Sorting Number Report

Prepared by Omar Abd Al-Majeed Ababneh

omarababneh1010@gmail.com

Introduction:

In this report I will explain Quick Sorting algorithm that useful in the sorting list

I will talk about Merge Sorting used in Collection.sort().

Quick Sorting:

Initially I want to explain how to work this algorithm.

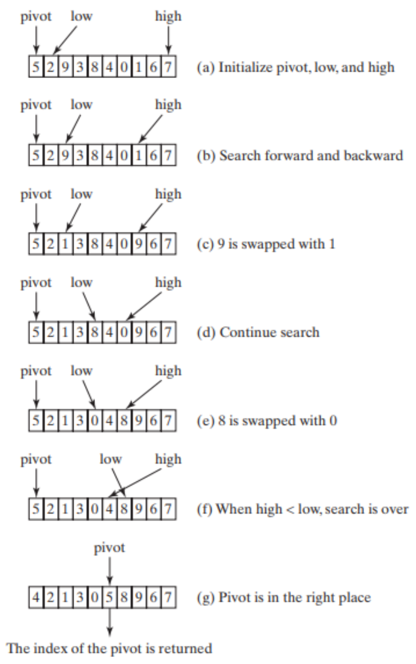
I will use array until explain this algorithm.

Description:

The Quick Sorting is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than pivot) before pivot, and put all greater elements (greater than pivot) after pivot.



This is explain to partitioning function and how to determine a pivot

Implementation:

This function consider Sort function it is receive list of object as argument then call another function used the same name but have three argument it is helper function.

```
private static void QuickSort(Object[] list) {
    QuickSort(list,0,list.length-1);
}
```

The helper function contains three argument.

1-list: contain data.

2- first: index beginning the list.

3 -last: index end the list.

This function will make recursive call until becomes first greater than last.

This function divides the problem to smaller problem in order become to deal with it easier.

```
private static void QuickSort(Object[] list, int first, int last) {
    if(last>first){
        int PivotIndex=Partition(list,first,last);
        QuickSort(list,first,PivotIndex-1);
        QuickSort(list,PivotIndex+1,last);
    }
```

partition function it is receiving list, first, last.

I will divide code to pieces until can explain.

In this code I assigned initially value to the variables and I pick first element as pivot.

```
int pivot = list[first]; // Choose the first element as the pivot
int low = first + 1; // Index for forward search
int high = last; // Index for backward search
```

In this code I try to check values in the list and put values in the correct location through make swapping and compare with a pivot.

```
while (high > low) {
  // Search forward from left
   while (low <= high && list[low] <= pivot)
      low++;
   // Search backward from right
   while (low <= high && list[high] > pivot)
      high--;
   // Swap two elements in the list
   if (high > low) {
      int temp = list[high];
      list[high] = list[low];
      list[low] = temp;
     }
   }
```

here I complete in the list until if there was list[high] <pivot it is becoming pivot.

```
while (high > first && list[high] >= pivot)
         high--;
```

Finally return index of pivot.

```
// Swap pivot with list[high]
if (pivot > list[high]) {
```

```
list[first] = list[high];
          list[high] = pivot;
          return high;
          }
       else {
          return first;
          }
       }
```

Why does Collection.sort() use merge sorting Algorithm?

Merge sort or more specifically TimeSort is used for sorting collections of objects Stability is required.

Stability is a big deal when Sorting arbitrary objects it’s a nice side benefit that merge sort guarantees nlog(n) (Time Complexity) performance no matter what the input object.

why did I use QuickSort algorithm to sort list?

Merge sort is more efficient than quick sort in the worst case, but the two are equally efficient in the average case. Merge sort requires a temporary array for sorting two subarrays. Quick sort does not need additional array space. Thus, quick sort is more space efficient than merge sort.

QuickSort is used for primitive arrays as it is slightly more efficient.

It’s very easy to avoid quicksort’s worst-case run time of O(n2) almost entirely by using an appropriate choice of the pivot – such as picking it at random (this is an excellent strategy).

In practice, QuickSort

Sample output:

```
Result: 13.990 ±(99.9%) 1.431 s/op [Average]
 Statistics: (min, avg, max) = (13.163, 13.990, 15.480), stdev = 0.946
 Confidence interval (99.9%): [12.559, 15.421]
```

For Merge Sort

Sample output:

```
Result: 20.094 ±(99.9%) 1.258 s/op [Average]
Statistics: (min, avg, max) = (19.171, 20.094, 21.716), stdev = 0.832
Confidence interval (99.9%): [18.836, 21.352]
```

Thank You