CSE 373 Design and Analysis of Algorithms

Mirza Mohammad Lutfe Elahi

Department of Electrical and Computer Engineering
North South University

Sorting - Quicksort
Chapter 07
Description of Quick Sort

Quicksort, like merge sort, applies the divide-and-conquer paradigm.

- **Divide**: Partition (rearrange) the array $A[p..r]$ into two (possibly empty) subarrays $A[p..q-1]$ and $A[q+1..r]$ such that each element of $A[p..q-1]$ is less than or equal to $A[q]$, which is, in turn, less than or equal to each element of $A[q+1..r]$. Compute the index $q$ as part of this partitioning procedure.

- **Conquer**: Sort the two subarrays $A[p..q-1]$ and $A[q+1..r]$ by recursive calls to quicksort.

- **Combine**: Because the subarrays are already sorted, no work is needed to combine them: the entire array $A[p..r]$ is now sorted.
Algorithm - Quick Sort

\textbf{QuickSort}(A, p, r)

1. \textbf{if} \ p \ < \ r
2. \hspace{1em} q = \text{Partition}(A, p, r)
3. \hspace{1em} \text{QuickSort}(A, p, q - 1)
4. \hspace{1em} \text{QuickSort}(A, q + 1, r)
Algorithm - Partitioning

\textbf{Partition}(\textit{A, p, r})

1. \( x = A[r] \)
2. \( i = p - 1 \)
3. \textbf{for} \( j = p \) \textbf{to} \( r - 1 \)
4. \textbf{if} \( A[j] \leq x \)
5. \hspace{1em} \( i = i + 1 \)
6. \hspace{1em} \text{exchange} \ A[i] \text{ with} \ A[j] \)
7. \text{exchange} \ A[i + 1] \text{ with} \ A[r] \)
8. \textbf{return} \( i + 1 \)

At the beginning of each iteration of the loop of lines 3 – 6, for any array index \( k \),

1. If \( p \leq k \leq i \), then \( A[k] \leq x \).
2. If \( i + 1 \leq k \leq j - 1 \), then \( A[k] > x \).
3. If \( k = r \), then \( A[k] = x \).
The operation of PARTITION

Figure: Array entry $A[r]$ becomes the pivot element $x$. Lightly shaded array elements are all in the first partition with values no greater than $x$. Heavily shaded elements are in the second partition with values greater than $x$. The unshaded elements have not yet been put in one of the first two partitions, and the final white element is the pivot $x$. 
The operation of PARTITION

Figure: The four regions maintained by the procedure PARTITION on a subarray $A[p..r]$. The values in $A[p..i]$ are all less than or equal to $x$, the values in $A[i+1..j-1]$ are all greater than $x$, and $A[r] = x$. The subarray $A[j..r-1]$ can take on any values.
Performance of Quick Sort

- The running time of quicksort depends on whether the partitioning is balanced or unbalanced.
- It depends on which elements are used for partitioning.
- For balanced partition, the algorithm runs asymptotically as fast as merge sort.
- For unbalanced partition, however, it can run asymptotically as slowly as insertion sort.
Worst-case Partitioning

- The partitioning routine produces one subproblem with \( n - 1 \) elements and one with 0 elements.
- Assume that this unbalanced partitioning arises in each recursive call.
- The partitioning costs \( \Theta(n) \) time.
- The recursive call on an array of size 0 just returns, \( T(0) = \Theta(1) \)
- The recurrence for running time:

\[
T(n) = T(n - 1) + T(0) + \Theta(n) \\
= T(n - 1) + \Theta(n)
\]
If we sum the costs incurred at each level of the recursion, we get an arithmetic series, which evaluates to $\Theta(n^2)$.

The worst-case running time of quicksort is no better than that of insertion sort.

Moreover, the $\Theta(n^2)$ running time occurs when the input array is already completely sorted.
Best-case Partitioning

- In the most even possible split, \textsc{Partition} produces two subproblems.
- Each of size no more than $n/2$, since one is of size $\lceil n/2 \rceil$ and one of size $\lfloor n/2 \rfloor - 1$.
- Quick sort runs much faster.
- The recurrence for running time:

$$T(n) = 2T(n/2) + \Theta(n)$$

$$= \Theta(n \log n)$$

- By equally balancing the two sides of the partition at every level of the recursion, we get an asymptotically faster algorithm.
The average-case running time of quicksort is much closer to the best case than to the worst case.

Suppose, for example, that the partitioning algorithm always produces a 9-to-1 proportional split.

Then the recurrence for running time:

\[ T(n) = T(9n/10) + T(n/10) + cn \]
Balanced Partitioning

Every level of the tree has cost \( cn \), until the recursion reaches a boundary condition at depth \( \log_{10} n = \Theta(lgn) \), and then the levels have cost at most \( cn \).

The recursion terminates at depth \( \log_{10/9} n = \Theta(lgn) \)

The total cost of quicksort is therefore \( O(n \log n) \)
In the average case, \textsc{Partition} produces a mix of "good" and "bad" splits.

Suppose, for the sake of intuition, that the good and bad splits alternate levels in the tree, and that the good splits are best-case splits and the bad splits are worst-case splits.
Average-case Partitioning

- Let's assume that the boundary-condition cost is 1 for the subarray of size 0.
- The combination of the bad split followed by the good split produces three subarrays of sizes 0, \((n - 1)/2 - 1\), and \((n - 1)/2\) at a combined partitioning cost of \(\Theta(n) + \Theta(n - 1) = \Theta(n)\).
- This situation is no worse than the other that is a single partitioning that produces two subarrays of size \((n - 1)/2\), at a cost of \(\Theta(n)\).
- Intuitively, the \(\Theta(n - 1)\) cost of the bad split can be absorbed into the \(\Theta(n)\) cost of the good split, and the resulting split is good.
- The running time of quicksort, when levels alternate between good and bad splits, is like the running time for good splits alone: still \(O(n \log n)\).