Author: Betsaleel Henry

HW4:

QUICKSORT and RANDOMIZED-QUICKSORT

Difference:

The two procedures differ only in how they select pivot elements; they are the same in all other respects. We can therefore couch our analysis of RANDOMIZED-QUICKSORT by discussing the QUICKSORT and PARTITION procedures, but with the assumption that pivot elements are selected randomly from the subarray passed to RANDOMIZED-PARTITION. The running time of QUICKSORT is dominated by the time spent in the PARTITION procedure. Each time the PARTITION procedure is called, it selects a pivot element, and this element is never included in any future recursive calls to QUICKSORT and PARTITION. Thus, there can be at most n calls to PARTITION over the entire execution of the quicksort algorithm. One call to PARTITION takes O.1/ time plus an amount of time that is proportional to the number of iterations of the for loop in lines 3–6. Each iteration of this for loop performs a comparison in line 4, comparing the pivot element to another element of the array A.

Therefore, if we can count the total number of times that line 4 is executed, we can bound the total time spent in the for loop during the entire execution of QUICKSORT.

The worst-case: split at every level of recursion in quicksort produces $\Theta(n^2)$. running time, which, intuitively, is the worst-case running time of the algorithm. Using the substitution method , we can show that the running time of quicksort is O(n^2). Let T (n) be the worst-case time for the procedure QUICKSORT on an input of size n. We have the recurrence

$$\begin{aligned} T(n) &\le \max_{0 \le q \le n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= c \cdot \max_{0 \le q \le n-1} (q^2 + (n-q-1)^2) + \Theta(n) . \end{aligned}$$

The expression $q^2 + (n-q-1)^2$ achieves a maximum over the parameter's range $0 \le q \le n-1$ at either endpoint. To verify this claim, note that the second derivative of the expression with respect to q is positive This observation gives us the bound max $0 \le q \le n-1$

Continuing with our bounding of T (n), we obtain:

$$\begin{aligned} T(n) &\le cn^2 - c(2n-1) + \Theta(n) \\ &\le cn^2 , \end{aligned}$$

Thus, the (worst-case) running time of quicksort is $\Theta(n^2)$.

Author: Betsaleel Henry

Running Time comparison:

The QUICKSORT and RANDOMIZED-QUICKSORT procedures differ only in how they select pivot elements; they are the same in all other respects. We can therefore couch our analysis of RANDOMIZED-QUICKSORT by discussing the QUICKSORT and PARTITION procedures, but with the assumption that pivot elements are selected randomly from the subarray passed to RANDOMIZED-PARTITION. The running time of QUICKSORT is dominated by the time spent in the PARTITION procedure. Each time the PARTITION procedure is called, it selects a pivot element, and this element is never included in any future recursive calls to QUICKSORT and PARTITION. Thus, there can be at most n calls to PARTITION over the entire execution of the quicksort algorithm. One call to PARTITION takes O.1/ time plus an amount of time that is proportional to the number of iterations of the for loop in lines 3–6. Each iteration of this for loop performs a comparison in line 4, comparing the pivot element to another element of the array A. Therefore, if we can count the total number of times that line 4 is executed, we can bound the total time spent in the for loop during the entire execution of QUICKSORT.
By Lemma 7.1 in the textbook we can say:
Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is O(n + X).

Program in c++ explanation

In our experiment code we implement a class to which we add up the different methods whose role would be to perform the RANDOMIZED-QUICKSORT and the QUICKSORT algorithm respectively.
The program include therefore also a driver program that operates the different steps respectively.

For 100 elements

```
Please input size:
100

number of execution of 4th line is : 370 times
 Sorted array With randomized quicksort time: 4.2e-05 secs.
 Sorted array in quicksort(Normal) time: 4.8e-05 secs.
```

```
Please input size:
100

number of execution of 4th line is : 449 times
 Sorted array With randomized quicksort time: 4.3e-05 secs.
 Sorted array in quicksort(Normal) time: 4.9e-05 secs.
```

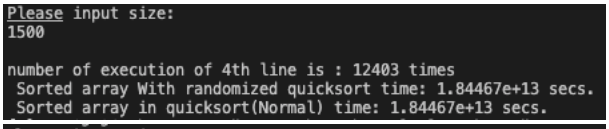
For 1000 elements

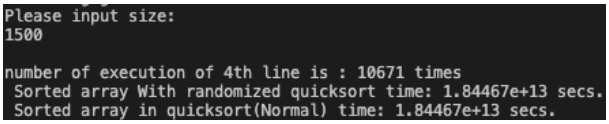
```
Please input size:
1000

number of execution of 4th line is : 26870 times
 Sorted array With randomized quicksort time: 0.000555 secs.
 Sorted array in quicksort(Normal) time: 0.001097 secs.
```

```
1 warning generated.
Please input size:
1000

number of execution of 4th line is : 25542 times
 Sorted array With randomized quicksort time: 0.00062 secs.
 Sorted array in quicksort(Normal) time: 0.001042 secs.
```

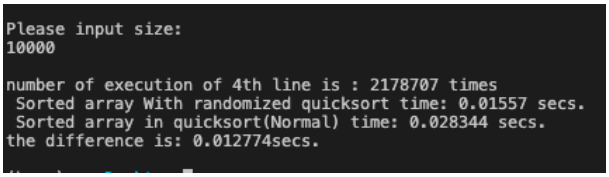
Author: Betsaleel Henry

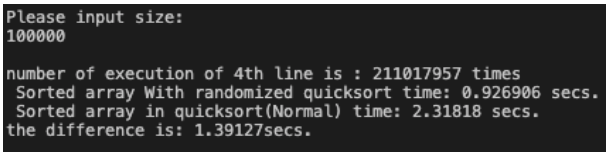
For 1500 elements



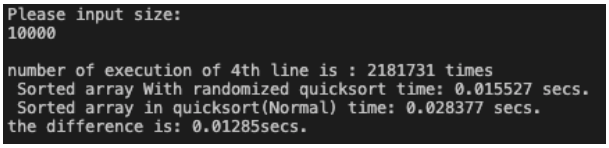


For 10000 elements





For 100000 elements~



The program is provided in the folder including this file for further simulations.

Summary

In this report we present the two quicksort algorithms and implement and record their running time for real application in c++.

From our result we can conclude that the average time taking by the randomized quicksort grows higher than the one taking by the simple fact that the randomized quicksort pseudocode add up to randomize part in it.