Quicksort algorithm

***Source:* https://github.com/rusevrosen/codehub.github.io/tree/main/Algorithms/Sorting_Algorithms**

1. What is the Quicksort algorithm?

Quicksort is a divide-and-conquer algorithm that sorts a list (or array) of items. It works by selecting a 'pivot' element from the list and partitioning the other elements into two sub-lists, according to whether they are less than or greater than the pivot. The sub-lists are then recursively sorted.

There are many different versions of Quicksort that pick pivots in different ways.

- Always pick the first element as a pivot.
- Always pick the last element as a pivot (implemented below).
- Pick a random element as a pivot.
- The Pick median as the pivot.

2. Code example.

```
#include <algorithm>
#include <iostream>
#include <vector>
void quickSort(std::vector<int> &arr, int left, int right) {
  int i = left, j = right;
  int tmp;
  int pivot = arr[(left + right) / 2];
  /* partition */
  while (i <= j) {
    while (arr[i] < pivot)
      i++;
    while (arr[j] > pivot)
      j--;
    if (i <= j) {
      tmp = arr[i];
      arr[i] = arr[j];
      arr[j] = tmp;
      i++;
      j--;
    }
  }
  /* recursion */
  if (left < j)
    quickSort(arr, left, j);
  if (i < right)
    quickSort(arr, i, right);
}
int main() {
  std::vector<int> arr = {5, 2, 9, 1, 3, 7, 4, 8, 6};
  quickSort(arr, 0, arr.size() - 1);
  for (auto &x : arr)
    std::cout << x << ' ';
  std::cout << '\n';
  return 0;}
```

This program will output the sorted array: 1 2 3 4 5 6 7 8 9.

3. *Code explanation.*

The **`quickSort`** function is a recursive implementation of the Quicksort algorithm. It takes in a vector of integers and the left and right indices of the portion of the vector that should be sorted.

The first thing the function does is select the pivot element. In this case, the pivot is chosen to be the middle element of the sub list being sorted. Then, the function enters a loop that will partition the sub list into two smaller sub lists, one containing elements that are less than the pivot, and the other containing elements that are greater than the pivot.

The partitioning is done by using two indices, **`i`** and **`j`**, that start at the left and right ends of the sub list, respectively. The loop continues until **`i`** is greater than j. At each iteration, the loop will find an element at index **`i`** that is greater than or equal to the pivot, and an element at index j that is less than or equal to the pivot. If **`i`** is less than or equal to **`j`**, the two elements are swapped, and **`i`** and **`j`** are incremented and decremented, respectively.

After the loop finishes, the sub list is partitioned into two smaller sub lists. If the left sub list is non-empty, the **`quickSort`** function is called recursively on it. If the right sub list is non-empty, the **`quickSort`** function is called recursively on it as well. This process continues until the sub lists are of size 0 or 1, at which point they are considered to be sorted.

Finally, the main function creates a vector of integers and calls the **`quickSort`** function on it, passing in the vector, and the left and right indices of the entire vector. After the function returns, the vector will be sorted in ascending order, and the sorted list is printed to the console.

4. *Time complexity.*

The time complexity of the Quicksort algorithm is **`O(n * log(n))`**. This means that, in the worst case, the algorithm will take approximately **`n * log(n)`** time to complete, where **`n`** is the number of elements in the list being sorted.

This time complexity is achieved because, on each level of the recursion, the list is partitioned into two smaller sub lists, and the size of the sub list being sorted is halved. As a result, the number of levels in the recursion is **`log(n),`** and the time complexity at each level is **`O(n).`** Therefore, the overall time complexity is **`O(n * log(n)).`**

It's important to note that the time complexity of Quicksort is sensitive to the choice of pivot element. In the worst case, if the pivot is always chosen to be the smallest or largest element in the list, the time complexity can degrade to **`O(n^2).`**

However, in the average case, the time complexity is **`O(n * log(n)).`**

I wish you happy learning, Your Rosen Rusev