Koray Kural – 150170053 10/12/2020

Algorithms I – HWI Report

Code:

```
1   void quick_sort(int lower_bound, int higher_bound)                                  0
2   {                                                                                   0
3     if (lower_bound >= higher_bound)                                                  1
4       return;                                                                         1
5                                                                                       0
6     int pivot_index = lower_bound;                                                    1
7     int first_high_index = lower_bound + 1;                                           1
8                                                                                       0
9     while (compare(first_high_index, pivot_index) && first_high_index <= higher_bound)  n – x + 1
10      first_high_index++;                                                             n - x
11                                                                                      0
12    for (int i = first_high_index + 1; i <= higher_bound; i++)                        x + 1
13    {                                                                                 0
14      if (compare(i, pivot_index))                                                    x
15      {                                                                               0
16        swap(i, first_high_index);                                                    x
17        first_high_index++;                                                           x
18      }                                                                               0
19    }                                                                                 0
20                                                                                      0
21    int new_pivot_index = first_high_index - 1;                                       1
22    if (pivot_index != new_pivot_index)                                               1
23      swap(pivot_index, new_pivot_index);                                             1
24                                                                                      0
25    quick_sort(lower_bound, new_pivot_index - 1);                                     1
26    quick_sort(new_pivot_index + 1, higher_bound);                                    1
27  }                                                                                   0
```

A)

Asymptotic Upper Bound for the Quicksort

Worst case	$O(n^2)$
Best case	$O(n\log_2 n)$
Average case	$O(n\log_2 n)$

For the best case, pivot should be the median of the dataset. For randomized quicksort, average case is close enough to best case so that I will examine the recurrence equations for both of them only once.

Best and average case:

$$T(N) = 2 * T\left(\frac{N}{2}\right) + N$$

$$T(N) = 2 * \left[2 * T\left(\frac{N}{4}\right) + \frac{N}{2}\right] + N$$

$$\dots$$

$$T(N) = 2^k * T\left(\frac{N}{2^k}\right) + kN$$

$$T(1) = 0$$

$$2^k = N$$

$$k = \log_2 N$$

$$T(N) = N\log_2 N$$

Worst case:

$$T(N) = T(N-1) + N$$

$$T(N) = T(N-2) + N - 1 + N - 2$$

$$\dots$$

$$T(N) = T(1) + \sum_{x=0}^{N} N - x$$

$$T(N) = \frac{N(N-1)}{2}$$

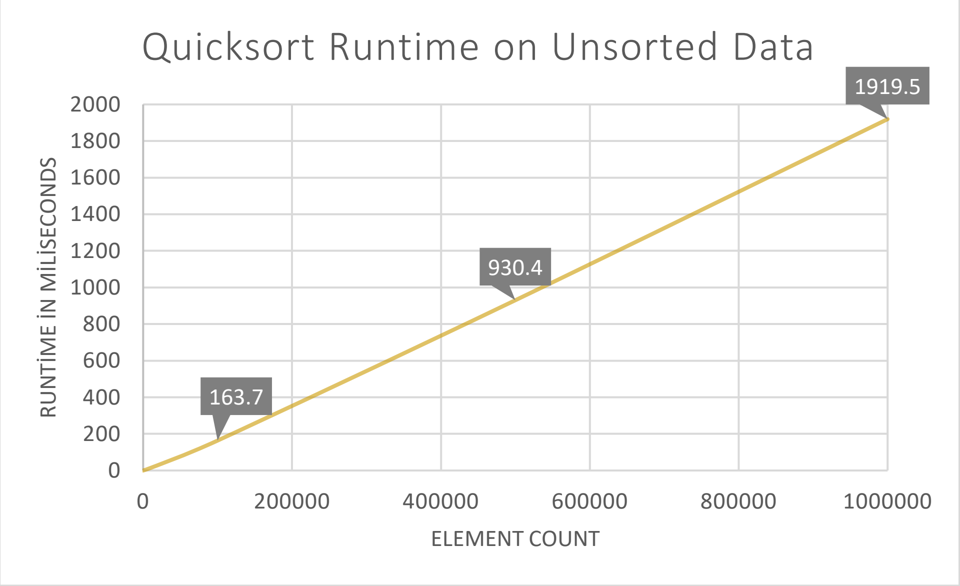
B)

No, this method does not sort sales in the desired way. Only stable algorithms can give the completely sorted results for these kinds of consecutive operations and quicksort is not a stable algorithm. Quicksort can be made stable but you need to make expensive shifting operations instead of simple swaps while partitioning. Some of the stable algorithms are insertion sort, merge sort and bubble sort.

C)

Runtimes in milliseconds on unsorted data

N	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Average
10	0	0	0	0	0	0	0	0	0	0	**0**
100	0	0	0	0	0	0	0	0	0	0	**0**
1000	1	2	1	1	1	2	1	1	2	1	**1.3**
10000	12	18	17	12	16	12	13	13	17	12	**14.2**
100000	191	151	152	152	153	192	149	150	152	195	**163.7**
500000	875	875	910	1047	922	911	937	963	913	951	**930.4**
1000000	1762	1931	2008	1954	1854	1911	1989	1807	2047	1932	**1919.5**

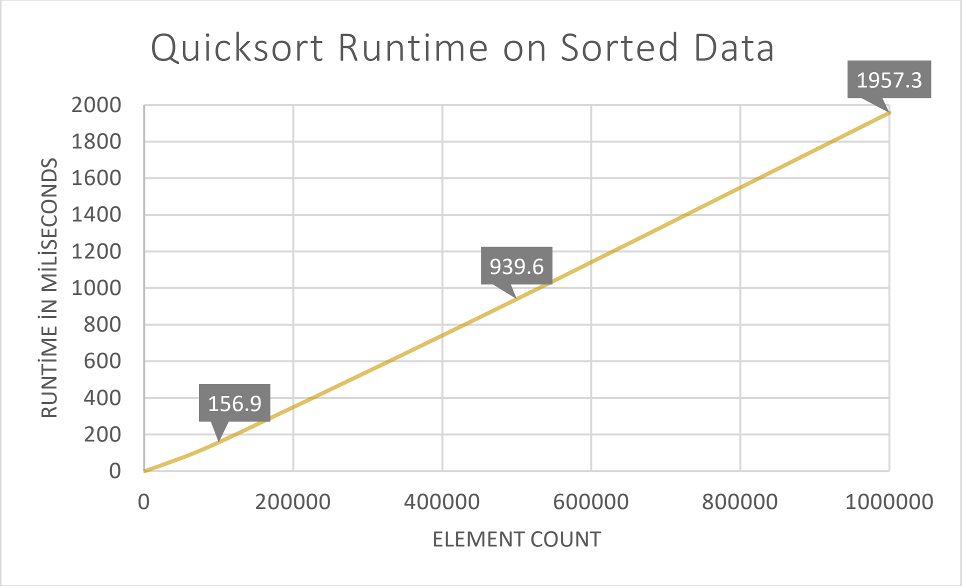


Since average case of quicksort is bounded with $O(n \log_2 n)$, a super-linear graph is expected. Resulting graph is also mostly linear and expected behavior is happened.

D)

Runtimes in milliseconds on sorted data

N	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10	Average
10	0	0	0	0	0	0	0	0	0	0	0
100	0	1	0	0	0	0	0	0	0	0	0.1
1000	1	0	1	1	1	0	1	0	1	0	0.6
10000	12	13	11	15	14	11	12	18	11	12	12.9
100000	152	179	150	151	151	181	150	151	152	152	156.9
500000	873	932	915	938	964	910	1023	927	932	982	939.6
1000000	1756	1919	2195	1814	1872	2240	1982	1958	1887	1950	1957.3



Since we are choosing first element as pivot and data is already sorted, we are choosing the smallest element as pivot every time. This is the worst-case scenario of quicksort. Considering worst case bound of quicksort is $O(n^2)$ runtimes should be much slower and maybe not even end in a reasonable time. However, for my case, runtimes are nearly identical to average case. This is a strange and unexpected result and I do not have any comment on this.