Algorithms and Data Structures:
Average-Case Analysis of Quicksort

10th October, 2014
Quicksort

Divide-and-Conquer algorithm for sorting an array. It works as follows:

1. If the input array has less than two elements, nothing to do . . . Otherwise, do the following partitioning subroutine: Pick a particular key called the pivot and divide the array into two subarrays as follows:

\[
\begin{array}{ccc}
\leq \text{pivot} & \text{piv.} & \geq \text{pivot}
\end{array}
\]

2. Sort the two subarrays recursively.
Quicksort Algorithm

Algorithm Quicksort \((A, p, r)\)
1. if \(p < r\) then
2. \(q \leftarrow\) Partition \((A, p, r)\)
3. Quicksort \((A, p, q - 1)\)
4. Quicksort \((A, q + 1, r)\)
Algorithm \textsc{Partition}(A, p, r)
1. \textit{pivot} \leftarrow A[r]
2. \textit{i} \leftarrow p - 1
3. \textbf{for } j \leftarrow p \textbf{ to } r - 1 \textbf{ do}
4. \hspace{1em} \textbf{if } A[j] \leq \textit{pivot} \textbf{ then}
5. \hspace{2em} \textit{i} \leftarrow \textit{i} + 1
6. \hspace{1em} \text{exchange } A[i], A[j]
7. \text{exchange } A[i + 1], A[r]
8. \textbf{return } \textit{i} + 1

Same version as [CLRS]
Analysis of Quicksort

- The size of an instance \((A, p, r)\) is \(n = r - p + 1\).
- Basic operations for sorting are **comparisons of keys**. We let \(C(n)\) be the **worst-case number of key-comparisons** performed by \textsc{Quicksort}(\(A, p, r\)). We shall try to determine \(C(n)\) as precisely as possible.

- It is easy to verify that the worst-case running time \(T(n)\) of \textsc{Quicksort}(\(A, p, r\)) is \(\Theta(C(n))\) if a single comparison requires time \(\Theta(1)\).

(i.e., for \textsc{Quicksort}, comparisons **dominate** the running time). In any case,

\[
T(n) = \Theta(C(n) \cdot \text{cost per comparison}).
\]
Analysis of \textsc{Partition}

\begin{itemize}
  \item \textsc{Partition}(A, p, r) does exactly \( n - 1 \) comparisons for every input of size \( n \).
  \item There is also an initial comparison \( p < r \) done on line 1 of \textsc{Quicksort}.
  \item Therefore in our theoretical analysis, we can assume, that apart from recursive calls, we do \( n \) comparisons on every input.
\end{itemize}
Worst-case Analysis of Quicksort

We get the following recurrence for $C(n)$:

$$C(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
\max_{1 \leq k \leq n} (C(k - 1) + C(n - k)) + n & \text{if } n \geq 2 
\end{cases}$$

Intuitively, worst-case seems to be $k = 1$ or $k = n$, i.e., everything falls on one side of the partition. This happens, e.g., if the array is sorted.
Worst-Case Analysis (cont’d)

- **Lower Bound**: \( C(n) \geq \frac{1}{2} n(n + 1) = \Omega(n^2) \).

  *Proof:*

  \[
  C(n) \quad \geq \quad C(n - 1) + n \\
  \quad \geq \quad C(n - 2) + n + n - 1 \\
  \vdots \\
  \sum_{i=0}^{n-1} i + 1 = \sum_{i=1}^{n} i = \frac{1}{2} n(n + 1).
  \]

- **Upper Bound**: \( C(n) \leq O(n^2) \).

  **CLASS?**

  - Thus  
    \[
    C(n) = \Theta(n^2).
    \]
Best-Case Analysis

- \( B(n) = \) number of comparisons done by \textsc{Quicksort} in the best case.

- \textit{Recurrence:}

\[
B(n) = \begin{cases} 
1 & \text{if } n \leq 1 \\
\min_{1 \leq k \leq n} \left( B(k - 1) + B(n - k) \right) + n & \text{if } n \geq 2
\end{cases}
\]

- Intuitively, the best case is if the array is always partitioned into two parts of the same size. This means

\[
B(n) \approx 2B(n/2) + \Theta(n),
\]

which implies \( B(n) = \Theta(n \log(n)) \).
Average-Case Analysis

- \( A(n) \) = number of comparisons done by **QuickSort** on average if all input arrays of size \( n \) are considered equally likely.

- **Intuition:** The average case is closer to the best case than to the worst case, because only **repeatedly very unbalanced** partitions lead to the worst case.

- **Recurrence:**

  \[
  A(n) = \begin{cases} 
    1 & \text{if } n \leq 1 \\
    \sum_{k=1}^{n} \frac{1}{n} (A(k - 1) + A(n - k)) + n & \text{if } n \geq 2
  \end{cases}
  \]

- **Solution:**

  \[
  A(n) \approx 2n \ln(n).
  \]
Average Case Analysis in Detail

We shall prove that for all $n \geq 1$ we have

$$A(n) \leq 2(\ln(n) + 1/3)(n + 1). \quad (\star)$$

Clearly, $(\star)$ holds for $n = 1$.

So assume that $n \geq 2$. We have

$$A(n) = \sum_{1 \leq k \leq n} 1_{\{A(k-1) + A(n-k)\}} + n = 2n - 1 \sum_{k=0}^{n} A(k) + n.$$

Thus

$$nA(n) = 2n - 1 \sum_{k=0}^{n} A(k) + n^2. \quad (\star\star)$$

Note that $(\star\star)$ does not hold for $n = 1$.

(For $n = 1$ the l.h.s. $A(1) = 1$, but the r.h.s. is $3$.)
We shall prove that for all $n \geq 1$ we have

$$A(n) \leq 2(\ln(n) + 1/3)(n + 1).$$

(\star)

Clearly, (\star) holds for $n = 1$. 

(For $n = 1$ the l.h.s. $1A(1) = 1$, but the r.h.s. is 3.)
Average Case Analysis in Detail

We shall prove that for all $n \geq 1$ we have

$$A(n) \leq 2(\ln(n) + 1/3)(n + 1).$$  (★)

Clearly, (★) holds for $n = 1$. So assume that $n \geq 2$. We have

$$A(n) = \sum_{1 \leq k \leq n} \frac{1}{n}(A(k - 1) + A(n - k)) + n$$

Note that (★★) does not hold for $n = 1$. (For $n = 1$ the l.h.s. $A(1) = 1$, but the r.h.s. is 3.)
Average Case Analysis in Detail

We shall prove that for all $n \geq 1$ we have

$$A(n) \leq 2(\ln(n) + 1/3)(n + 1). \quad (\star)$$

Clearly, $(\star)$ holds for $n = 1$. So assume that $n \geq 2$. We have

$$A(n) = \sum_{1 \leq k \leq n} \frac{1}{n} (A(k - 1) + A(n - k)) + n$$

$$= \frac{2}{n} \sum_{k=0}^{n-1} A(k) + n.$$
Average Case Analysis in Detail

We shall prove that for all \( n \geq 1 \) we have

\[
A(n) \leq 2(\ln(n) + 1/3)(n + 1). \tag{⋆}
\]

Clearly, (⋆) holds for \( n = 1 \). So assume that \( n \geq 2 \). We have

\[
A(n) = \sum_{1 \leq k \leq n} \frac{1}{n} (A(k - 1) + A(n - k)) + n
\]

\[
= \frac{2}{n} \sum_{k=0}^{n-1} A(k) + n.
\]

Thus

\[
nA(n) = 2 \sum_{k=0}^{n-1} A(k) + n^2. \tag{⋆⋆}
\]
Average Case Analysis in Detail

We shall prove that for all $n \geq 1$ we have

$$A(n) \leq 2(\ln(n) + 1/3)(n + 1). \quad (\star)$$

Clearly, $(\star)$ holds for $n = 1$. So assume that $n \geq 2$. We have

$$A(n) = \sum_{1 \leq k \leq n} \frac{1}{n} (A(k - 1) + A(n - k)) + n$$

$$= \frac{2}{n} \sum_{k=0}^{n-1} A(k) + n.$$

Thus

$$nA(n) = 2 \sum_{k=0}^{n-1} A(k) + n^2. \quad (\star\star)$$

Note that $(\star\star)$ does not hold for $n = 1$.
(For $n = 1$ the l.h.s. $A(1) = 1$, but the r.h.s. is 3.)
Applying (⋆⋆) to \((n - 1)\) for \(n \geq 3\), we obtain

\[
(n - 1)A(n - 1) = 2 \sum_{k=0}^{n-2} A(k) + (n - 1)^2.
\]
Applying (⋆⋆) to \((n - 1)\) for \(n \geq 3\), we obtain

\[(n - 1)A(n - 1) = 2 \sum_{k=0}^{n-2} A(k) + (n - 1)^2.\]

Subtracting this equation from (⋆⋆) (when \(n \geq 3\))

\[nA(n) - (n - 1)A(n - 1) = 2A(n - 1) + n^2 - (n - 1)^2,\]
Average Case Analysis in Detail (cont’d)

Applying (⋆⋆) to \((n − 1)\) for \(n \geq 3\), we obtain

\[
(n − 1)A(n − 1) = 2 \sum_{k=0}^{n-2} A(k) + (n − 1)^2.
\]

Subtracting this equation from (⋆⋆) (when \(n \geq 3\))

\[
nA(n) − (n − 1)A(n − 1) = 2A(n − 1) + n^2 − (n − 1)^2,
\]

thus

\[
nA(n) = (n + 1)A(n − 1) + 2n − 1,
\]
Average Case Analysis in Detail (cont’d)

Applying (⋆⋆) to \((n - 1)\) for \(n \geq 3\), we obtain

\[(n - 1)A(n - 1) = 2 \sum_{k=0}^{n-2} A(k) + (n - 1)^2.\]

Subtracting this equation from (⋆⋆) (when \(n \geq 3\))

\[nA(n) - (n - 1)A(n - 1) = 2A(n - 1) + n^2 - (n - 1)^2,\]

thus

\[nA(n) = (n + 1)A(n - 1) + 2n - 1,\]

and therefore

\[\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2n-1}{n(n+1)} \leq \frac{A(n-1)}{n} + \frac{2}{n}\]
Average Case Analysis in Detail (cont’d)

Applying (⋆⋆) to \((n - 1)\) for \(n \geq 3\), we obtain

\[(n - 1)A(n - 1) = 2 \sum_{k=0}^{n-2} A(k) + (n - 1)^2.\]

Subtracting this equation from (⋆⋆) (when \(n \geq 3\))

\[nA(n) - (n - 1)A(n - 1) = 2A(n - 1) + n^2 - (n - 1)^2,\]

thus

\[nA(n) = (n + 1)A(n - 1) + 2n - 1,\]

and therefore

\[\frac{A(n)}{n+1} = \frac{A(n-1)}{n} + \frac{2n-1}{n(n+1)} \leq \frac{A(n-1)}{n} + \frac{2}{n}\]

We now apply unfold-and-sum to this recurrence (stopping at \(n = 2\)):

\[\frac{A(n)}{n+1} \leq \frac{A(n-1)}{n} + \frac{2}{n}\]

\[\vdots\]
Average Case Analysis in Detail (cont’d)

\[
\frac{A(n)}{n+1} \leq \frac{A(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n-1}
\]
Average Case Analysis in Detail (cont’d)

\[
\frac{A(n)}{n+1} \leq \frac{A(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n-1} \\
\vdots \\
\leq \frac{A(2)}{3} + 2 \sum_{k=3}^{n} \frac{1}{k}
\]
Average Case Analysis in Detail (cont’d)

\[
\frac{A(n)}{n+1} \leq \frac{A(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n-1} \\
\vdots \\
\leq \frac{A(2)}{3} + 2 \sum_{k=3}^{n} \frac{1}{k} \\
= \frac{4}{3} + 2 \sum_{k=3}^{n} \frac{1}{k} = \frac{1}{3} + 2 \sum_{k=2}^{n} \frac{1}{k}.
\]
Average Case Analysis in Detail (cont’d)

\[
\frac{A(n)}{n + 1} \leq \frac{A(n - 2)}{n - 1} + 2 \frac{2}{n} + \frac{2}{n - 1}
\]

\[
\vdots
\]

\[
\leq \frac{A(2)}{3} + 2 \sum_{k=3}^{n} \frac{1}{k}
\]

\[
= \frac{4}{3} + 2 \sum_{k=3}^{n} \frac{1}{k} = \frac{1}{3} + 2 \sum_{k=2}^{n} \frac{1}{k}.
\]

It is easy to verify this result by induction. Thus

\[
\frac{A(n)}{n + 1} \leq \frac{1}{3} + 2 \sum_{k=2}^{n} \frac{1}{k} = \frac{1}{3} + 2 \sum_{k=1}^{n-1} \frac{1}{k + 1} \leq \frac{1}{3} + 2 \int_{1}^{n} \frac{1}{x} = \frac{1}{3} + 2 \ln(n).
\]

Multiplying by \((n + 1)\) completes the proof of (⋆).

\[
ADS: \text{lect 7 – slide 13 – 10th October, 2014}
\]
Improvements

- Use insertion sort for small arrays.
- Iterative implementation.

Main Question

Is there a way to avoid the bad worst-case performance, and in particular the bad performance on sorted (or almost sorted) arrays?

Different strategies for choosing the pivot-element help (in practice).

ADS: lect 7 – slide 14 – 10th October, 2014
Median-of-Three Partitioning

Idea: Use the median of the first, middle, and last key as the pivot.

Algorithm \texttt{M3Partition}(A, p, r)

1. exchange $A[(p + r)/2]$, $A[r - 1]$
5. \texttt{Partition}(A, p + 1, r - 1)

Note that \texttt{M3Partition}(A, p, r) only requires 1 more comparison than \texttt{Partition}(A, p, r)
Median-of-Three Partitioning (cont’d)

Algorithm \text{M3Quicksort}(A, p, r)
1. \textbf{if} \ p < r \textbf{ then}
2. \hspace{1em} q \leftarrow \text{M3Partition}(A, p, r)
3. \hspace{1em} \text{M3Quicksort}(A, p, q - 1)
4. \hspace{1em} \text{M3Quicksort}(A, q + 1, r)

In can be shown that the worst-case running time of \text{M3Quicksort} is still $\Theta(n^2)$, but at least in the case of an almost sorted array (and in most other cases that are relevant in practice) it is very efficient.
**Randomized Quicksort**

**Idea:** Use key of random element as the pivot.

**Algorithm** $R_{\text{Partition}}(A, p, r)$

1. $k \leftarrow \text{Random}(p, r)$ \hspace{1em} ▷ choose $k$ randomly from $\{p, \ldots, r\}$
2. exchange $A[k], A[r]$
3. $\text{Partition}(A, p, r)$

**Algorithm** $\text{Randomized Quicksort}(A, p, r)$

1. if $p < r$ then
2. 
3. $q \leftarrow R_{\text{Partition}}(A, p, r)$
4. $\text{Randomized Quicksort}(A, p, q - 1)$
5. $\text{Randomized Quicksort}(A, q + 1, r)$
Analysis of Randomized Quicksort

The running time of Randomized Quicksort on an input of size \( n \) is a random variable.

An analysis similar to the average case analysis of Quicksort shows:

**Theorem**
For all inputs \((A, p, r)\), the expected number of comparisons performed during a run of Randomized Quicksort on input \((A, p, r)\), is at most \(2(\ln(n) + 1/3)(n + 1)\), where \( n = r - p + 1 \).

**Corollary**
Thus the expected running time of Randomized Quicksort on any input of size \( n \) is \( \Theta(n \lg(n)) \).
Reading Assignment

Sections 7.2, 7.3, 7.4 of [CLRS] (edition 2 or 3)

Problems

1. Convince yourself that Partition works correctly by working a few examples, or (better) try to prove that it works correctly. Find an example that shows what may go wrong if we omit the test $j > p$ in Line 6. Is this test also needed when we call Partition from M3Partition?

2. In our proof of the Average-running time $A(n)$, we can think of the input as being some permutation of $(1, \ldots, n)$, and assume all permutations are equally likely. Why does this explain the $1/n$ factor in the recurrence on slide 10?

3. Show that if the array is initially in decreasing order, then the running time is $\Theta(n^2)$.

ADS: lect 7 – slide 19 – 10th October, 2014