, 25, 35, 45, 55, 54, 53, 52, 51\} \\
&+ 6 \cdot Pr\{16, 26, 36, 46, 56, 66, 65, 64, 63, 62, 61\} \\
&= 1 \cdot \frac{1}{36} + 2 \cdot \frac{3}{36} + 3 \cdot \frac{5}{36} + 4 \cdot \frac{7}{36} + 5 \cdot \frac{9}{36} + 6 \cdot \frac{11}{36} \\
&= \frac{161}{36}.
\end{aligned}$$

This means that if we roll two dice, we expect the maximum value of both dice to be $\frac{161}{36} = 4.4722....$

Property of linearity: if X and Y are random variables, then

$$E[X + Y] = E[X] + E[Y].$$

2 The Hiring Problem

Suppose you want to hire a candidate from a list of n candidates, such that you have to pay a cost (in time or money) for interviewing or hiring any candidate. Suppose also that you can only evaluate the candidates one at a time, and you have the option to fire as many times as you wish.

```
HIRE-ASSISTANT(n)
best = 0          // candidate 0 is a least-qualified dummy candidate
for i = 1 to n
    interview candidate i
    if candidate i is better than candidate best
        best = i
        hire candidate i
```

Figure 1: Taken from Cormen's book, Introduction to Algorithms

Let c_e be the cost of interviewing a person and c_d be the cost of hiring a person. Let m be the number of people hired. We have that the total cost is $O(c_e n + c_d m)$.

Note that the variable cost depends on m. In the worst case, $m = n$ and the time is $O(c_e n + c_d n)$. We will now analyze what happens in the average case using probability techniques.

We want to estimate the expected number of times we hire a new employee. The experiment space is all the possible $n!$ permutations in which the candidates can arrive.

Let X be the random variable that holds the number of hires. We can find the expected value of X as

$$E[X] = \sum_{x=1}^{n} x \cdot Pr\{X = x\}.$$

But it is a bit complicated to do it this way. We will calculate it with auxiliary random variables. These types of random variables are also commonly called *indicator variables*.

For each $i \in \{1, 2, \ldots, n\}$, we define the random variable X_i, as follows:

$$X_i(p) = \begin{cases} 1 & \text{if candidate } i \text{ is hired in permutation } p \\ 0 & \text{if candidate } i \text{ is not hired in permutation } p \end{cases}$$

For example, if $n = 3$ and the candidates are $\{1, 2, 3\}$, the space is $\{123, 132, 231, 213, 312, 321\}$. And we have that $X_2(123) = 1, X_2(132) = 1, X_2(213) = 0, X_2(231) = 1, X_2(312) = 0, X_2(321) = 0$.

Exercise 2.1. *Find* X_1, X_2, X_3 *when* $n = 3$ *and the candidates are* $\{1, 2, 3\}$.

Exercise 2.2. *Find* X_1, X_2, X_3, X_4 *when* $n = 4$ *and the candidates are* $\{1, 2, 3, 4\}$.

Note then that X, the random variable that holds the number of hires, can be expressed as

$$X = X_1 + X_2 + \cdots + X_n.$$

Also note that $E[X_i] = 1 \cdot Pr\{X_i = 1\} + 0 \cdot Pr\{X_i = 0\} = Pr\{X_i = 1\} = Pr\{\text{candidate i is hired}\}$.

So we must answer, what is the probability that the i-th candidate is hired?

Let $a_1, a_2, \ldots, a_i, a_{i+1}, \ldots a_n$ be the order in which the candidates arrive. Note that

$$\begin{aligned} Pr\{\text{candidate i is hired}\} &= Pr\{a_i \text{ is the maximum in the sequence } a_1, a_2, \ldots, a_i\} \\ &= Pr\{\text{the maximum in the sequence } a_1, a_2, \ldots, a_i \text{ appears at the end}\} \\ &= \frac{(i-1)!}{i!} \\ &= \frac{1}{i}. \end{aligned}$$

Then

$$
\begin{aligned}
E[X] &= E[\sum_{i=1}^{n} X_i] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} (1/i) \\
&= \ln n + O(1).
\end{aligned}
$$

That is, on average, the estimated hiring cost is $O(c_d \ln n)$.

3 Quicksort

We will present an algorithm for the sorting problem. This algorithm has a worst-case running time of $\Theta(n^2)$. However, in the average case, its running time is $\Theta(n \lg n)$.

The QUICKSORT algorithm uses the divide-and-conquer paradigm. Below, we show these three steps for an array $A[p..r]$.

- **Divide:** Rearrange the array such that there exists an index q that satisfies $A[p..q-1] \leq A[q] < A[q+1..r]$. The subarrays $A[p..q-1]$ and $A[q+1..r]$ are the subproblems.
- **Conquer:** Sort the subarrays $A[p..q-1]$ and $A[q+1..r]$ by recursive calls to the same algorithm.
- **Combine:** Since the subarrays are already sorted, no merging operations are needed.

Input: An array $A[p..r]$ of integers.
Sorts the array in increasing order.

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

Figure 2: Taken from Cormen's book, Introduction to Algorithms

Input: An array $A[p..r]$ of integers.
Reorganizes the array A and returns an index q such that
$A[p..q-1] \leq A[q] < A[q+1..r]$.

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Figure 3: Taken from Cormen's book, Introduction to Algorithms

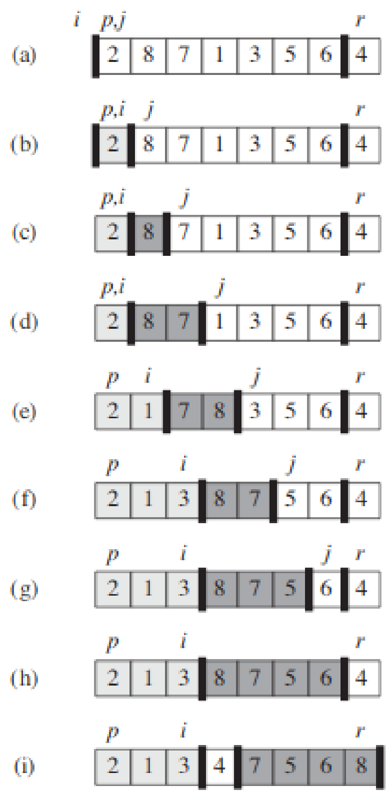

Figure 4: Taken from Cormen's book, Introduction to Algorithms

We have the following invariant for the subroutine PARTITION:
At the start of each iteration of the loop in lines 3–6,

1. $A[p..i] \leq x$
2. $A[i+1..j-1] > x$
3. $A[r] = x$

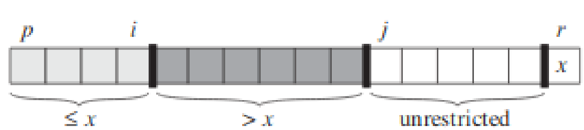

Figure 5: Taken from Cormen's book, Introduction to Algorithms

- Initialization. Before the first iteration of the for loop, we have $i = p - 1$ and $j = p$. Therefore, the first two conditions of the invariant are trivially satisfied. Additionally, due to line 1, we have $A[r] = x$.

- Maintenance. Depending on the if statement in line 4, we have two possibilities. If $A[j] > x$, then j is simply incremented and conditions 1 and 2 hold for $j + 1$ instead of j.

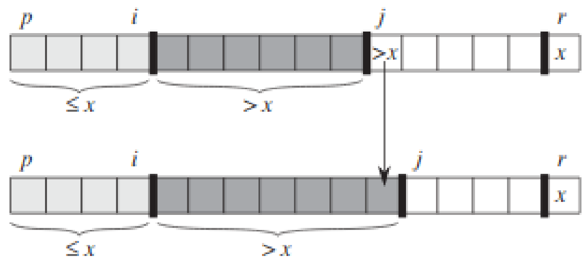

Figure 6: Taken from Cormen's book, Introduction to Algorithms

If $A[j] \leq x$, then i is incremented, $A[i]$ is swapped with $A[j]$, and j is incremented. Since $A[i] > x$, condition 1 will be satisfied. Since $A[j-1] \leq x$, condition 2 will also be satisfied.

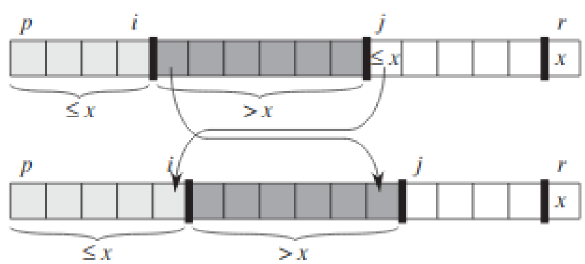

Figure 7: Taken from Cormen's book, Introduction to Algorithms

- Termination. At the end of the for loop, we have $j = r$ and therefore, $A[1..i] \leq x < A[i + 1..r - 1]$ and $A[r] = x$.

  After this, due to line 7, we will have an index q such that $A[1..q] \leq A[q] < A[q + 1..r]$.

It is relatively easy to see that the running time of the subroutine Partition on the subarray $A[p..r]$ is $\Theta(n)$, with $n = r - p + 1$.

Exercise 3.1. *Illustrate the* Partition *operation on the array* $A = [13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11]$.

4 Time Analysis of Quicksort

Let $T(n)$ be an upper bound on the execution time of QUICKSORT when receiving n elements, then, for all $n > 0$, there exists $k > 0$ such that

$$T(n) \leq \max_{0 \leq q \leq n-1} \{T(q) + T(n-1-q)\} + kn.$$

On the other hand, if $T(n)$ is a lower bound on the execution time of QUICKSORT when receiving n elements, then, for all $n > 0$, there exists $k > 0$ such that

$$T(n) \geq \min_{0 \leq q \leq n-1} \{T(q) + T(n-1-q)\} + kn.$$

In general, the execution time of QUICKSORT depends on whether the partition is balanced or unbalanced. If the partition is balanced, it is as fast as MERGESORT ($\Theta(n \lg n)$); if the partition is not, it can be as slow as INSERTIONSORT ($\Theta(n^2)$).

4.1 Worst Case

Intuitively, a worst case occurs when the partition routine produces a subproblem with $n-1$ elements and another with 0 elements. This situation can occur in each recursive call. Let $T(n)$ be the execution time of QUICKSORT for $A = [1, 2, \ldots, n]$. In that case, assuming $T(0) = 0$, we have

$$T(n) = T(n-1) + T(0) + kn = T(n-1) + kn.$$

Therefore $T(n) = \Theta(n^2)$ in this case.

Next, we will prove that, for the general case, the execution time of QUICKSORT is $O(n^2)$ and thus the previous case is indeed the worst case.

Let $T(n)$ be an upper bound on the execution time of QUICKSORT (in any case). We will prove by induction that $T(n) \leq kn^2$ for all $n \geq 0$. If $n = 0$ then, since we assume $T(0) = 0$, the inequality holds. Now suppose that $n > 0$. We have that,

$$\begin{aligned}
T(n) &\leq \max_{0 \leq q \leq n-1} \{T(q) + T(n-1-q)\} + kn \\
&\leq \max_{0 \leq q \leq n-1} \{kq^2 + k(n-q-1)^2\} + kn \\
&\leq k \cdot \max_{0 \leq q \leq n-1} \{q^2 + (n-q-1)^2\} + kn \\
&\leq k \cdot (n-1)^2 + kn \\
&\leq kn^2 - k(2n-1) + kn \\
&\leq kn^2.
\end{aligned}$$

Therefore, in the worst case, QUICKSORT has an execution time of $\Theta(n^2)$.

Exercise 4.1. *Prove that* $\max_{0 \leq q \leq n-1} \{q^2 + (n-q-1)^2\} = (n-1)^2$

4.2 Best Case

Intuitively, the best case occurs when both subproblems have approximately the same size ($(n-1)/2$). In that case we have

$$T(n) = T(\lceil (n-1)/2 \rceil) + T(\lfloor (n-1)/2 \rfloor) + kn$$

Therefore $T(n) = \Theta(n \lg n)$ in this case.

Next, we will prove that, in the general case, $T(n) = \Omega(n \lg n)$ and thus the previous case is indeed the best case.

Let $T(n)$ be a lower bound on the execution time of Quicksort (in any case). We will prove by induction that $T(n) \geq cn \lg n$ for $c = k/3$. If $n = 1$ then, since $T(1) \geq 0$, the inequality holds.

Now suppose that $n > 1$. We have that,

$$\begin{aligned}
T(n) &\geq \min_{0 \leq q \leq n-1} \{T(q) + T(n-1-q)\} + kn \\
&\geq \min_{0 \leq q \leq n-1} \{cq \lg q + c(n-q-1) \lg (n-q-1)\} + kn \\
&\geq c \cdot \min_{0 \leq q \leq n-1} \{q \lg q + (n-q-1) \lg (n-q-1)\} + kn \\
&= c \cdot (n-1) \lg (\frac{n-1}{2}) + kn \\
&= c(n-1) \lg (n-1) - c(n-1) + kn \\
&= cn \lg (n-1) - c \lg (n-1) - c(n-1) + kn \\
&\geq cn \lg (n/2) - c \lg (n-1) - c(n-1) + kn \\
&= cn(\lg n - 1) - c \lg (n-1) - c(n-1) + kn \\
&= cn \lg n - cn - c \lg (n-1) - c(n-1) + kn \\
&= cn \lg n - c(n + \lg (n-1) + (n-1)) + kn \\
&= cn \lg n - c(2n + \lg (n-1) - 1) + kn \\
&\geq cn \lg n - c(2n + \lg (n-1) - 1) + 3cn \\
&\geq cn \lg n + c(n - \lg (n-1) + 1) \\
&\geq cn \lg n.
\end{aligned}$$

Therefore, in the best case, Quicksort has an execution time of $\Theta(n \lg n)$.

Exercise 4.2. *Prove that* $\min_{0 \leq q \leq n-1} \{q \lg q + (n-1-q) \lg (n-1-q)\} = (n-1) \lg((n-1)/2)$

Exercise 4.3. *Give an example of a permutation of the first seven natural numbers* $[1 \ldots 7]$ *where* Quicksort *makes the fewest possible comparisons. Justify your answer. How many comparisons does* Quicksort *make in your example?*

4.3 Average Case

The execution time of Quicksort is dominated by the time of the Partition procedure.

Recall the PARTITION procedure:

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p - 1
3  for j = p to r - 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Figure 8: Taken from the book Cormen, Introduction to Algorithms

Notice that each time this procedure is called at most n times. Furthermore, each call to partition takes $O(r-p+1)$ execution time (number of times the for loop from lines 3–6 is executed). This execution time can also be expressed by the number of times line 4 is executed.

Let X be the random variable that counts the number of times line 4 of the PARTITION procedure is executed during the entire execution of QUICKSORT. We will show that $E[X] = O(n \lg n)$.

We denote by $z_1, z_2, \ldots z_n$ the elements of the array A, with $z_1 \leq z_2 \leq \cdots \leq z_n$. Also, for each i, j, let $Z_{ij} = \{z_i, \ldots, z_j\}$. And for each ij we define the following random variables:

$$X_{ij} = \begin{cases} 1 & \text{si } z_i \text{ es comparado con } z_j \\ 0 & \text{si } z_i \text{ no es comparado con } z_j \end{cases}$$

Note that

$$
\begin{aligned}
E[X] &= \sum_{1 \le i < j \le n} E[X_{ij}] \\
&= \sum_{1 \le i < j \le n} Pr[z_i \text{ compared to } z_j] \\
&= \sum_{1 \le i < j \le n} Pr[z_i \text{ o } z_j \text{ is the first chosen pivot of } Z_{ij}] \\
&= \sum_{1 \le i < j \le n} \frac{2}{j - i + 1} \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1} \\
&\le \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\lg n) \\
&= O(n \lg n)
\end{aligned}
$$