Chapter 1

Quicksort

1.1 Prerequisites

You need to be familiar with the divide-and-conquer paradigm, standard notations for expressing
the timecomplexity of an algorithm, like the big-Oh, big-Omega notations. Also needed, some
knowledge of elementary probability theory.

Recommended readings: Description of Quicksort in my 60-254 notes, Ch7 of your CLRS text.

1.2 Introduction

Quicksort is based on the divide-and-conquer paradigm. A call to Quicksort on the array segment
\(A[p..r]\) results in the following in-place partition of \(A[p..r]\):

\[
\begin{align*}
\leq A[q] & \quad > A[q]
\end{align*}
\]

(Lomuto-partition of the array segment \(A[p..r]\))

The next step is to call Quicksort recursively on the array segments \(A[p..q-1]\) and \(A[q+1..r]\)

Various partitioning schemes are available. The one shown below is due to Lomuto as given in your
CLRS text.

LOMOUTO-PARTITION\((A, p, r)\)

1. \(x \leftarrow A[r]\)
2. \(i \leftarrow p - 1\)
3. for \(j \leftarrow p\) to \(r - 1\) do
4. \hspace{1em} if \(A[j] \leq x\)
5. \hspace{2em} then \(i \leftarrow i + 1\)
6. \hspace{1em} exchange\((A[i], A[j])\)
1.2. INTRODUCTION

7 exchange($A[i+1], A[r]$)
8 return $i+1$

The above scheme maintains a 3-fold split of $A[p..r]$ with respect to the pivot element $A[r] = x$. At the beginning of the $j$-th iteration, all elements of the subarray $A[p..i]$ are at most equal to $x$; all elements of the subarray $A[i+1..j-1]$ are greater than $x$, while the status of the rest of the array elements are yet to be determined. From the initial value of $i$ it is clear that upon the first entry into the for-loop the first two subarrays are empty.

**Example 1** On the array segment $[2, 8, 7, 1, 3, 5, 6, 4]$, Lomuto’s scheme outputs the following array $[2, 1, 3, 4, 7, 5, 6, 8]$ and returns the index of the array element 4.

**Exercise 1** What are the loop invariants of the Lomuto partition? Use this to show that Lomuto partition is correct.

**Note 1** The exercises 7.1-1 to 7.1-4 of your CLRS textbook are trivial.

**Definition 1** A function $f(n) = \Theta(g(n))$ iff $f(n) = \Omega(g(n))$ and $f(n) = O(g(n))$

**Example 2** Let $f(n) = 5 \times n^2 + 3 \times n + 1$, then $f(n) = \Theta(n^2)$.

**Exercise 2** Prove the statement of the above example

**Exercise 3** Argue that if the input array is of size $n$ the time-complexity of Lomuto’s partition scheme is in $\Theta(n)$.

**Worst-case Analysis of Quicksort**

Let $T_w(n)$ be the worst-case running time of quicksort. Then

$$T_w(n) = \max_{0 \leq q \leq n-1} \{T_w(q) + T_w(n - q - 1)\} + \Theta(n), n \geq 1 \quad (1.1)$$

with $T_w(0) = 0$ and where $\Theta(n)$ represents a linear function that accounts for the time required to partition the array.

**Solution:** We suspect that $T_w(n)$ is in $O(n^2)$. So let’s assume that $T_w(n) \leq c \times n^2$ when $n \geq 1$, for some $c > 0$ ($c$ must be fixed more precisely; we will do this subsequently). Assuming inductively that the claim is true for values smaller than some fixed $n$, by substitution of the upper bounds for $T_w(q)$ and $T_w(n-1-q)$ in the right-hand side of the above recurrence, we get:

$$T_w(n) \leq \max_{0 \leq q \leq n-1} \{c \times q^2 + c \times (n - q - 1)^2\} + \Theta(n)$$
$$\leq c \times \max_{0 \leq q \leq n-1} \{q^2 + (n - q - 1)^2\} + \Theta(n)$$

If we treat $q$ as a continuous variable, the expression $q^2 + (n - q - 1)^2$ attains its maximum values at the end-points of the given range, as it represents a parabola that is downward-convex and becomes minimum in the middle of the interval. Thus

$$T_w(n) \leq c \times (n - 1)^2 + \Theta(n)$$
$$\leq c \times n^2 - c \times (2n - 1) + \Theta(n)$$
$$\leq c \times n^2$$
provided we choose $c$ large enough so that the term $-c(2n-1)$ dominates the contribution of the \( \Theta(n) \) term, which ensures that the last inequality holds.

Let’s make this more precise. Set \( an \) for the function that \( \Theta(n) \) represents, ignoring any lower-order terms of this function. Then, in order for the last inequality to hold, we must have $-c(2n-1)+an \leq 0$, or $c \geq a*n/(2n-1)$, provided $n \geq 1$. By choosing $c \geq a$, we can guarantee that $c \geq a*n/(2n-1)$ holds for all $n \geq 1$. Thus we have shown that \( T_w(n) \leq c*n^2 \), for $n \geq 1$ provided $c \geq a$.

\( T_w(n) = \Theta(n^2) \), because for each $n$ there is there is one partition sequence for which \( T_w(n) = \Omega(n^2) \)

Exercise 7.4-1 of CLRS asks you to prove this formally. Here is one possible solution.

**Solution:** From (1) above,

\[ T_w(n) \geq T_w(1) + T_w(n-2) + \Theta(n), \]

since the right-hand side is one of the $n$ values obtained by setting $q = 1$.

Assume once again that \( T_w(n) \geq c*n^2 \) for $n \geq 1$ and some $c > 0$ that we must determine. Then, \( T_w(n) \geq c + c * (n-2)^2 + a*n \), so that \( T_w(n) \geq c*n^2 + 5*c - 4*c*n + a*n \).

Now, \( T_w(n) \geq c*n^2 \) if \( 5*c - 4*c*n + a*n \geq 0 \) or \( c \leq a*n/(4n-5) \);

Since $a/4 \leq an/(4n-5)$ for $n \geq 2$, this means that the inequality will hold for $n \geq 2$ if we choose $c \leq a/4$.

We are not done yet. From (1), \( T_w(n) = a \) for $n = 1$. Thus in order for \( T_w(n) \geq c*n^2 \) to hold for $n = 1$, we must have $a \geq c$, which is true by the choice of $c$ in the previous paragraph.

Thus \( T_w(n) \geq c*n^2 \) for $n \geq 1$, provided $0 < c \leq a/4$.

\[ \square \]

**Summary:** \( T_w(n) = \Theta(n^2) \)

CLRS, Ex 7.4-2. Show that Quicksort’s best case running time is in \( \Omega(n \log n) \)

**Solution:** Let \( T_b(n) \) be the best-case running time of Quicksort. Then

\[ T_b(n) = \min_{0 \leq q \leq n-1} \{ T_b(q) + T_b(n-q-1) \} + \Theta(n) \]  

(1.2)

Now, we assume that \( T_b(n) \geq c*n \log n \) for $n \geq 1$ and some $c > 0$, and proceed exactly as we did when we were trying to establish an upper bound for the worst case.

We can also establish an upper bound on \( T_b(n) \) exactly as we did for the worst case. We provide a sketch of the proof. The details are to be filled in by you.

From (2) above, we have \( T_b(n) \leq T_b([n/2]) + T_b([n/2] - 1) + \Theta(n) \), by setting $q = [n/2]$.

Assume that \( T_b(n) \leq c*n \log n \). Then

\[ T_b(n) \leq c * ([n/2]) \log([n/2]) + c * ([n/2] - 1) \log([n/2] - 1) + \Theta(n) \]
\[ \leq 2c * (n/2) \log(n/2) + \Theta(n) \]
\[ \leq c * n \log n - c * n + \Theta(n) \]
\[ \leq c * n \log n \text{, if we choose } c \text{ large enough so that the term } -c*n \text{ dominates the } \Theta(n) \text{ term.} \]
1.3 Randomized Quicksort

Summary: \( T_b(n) = \Theta(n \log n) \)

CLRS, Ex 7.4-3.

Solution: You don’t need any calculus to show this. Let \( A = q \) and \( B = n - q - 1 \), then \( A^2 + B^2 = 1/2((A + B)^2 + (A - B)^2) \), which is minimum when \( A = B \) or \( q = (n - 1)/2 \).

1.3 Randomized Quicksort

In randomized quicksort, the index of the pivot or partitioning element is picked at random. We need to make a small modification to the procedure PARTITION:

RANDOMIZED-PARTITION\((A, p, r)\)
1. \( i \leftarrow \text{RANDOM}(p, r) \)
2. exchange\((A[r], A[i])\)
3. return LOMUTO-PARTITION\((A, p, r)\)

RANDOMIZED-QUICKSORT\((A, p, r)\)

1. If \( p < r \)
2. then \( q \leftarrow \text{RANDOMIZED-PARTITION}(A, p, r) \)
3. RANDOMIZED-QUICKSORT\((A, p, q - 1)\)
4. RANDOMIZED-QUICKSORT\((A, q + 1, r)\)

Analysis of Randomized Quicksort

The number of comparisons made by RANDOMIZED-QUICKSORT is a discrete random variable \( X \). We wish to determine the expected (or mean) value \( E[X] \) of \( X \).

Let \( x_1, x_2, \ldots, x_n \) be the sorted order of the array elements \( A[1], A[2], \ldots, A[n] \). Let \( p_{ij} \) be the probability that \( x_i \) and \( x_j \) are compared. We note that these never get compared if an array element \( x \) such that \( x_i < x < x_j \) is picked as pivot earlier than \( x_i \) or \( x_j \). These are compared otherwise. Since there are \( j - i + 1 \) elements lying between \( x_i \) and \( x_j \) inclusive, the probability \( p_{ij} \) is \( 2/(j - i + 1) \), since this is the probability that \( x_i \) or \( x_j \) is picked first rather than any intermediate element \( x \).

Let \( X = \sum_{i<j} X_{ij} \), where \( X_{ij} = 1 \) if \( x_i \) and \( x_j \) are compared and \( 0 \) otherwise. Now, by the linearity of the expectation operator \( E[] \),

\[
E[X] = \sum_{i<j} E[X_{ij}]
\]  \( \quad (1.3) \)

As \( E[X_{ij}] = p_{ij} = 2/(j - i + 1) \),

\[
E[X] = \sum_{i<j} p_{ij} = \sum_{i<j} 2/(j - i + 1) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 2/(j - i + 1)
\]
Thus $E[X] = O(n \log n)$ \qed

If $T_e(n)$ denotes the expected running time of Quicksort on an input of size $n$, then $T_e(n) = E[\Theta(n) + X] = E[an + X] = an + E[X] = an + O(n \log n)$, where the $\Theta(n)$ term accounts for the $n$ calls to PARTITION, each call costing $\Theta(1)$ time. Since $an$ is of lower order than $n \log n$, $T_e(n) = O(n \log n)$.

Exercise 4 Show that if $f(n) = O(n \log n)$ then $an + f(n) = O(n \log n)$.

Let us look at some problems from the CLRS textbook.

CLRS, Ex 7.4-4.

Solution: This can be solved as follows.

\[
E[X] = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{1}{k+1} \\
= \sum_{i=1}^{n-1} \left( \frac{1}{(k+1)} - \frac{1}{n-i+1} \right) \\
= (n-1) * \left( \sum_{k=1}^{n} \frac{1}{k+1} - \sum_{i=1}^{n-1} \frac{1}{n-i+1} \right) \\
\geq (n-1) * c * \log(n+1) - (n-1) - \left( (n+1)/(n+1) + n/n + (n-1)/(n-1) + \ldots + 2/2 + 1/1 \right).
\]

The second term on the right hand side of the last inequality is obtained this way. The fractions $1/(n-j+2), j = 1, 2, \ldots, n-1$, each occur $n-j$ times. Therefore, the last inequality holds as we are subtracting a larger amount from the first term.

Hence

\[
E[X] \geq (n-1) * c * \log(n+1) - (n-1) - (n+1) \\
\geq n * c * \log(n+1) - c * \log(n+1) - (n-1) - (n+1) \\
\geq n * c * \log(n+1) - c * \log(n+1) - 2 * n \\
\geq c * n * \log n - c * \log(n+1) - 2 * n \\
\geq c/2 * n * \log n + (c/2 * n * \log n - c * \log(n+1) - 2 * n) \\
\geq c/2 * n * \log n,
\]

since for a sufficiently large $n$ we can make sure that the second term of the penultimate inequality is positive.

The above result shows that $T_e(n) = \Omega(n \log n)$.

Summary: $T_e(n) = \Theta(n \log n)$

CLRS, Ex 7.4-5.

Solution: This problem is interesting. In this case,
1.3. RANDOMIZED QUICKSORT

\[ E[X] = \sum_{i \leq j-k+1} E[X_{ij}] \]
\[ = \sum_{i \leq j-k+1} \frac{2}{(j-i+1)} \]
\[ = \sum_{i=1}^{n-k+1} \sum_{j=i+k-1}^{n} \frac{2}{(j-i+1)} \]

Setting \( j - i = l \), we get

\[ E[X] = \sum_{i=1}^{n-k+1} \sum_{l=1}^{n-i} \frac{2}{(l+1)} \]
\[ = \sum_{i=1}^{n-k+1} \sum_{l=1}^{n-i} \frac{2}{(l+1)} - \sum_{i=1}^{n-k+1} \sum_{l=1}^{n-k+2} \frac{2}{(l+1)} \]
\[ \leq c \cdot (n-k+1) \cdot \log n - c \cdot (n-k+1) \cdot \log k \]
\[ \leq c \cdot \log(n/k) \]

The above was obtained by the following reasoning. Two elements \( x_i \) and \( x_j \) are not compared if they are separated by less than \( k-2 \) elements in the sorted order. This takes care of one part.

The expected number of comparisons to insertion-sort an array of size \( k \) can be worked out this way. Assume that the first \( j \) elements are in place; in order to put the \( j+1 \)-th element in place the expected number of comparisons that we will make is \((1 + 2 + \ldots + j)/j = (j+1)/2\). Thus the total expected number of comparisons is \( \sum_{j=1}^{k} \frac{j(j+1)}{2} = k(k+3)/4 = O(k^2) \).

If we assume that each subarray to be sorted using insertion sort is of size \( O(k) \), then the total time to sort \( O(n/k) \) subarrays each of size \( O(k) \), takes \( O(n/k \cdot k^2) \) that is \( O(nk) \) time in an expected sense.

Thus \( O(nk + n \log(n/k)) \) expected time, adding up both parts.

\textit{CLRS, Ex 7.4-6}.

\textbf{Solution:} This problem is straightforward. The probability estimate is

\[ \sum_{j=n\alpha}^{n(1-\alpha)} (j-1)(n-j)/nC3 \]

The argument for this is as follows. In order to have an \( \alpha \) to \( 1-\alpha \) split, the median of the three has to be an element with index \( j \) in the sorted order where \( j \) varies from the index \( n\alpha \) to the index \( n(1-\alpha) \). The other two elements are any one of the \( j-1 \) elements smaller than it and any one of the \( n-j \) elements greater than it. The number \( nC3 \) in the denominator stands for all possible choices of 3 distinct elements from \( n \) elements.