CS29002 Algorithms Laboratory

Assignment No: 7

Last date of submission: 07–September–2016

So far, you have seen running times of algorithms expressed in the big-Oh notation. In order to put these theoretical metrics to practical tests, we take two sorting algorithms in this assignment (quick sort and merge sort), and measure their actual running times on inputs of various sizes. Let n denote the size of an array A (of integers) which we want to sort.

The performance of quick sort is very sensitive to the initial distribution of A and also to the choice of the pivot. You will work with random, sorted, and almost sorted arrays. Also, the following ways of choosing the pivot for partitioning should be experimented with.

FIRST	Choose the first element $A[0]$ as the pivot.
RANDOM	Choose $A[r]$ as the pivot for a random $r \in \{0, 1, 2, \ldots, n-1\}$.
MEDIAN OF THREE	Choose $r, s, t \in \{0, 1, 2, \ldots, n-1\}$, and take the median of $A[r], A[s], A[t]$ as the pivot. Use both the following choices for r, s, t. (1) $r = 0$, $s = n/2$, and $t = n-1$. (2) $r = n/4$, $s = n/2$, and $t = 3n/4$.

- Write a function *quicksort*$(A, n, pivot\_type)$ to sort an array A with n (non-negative) integers. The third argument *pivot_type* indicates how you choose the pivot for partitioning: 0 means FIRST, 1 means RANDOM, 2 means MEDIAN OF THREE (1), and 3 means MEDIAN OF THREE (2).

The working of merge sort is not theoretically sensitive to the distribution of A or other parameters (merge sort has nothing like a pivot).

- Write a function *mergesort*(A, n) to sort an array A of n (non-negative) integers.

The *main*() function

(a) For $n = 10^k$, $k = 4, 5, 6, 7$, and for each choice of the pivot type, repeat the following:

Populate an array A with n random integers in the range 0 to $10^9 - 1$. The random choices of elements may lead to repetitions. You do not need to avoid repetitions. Sort A by calling *quicksort*. Now, A is sorted. Call *quicksort* again on this sorted array. Now, swap a few elements in A (take $k = n/100$, and swap k randomly chosen pairs of A). This gives you an almost sorted array. Run *quicksort* again on this array. Report the times taken by the three calls. Do not include the array-preparation times.

(b) For $n = 10^k$, $k = 4, 5, 6, 7$, repeat the following:

Populate A with n random integers in the range 0 to $10^9 - 1$. Sort A by calling *mergesort*. Call *mergesort* again on this sorted array. Take $k = n/100$, and swap k randomly chosen pairs of A. Run *mergesort* again on this almost sorted array. Report the times taken by the three calls.

A random integer in the range $[0, 10^9 - 1]$ can be generated by the call (`#include <stdlib.h>`):

```
A[i] = rand() % 1000000000;
```

Different runs of your program should handle different random sequences. In order to achieve that, write the following line at the beginning of your `main()` function. You need to include `<stdlib.h>` and `<time.h>`.

```
srand((unsigned int)time(NULL));
```

Finally, this is how you can measure (calendar) times in C/C++ programs (include `<time.h>`).

```
clock_t c1, c2;
double runtime;

c1 = clock();

--- Code for which the running time is to be measured ---

c2 = clock();
runtime = (double)(c2 - c1) / (double)CLOCKS_PER_SEC; /* Time in seconds */
```

Sample output

Present your output in the following format. You may compile your program with the `-O2` optimization flag.

```
+++ Performance of Quick Sort

        n    Pivot type                Random      Sorted    Almost sorted

    10000 FIRST                         0.001       0.002       0.002
    10000 RANDOM                        0.001       0.001       0.001
    10000 MEDIAN OF THREE 1             0.001       0.002       0.001
    10000 MEDIAN OF THREE 2             0.001       0.000       0.001

   100000 FIRST                             -           -           -
   100000 RANDOM                            -           -           -
   100000 MEDIAN OF THREE 1                 -           -           -
   100000 MEDIAN OF THREE 2                 -           -           -

  1000000 FIRST                             -           -           -
  1000000 RANDOM                            -           -           -
  1000000 MEDIAN OF THREE 1                 -           -           -
  1000000 MEDIAN OF THREE 2                 -           -           -

 10000000 FIRST                             -           -           -
 10000000 RANDOM                            -           -           -
 10000000 MEDIAN OF THREE 1                 -           -           -
 10000000 MEDIAN OF THREE 2                 -           -           -


+++ Performance of Merge Sort

        n                              Random      Sorted    Almost sorted

    10000                               0.001       0.001       0.001
   100000                                   -           -           -
  1000000                                   -           -           -
 10000000                                   -           -           -
```

Submit a single C/C++ source file. Do not use global/static variables.