Quicksort

David Kauchak

- Quicksort

QUICKSORT(A, p, r)
1  if p < r
2    q ← PARTITION(A, p, r)
3    QUICKSORT(A, p, q - 1)
4    QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)
1  i ← p - 1
2  for j ← p to r - 1
4        i ← i + 1
5      swap A[i] and A[j]
6  swap A[i + 1] and A[r]
7  return i + 1

- Is it correct?
  Loop invariant: Elements in the subarray A[p...i] are all less than or equal to A[r] and elements in the subarray A[i + 1...j - 1] are all greater than A[r]

Proof by induction:
Base case: i = p - 1, so A[p...i] is empty and j = p and i + 1 = p, so A[i + 1...j - 1] is also empty.

Inductive case: We'll assume that the invariant is true for iteration j and show that iteration j + 1 is also true. There are two cases based on line the if statement in line 4.
1. If $A[j] > A[r]$ the only thing that happens is that $j$ is incremented. This means that $A[p...i]$ remains unchanged and will still contain elements that are less than or equal to $A[r]$. $A[i+1...j]$ will consist of $A[i+1...j-1]$, which contains elements greater than $A[r]$ (by induction), and one additional element $A[j]$ which we know is greater than $A[r]$, so we know the entire subarray $A[i+1...j]$ contains elements that are greater than $A[r]$.

2. If $A[j] \leq A[r]$ then two things happen. $i$ is incremented and $A[i]$ is swapped with $A[j]$. $A[p...i]$ will then contain the elements $A[p...i-1]$, which we already know are less than or equal to $A[r]$, and element $A[j]$, which is also less than or equal to $A[r]$. Subarray $A[i+1...j]$ will contain the same elements, except the last element, $A[j]$, will be the old first element, $A[i+1]$, and the other elements will be shifted down.

At termination, what does this tell us about the PARTITION procedure?

If PARTITION is correct, is QUICKSORT correct?

– Running time?

What is the running time of PARTITION?

Iterates over each element of the array and does at most a constant amount of work for each iteration: $\Theta(n)$

**Running time of QUICKSORT**

* Worst case: Array is sorted (or reverse sorted) and each call to partition subdivides the array into a subarray of length $n-1$ and a subarray of length 0.

**Draw the tree**

\[
T(n) = T(n-1) + T(0) + \Theta(n)
\]

which we’ve seen before: $\Theta(n^2)$

* Best case: The partition algorithm splits the array into two equal (or nearly) equal halves, e.g. 11 elements into two subarrays of length 5 or 10 elements into a subarray of length 4
and a subarray of length 5.

Draw the tree

\[ T(n) \leq 2T(n/2) + \Theta(n) \]

which we have also seen before with MERGE-SORT: \( \Theta(n \log n) \)

* Average case: Intuition 1

How balanced do the splits have to be to maintain the \( \Theta(n \log n) \) running time?

Say the PARTITION procedure always splits the array into constant ratio \( b \)-to-\( a \), e.g. 9-to-1.

\[ T(n) \leq T\left(\frac{a}{a+b}n\right) + T\left(\frac{b}{a+b}n\right) + cn \]

Recursion tree: Level 0: \( cn \)
Level 1: \( cn\left(\frac{a}{a+b}\right) + cn\left(\frac{b}{a+b}\right) = cn \)
Level 2: \( cn\left(\frac{a^2}{(a+b)^2}\right) + cn\left(\frac{ab}{(a+b)^2}\right) + cn\left(\frac{b^2}{(a+b)^2}\right) = cn\frac{a^2 + 2ab + b^2}{(a+b)^2} = cn \)
Level 3 \( cn\left(\frac{(a+b)^2a + (a+b)^2b}{(a+b)^4}\right) = cn\frac{(a+b)(a+b)^2}{(a+b)^4} = cn \)
Level d: \( cn\frac{(a+b)^d}{(a+b)^d} \leq cn \)

What is the depth of the tree?
What is the minimum depth of the tree?

Assume \( a < b \).

\[
\left(\frac{a}{a+b}\right)^d n = 1
\]
\[
\log\left(\frac{a}{a+b}\right)^d n = \log 1
\]
\[
\log n + \log\left(\frac{a}{a+b}\right)^d = 0
\]
\[
\log n + d \log\left(\frac{a}{a+b}\right) = 0
\]
\[ d \log \left( \frac{a}{a+b} \right) = -\log n \]
\[ d = \frac{-\log n}{\log \left( \frac{a}{a+b} \right)} \]
\[ d = \frac{\log n}{\log \left( \frac{a+b}{a} \right)} \]
\[ d = \log_{\frac{a+b}{a}} n \]

What is the maximum depth of the tree?

\[ d = \log_{\frac{a+b}{a}} n \]

Runtime: Each level has a cost of at most \( cn \) with maximum depth \( d = \log_{\frac{a+b}{a}} n \): \( O(n \log_{\frac{a+b}{a}} n) \)

Why not \( \Theta(n \log_{\frac{a+b}{a}} n) \)?

* Average case: Intuition 2

What would happen if half the time \textsc{Partition} produced a “bad” split of parts sized 0 and \( n - 1 \) and the other half of the time it produced a “good” split of equal sized parts?

Draw the trees for these two cases.

Cost for the 50/50:
Partition cost = \( \Theta(n) \)
Recursion = \( T\left(\frac{n-1}{2}\right) + T\left(\frac{n-1}{2}\right) \)

\[ T(n) = 2T\left(\frac{n-1}{2}\right) + \Theta(n) \]

Cost of “bad” followed by 50/50:
Partition cost = \( \Theta(n) + \Theta(n-1) = \Theta(n) \)
Recursion = \( T(0) + T\left(\frac{n-1}{2} - 1\right) + T\left(\frac{n-1}{2}\right) \)

\[ T(n) = T\left(\frac{n-1}{2} - 1\right) + T\left(\frac{n-1}{2}\right) + \Theta(n) \]

The cost of the “bad” partition is absorbed. In general, any constant number of “bad” partitions intermixed with “good” partitions will still result in \( O(n \log n) \) runtime.

* \textsc{Randomized-Quicksort}
How can we avoid the worst case situation for Quicksort?

**RANDOMIZED-PARTITION**(A, p, r)

1. \( i \leftarrow \text{RANDOM}(p, r) \)
2. swap \( A[r] \) and \( A[i] \)
3. return\( \text{PARTITION}(A, p, r) \)

* Analysis of RANDOMIZED-QUICKSORT: Expected running time

How many calls to \( \text{PARTITION} \) are made for an input of size \( n \)?

\( n \) - Each time a pivot element is selected and that element is never selected again.

What is the cost of an individual call to \( \text{PARTITION} \)?

Proportional to the number of iterations of the for loop.

Therefore, if we count the number of comparisons made (if \( A[j] \leq A[r] \)) then this is a bound on the running time of QUICKSORT.

**Counting the number of comparisons:**

Don’t try and analyze each call, but analyze the global number of comparisons.

Let \( z_i \) of \( z_1, z_2, ..., z_n \) be the \( i \)th smallest element and \( Z_{ij} \) be the set of elements \( Z_{ij} = z_i, z_{i+1}, ..., z_j \) between \( z_i \) and \( z_j \).

For example, if \( A = [3, 9, 7, 2] \) then, \( z_1 = 2, z_2 = 3, z_3 = 7, z_4 = 9 \) and \( Z_{24} = \{3, 7, 9\} \).

Let \( X_{ij} = I\{z_i \text{ is compared to } z_j\} = \begin{cases} 
1 & \text{if } z_i \text{ is compared to } z_j \\
0 & \text{otherwise}
\end{cases} \)

(indicator random variable)

How many times can \( z_i \) and \( z_j \) be compared? - At most once, since for a comparison to happen, one of the two must be the pivot, after which it is not included in recursive calls.

\[
X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}
\]
i.e., the total number of comparisons (and a bound on the overall runtime) - \(O(n + X)\), where \(n\) is for the calls to \textsc{Partition} and \(X\) for each iteration in \textsc{Partition}.

Remember, we want to know what the expected (on average) running time:

\[
E[X] = E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p\{z_i \text{ is compared to } z_j\}
\]

The pivot element separates the set of numbers into two sets (those less than the pivot and those larger). Elements from one set will \textit{never} be compared to elements of the other set.

If a pivot \(x\) is chosen \(z_i < x < z_j\), then \(z_i\) and \(z_j\) will not be compared.

Similarly, from the set \(Z_{ij}\), the only time \(z_i\) and \(z_j\) will be compared is if either \(z_i\) or \(z_j\) is chosen as a pivot. Why?

\[
p\{z_i \text{ is compared to } z_j\} = p\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\}
\]

\[
= p\{z_i \text{ is first pivot chosen from } Z_{ij}\} + p\{z_j \text{ is first pivot chosen from } Z_{ij}\}
\]

\[
= \frac{1}{j - i + 1} + \frac{1}{j - i + 1}
\]

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

Line 2: Independent events \((p(a, b) = p(a) + p(b) \text{ if } a \text{ and } b \text{ are independent events})\)

Line 3: Because the pivot is chosen randomly and there are
$j - i + 1$ elements in the set $Z_{ij}$

Let $k = j - i$:

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j - i + 1}$$

$$= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k + 1}$$

$$< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$= \sum_{i=1}^{n-1} O(\log n)$$

$$= O(n \log n)$$

where line 4 occurs because $\sum_{k=1}^{n} 2/k = \ln n + O(1) = O(\log n)$

Can a run of RANDOMIZED-QUICKSORT take time $\Theta(n^2)$?

- Memory usage?
- Ease of implementation?
- How does randomized quicksort compare to mergesort?

• Comparison based sorting
  Asks the question is $i \leq j$.

We’ve seen MERGE-SORT and randomized QUICKSORT which both run on average in time $\Theta(n \log n)$. Can we do better?

**Decision tree model**

Picture

- A binary tree where each node represents comparison between two elements, $i$ and $j$
- The branches are labeled with the decision outcome
Each leaf contains a permutation of the original data representing the sorted order.

To determine the correct output for a given input, follow the path based on the decisions from the root to a leaf node.

How many leaf nodes are there for a decision tree representing the sorting of $n$ elements? $n!$, all possible permutation of the original $n$ elements.

Why can’t there be less?

What is the height of the tree?

Binary tree of height $h$ contains $2^h$ leaves so,

\[
2^h = n!
\]

\[
\log 2^h = \log n!
\]

\[
h = \Omega(n \log n)
\]

using Stirling’s approximation,

\[
n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)
\]

• Other uses/sources of randomness in algorithms
  
  – Contention resolution
  – Algorithm initialization (e.g. clustering)
  – Game playing, i.e. inherent randomness in the interaction

• Sorting in linear time
  
  Counting sort

  Radix sort

These notes are adapted from material found in chapters 7,8 of [1].

References