Quicksort

Textbook, Chapter 8

CSCE310: Data Structures and Algorithms
www.cse.unl.edu/~choueiry/S01-310/
Berthe Y. Choueiry (Shu-we-ri)
Ferguson Hall, Room 104
choueiry@cse.unl.edu, Tel: (402)472-5444
Quicksort

- Worst-case running time is in $\Theta(n^2)$ \textit{(relatively slow)}
- Remarkably efficient: average running time is in $\Theta(n \lg n)$
  constants factors hidden in $\Theta(n \lg n)$ quite small
- Sorts in place (no need for external storage)
**Quicksort**: divide and conquer

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

Consider a subarray \( A[p \ldots r] \)

**Divide**: \( A[p \ldots r] \) is partitioned into \( A[p \ldots q] \) and \( A[q + 1 \ldots r] \) such that each element in \( A[p \ldots q] \) is smaller or equal to each element in \( A[q + 1 \ldots r] \)

Index \( q \) is computed here

**Conquer**: Subarrays \( A[p \ldots q] \) and \( A[q + 1 \ldots r] \) sorted by recursive call to **Quicksort**

**Combine**: Since subarrays sorted in place, no need to combine them!

\( A[p \ldots r] \) is sorted

Key to **Quicksort** is the **Partition** procedure
Partition procedure: rearranges $A[p \ldots r]$ in place

Puts elements smaller than $x$ in bottom region of array
Puts elements larger than $x$ in top region

\begin{verbatim}
PARTITION($A, p, r$)
1   $x \leftarrow A[p]$
2   $i \leftarrow p - 1$
3   $j \leftarrow r + 1$
4   while TRUE
5       do repeat $j \leftarrow j - 1$
6       until $A[j] \leq x$
7       repeat $i \leftarrow i + 1$
8       until $A[i] \geq x$
9       if $i < j$
10          then exchange $A[i] \leftrightarrow A[j]$
11          else return $j$
\end{verbatim}
Partition procedure: example

- First, selects $x = A[p]$ as a pivot, around which to partition $A[p \ldots r]$
- Then, grows two regions $A[p \ldots i]$ and $A[j \ldots r]$, from top and bottom of $A[p \ldots r]$ such that
  - every element in $A[p \ldots i] \leq x$ and
  - every element in $A[j \ldots r] \leq x$
- Initially, the regions are empty ($i = p - 1$ and $j = r + 1$)
Partition procedure: example

- Then, \( i \) is incremented and \( j \) is decremented until \( A[i] \geq x \geq A[j] \)
- At this point, \( A[i] \) is too big and \( A[j] \) is too small to be in respective region
  They should be swapped
  which allows us to continue extending the regions
- This continues until \( i \geq j \), with \( A[p \ldots r] \) partitioned in
  \( A[p \ldots q] \) and \( A[q + 1 \ldots r] \), where
  no element of \( A[p \ldots q] \) is larger than no element of \( A[q + 1 \ldots r] \)
Warning: \( A[p] \) must be chosen as pivot \( x \)

Choose \( A[r] \) as pivot, and suppose \( A[r] \) is largest element in \( A \)

What does \textbf{Partition} return?

\begin{verbatim}
\textbf{PARTITION}(A, p, r)
1 \hspace{1em} x \leftarrow A[p]
2 \hspace{1em} i \leftarrow p - 1
3 \hspace{1em} j \leftarrow r + 1
4 \hspace{1em} \textbf{while} \ TRUE
5 \hspace{2em} \textbf{do} \hspace{1em} \textbf{repeat} \hspace{1em} j \leftarrow j - 1
6 \hspace{3em} \textbf{until} \ A[j] \leq x
7 \hspace{2em} \textbf{repeat} \hspace{1em} i \leftarrow i + 1
8 \hspace{3em} \textbf{until} \ A[i] \geq x
9 \hspace{2em} \textbf{if} \ i < j
10 \hspace{3em} \textbf{then} \hspace{1em} \text{exchange} \ A[i] \leftrightarrow A[j]
11 \hspace{3em} \textbf{else} \hspace{1em} \text{return} \ j
\end{verbatim}

what was it supposed to return?
what happens to \textbf{Quicksort}?
Partition: running time

Running time on array $A[p \ldots r]$ is in $\Theta(n)$

1. $x \leftarrow A[p]
2. i \leftarrow p - 1
3. j \leftarrow r + 1
4. while TRUE
5. do repeat $j \leftarrow j - 1$
6. until $A[j] \leq x$
7. repeat $i \leftarrow i + 1$
8. until $A[i] \geq x$
9. if $i < j$ then exchange $A[i] \leftrightarrow A[j]$
else return $j$
10. 
11. 

Exercise 8.1-1
Choice of pivot

There are various methods that can be used to pick the pivot element, including:

- Use leftmost element as the pivot.
- Use the “median-of-three” rule to pick the pivot.
  - Choose as partitioning: \( \text{median}(a[p], a[(p + r)/2], a[r]) \)
  - Partitioning unlikely to generate a “degenerate” partition.
- Use a random element as the pivot
  - yields randomized-partition
  - The average complexity does not depend on the distribution of input sequences
Performance of Quicksort

Is partitioning balanced or unbalanced?
  Depends on which elements are used for partitioning
If balanced, Quicksort runs asymptotically as fast as Mergesort
If unbalanced, Quicksort runs asymptotically as slow as Insertion-sort

1. Worst-case partitioning
   Runs in $\Theta(n^2)$

2. Best-case partitioning
   Runs in $\Theta(n \log n)$

3. Average (Balanced) partitioning
   Runs in $\Theta(n \log n)$
Worst-case partitioning

Partition produces 2 regions:

(1) 1 element
(2) \((n - 1)\) elements

Suppose this happens at every step
(When does this happen?)

But, since partitioning costs \(\Theta(n)\) and \(T(1) = \Theta(1)\), we have

\[
T(n) = T(n - 1) + \Theta(n) = \sum_{k=1}^{n} \Theta(k) = \Theta(\sum_{k=1}^{n} k) = \Theta(n^2)
\]

Worst-case of Quicksort is as bad as that of Insertion-sort
And this happens in cases where Insertion-sort is linear :— (}
Worst-case partitioning: Recurrence tree
Best-case partitioning

Partition produces two regions, each of size $n/2$

Recurrence is: $T(n) = 2T(n/2) + \Theta(n)$

Apply Case 2 of Master Theorem
Solution: $T(n) = \Theta(n \lg n)$
Best-case partitioning: Recurrence tree

\[ \Theta(n \log n) \]
Average-case partitioning

Average case much closer to best case than to worst case

Balanced partitioning: Suppose \texttt{Partition} always produces 9-to-1 proportional split

Recurrence is: $T(n) = T(9n/10) + T(n/10) + n$

Look at recurrence tree..
Every level has cost \( n \), until reaching boundary condition at depth

\[
\log_{10} n = \Theta(\lg n)
\]

Then levels have cost at most \( n \), until recursion terminates at depth

\[
\log_{10/9} n = \Theta(\lg n)
\]
Constant proportionality

Thus, with 9-to-1 proportional split, quicksort runs is in $\Theta(n \lg n)$.

It can be shown that the solution is $T(n) = \Theta(n \lg n)$, whenever the split has constant proportionality.
Performance of Quicksort

Courtesy of C. Cusack

Pivot at position $i$: $T(n) = T(n-i) + T(i-1) + n$

$i$ may be different for each subarray, and at each level of recursion

**Best case:** $T(n) \leq 2T(n/2) + n$.

Solution: $T(n) = \Theta(n \log n)$.

**Worst case:**

\[
T(n) = T(n-1) + n = n + (n-1) + \ldots + 1 = n(n+1)/2 = O(n^2)
\]

**Average case:**

\[
T_a(n) = n + \frac{1}{n} \sum_{i=1}^{n} (T_a(i-1) + T_a(n-i))
\]
Average Case

- We assume that the pivot has the same probability \(1/n\) to go into each of the \(n\) possible positions. This gives

\[
T_a(n) = n + \frac{1}{n} \sum_{i=1}^{n} (T_a(i - 1) + T_a(n - i)) \\
= n + \frac{1}{n} \sum_{i=1}^{n} T_a(i - 1) + \frac{1}{n} \sum_{i=1}^{n} T_a(n - i) \\
= n + \frac{2}{n} \sum_{i=1}^{n} T_a(i - 1)
\]

- The last step comes from the fact that the two sums are the same, but in reverse order.

- We have seen that in the best case, 
  \(T(n) = O(n \log n)\) and in the worst case, 
  \(T(n) = O(n^2)\).

- We guess that \(T_a(n) \leq an \log n + b\), for two positive constants \(a\) and \(b\).

- We will prove this by induction.
Proof: \( T(n) \leq an \log n + b \)

*Courtesy of C. Cusack*

- We can pick \( a \) and \( b \) so that the condition holds for \( T(1) \).
- Assume it holds for all \( k < n \). Then

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

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

\[
\leq n + \frac{2}{n} \sum_{k=1}^{n-1} (ak \log k + b)
\]

\[
= n + \frac{2b}{n} (n - 1) + \frac{2a}{n} \sum_{k=1}^{n-1} k \log k
\]
• It can be shown that

\[
\sum_{k=1}^{n-1} k \log k \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2.
\]

• Substituting, we get

\[
T_a(n) = n + \frac{2b}{n} (n - 1) + \frac{2a}{n} \sum_{k=1}^{n-1} k \log k
\]

\[
\leq n + \frac{2b}{n} (n - 1) + \frac{2a}{n} \left( \frac{1}{2} n^2 \log n - \frac{1}{8} n^2 \right)
\]

\[
= n + 2b - \frac{2b}{n} + an \log n - \frac{a}{4} n
\]

\[
= an \log n + b + (n + b - \frac{2b}{n} - \frac{a}{4} n)
\]

\[
\leq an \log n + b + (n + b - \frac{a}{4} n)
\]

\[
\leq an \log n + b.
\]

• The last step can be obtained by choosing \(a\) large enough.

• Thus, \(T(n) = O(n \log n)\).