Quicksort is a well-known **divide-and-conquer** algorithm with a variety of applications for its fast sorting of large data sets. The concept at the heart of quicksort is to reduce a large sorting problem into smaller ones that can be solved recursively and combine the results to obtain the sorted final array. The whole task is completed by the key element named a **pivot** which is the core element of arranging the array elements when we are sorting an array.

To start the quicksort algorithm, a **pivot element** is chosen from the array. The pivot to be chosen is key as it decides how well the list is broken. After selecting the pivot, the array is **partitioned** in two arrays: one array with elements that are smaller than the pivot and other array has elements that are greater or equal to the pivot. This step places the pivot in its final sorted position in the array. The procedure is repeated on the two sub-arrays until all elements are in order.

There are numerous ways for **pivot selection**, and the decision can considerably affect the effectiveness of the algorithm. A frequent and straightforward way is to select the first or final entry of the array as the pivot. However, this can lead to poor performance ($O(n^2)$) on previously sorted or almost sorted arrays. A superior option is the **median-of-three method**, where the pivot is determined as the median of the first, middle, and last members of the array. This method helps avoid worse-case performance circumstances by assuring a more balanced division of the data (Sedgewick & Wayne, 2011). Some advanced versions even use randomized pivot selection to limit the chance of meeting worst-case input sequences.

The value of the pivot in quicksort cannot be emphasized. A well-chosen pivot leads to balanced partitions, resulting in an ideal temporal complexity of **$O(n \log n)$**. On the other hand, a badly designed pivot might lead to unbalanced partitions, lowering the performance to **$O(n^2)$**. Thus, the effectiveness of the algorithm hinges on how effectively the pivot divides the array.

After the pivot is selected and the array is partitioned, the quicksort algorithm **recursively sorts** the two resulting sub-arrays. Because each step reduces the issue size, the method quickly converges on a solution. One of the primary advantages of quicksort is that it sorts in place, using only a little, constant amount of additional memory space (Cormen et al., 2009). This makes quicksort not only fast but also space-efficient compared to other sorting algorithms like merge sort.

In summary, quicksort exhibits the divide and conquer technique by utilizing a pivot to separate the array into smaller sections, which it then sorts recursively. The pivot's selection is crucial for performance, as it directly affects the balance of the partitioning process. With excellent pivoted choices, quicksort performs effectively on a wide variety of data sets, making it a preferred technique in practical applications.

References:

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. https://mitpress.mit.edu/9780262533058/introduction-to-algorithms/

Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley. https://www.amazon.com/Algorithms-4th-Robert-Sedgewick/dp/032157351X

Wordcount: 477