COMPSCI 342: Data Structures and Algorithms

Programming Assignment 2 (60 points + 20 extra credit points)

Due Date: see class website

1 Description

Implement radixsort in Java.

1. Fix the number of elements being sorted at 5000000. Now vary the number of bits that you use for counting sort as follows.

   $$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16$$

   Note that the size of the counting array will be $k = 2^{bits}$. Observe the run-time for each value of *bits*. Make a table for the run-time versus d, where d is the number of passes radixsort makes. Remember that the number of passes made by radixsort is $d = \lceil \text{MACHINE\_WORD\_SIZE}/bits \rceil$. For most machines the constant MACHINE_WORD_SIZE is 32, but your program should not rely on this assumption. For example, in Java, we can get the word size from the constant `Integer.SIZE`.

2. Now run your best radixsort implementation for the following input sizes:

   $$500000, 1000000, 3000000, 10000000, 20000000, 30000000$$

   and make a table of input size versus run-time.

Notes:

- For the purposes of testing use random numbers generated using the `java.util.Random()` class. Don't restrict the range of random numbers being generated (so that all 32 bits contain useful information). The argument seed is a random seed, which can be any non-negative integer.

```
public void generate_random_array(int A[], int n, long seed)
{
    Random generator = new Random(seed);
    for (int i=0; i<n; i++)
        A[i] = generator.nextInt(Integer.MAX_VALUE);
}
```

- The program should take three arguments, one for the number of bits to use in the counting sort, the second for the size of the input and the last one for a random seed.

```
java RadixSort <bits> <input-size> <random seed>
```

- Make sure that you check the sorting by writing a simple check function with the following prototype.

```
// returns true if A[0..n-1] is sorted.
// return  false if A[0..n-1] is not sorted.
boolean CheckIfSorted(int A[])
```

- You should time only the radixsort function. For Java programs, see the example in `~amit/cs342/lab/tim` for timing the program accurately. If you are developing on MS Windows, you can do simpler (but less accurate) timing by using System.currentTimeMillis() method as follows.

```
long totalTime = System.currentTimeMillis();
radixsort(...);
totalTime = System.currentTimeMillis() - totalTime;
```

  However, you will need to use the method `report_cpu_time` from the `NativeTiming` class for the final version that you submit.

- Compare your run time with the run time for sorting with quick-sort (as used in `Arrays.sort` method).

- You may find it simpler to test your program on inputs with only 0s and 1s before testing on 32-bit random numbers generated by `java.util.Random()` class.

- The Java virtual machine has limited amount of memory. You will need to run it with larger memory for this program. Use the following command as an example of increasing the memory size for the Java Virtual Machine.

  ```
  java -Xms384m -Xmx384m RadixSort 12 20000000 123
  ```

- You are expected to develop this program on your own. Finding an implementation on the Internet or another book or from another person would be considered as cheating.

- **Bit Fiddling**. The following function extracts a k-bit field at position `start` in a integer. The operator `>>>` is the arithmetic shift right operator in Java (which fills in zeroes at the leading bits).

```
/*
 *
 * @param  x a positive number from which we extract a bit-field
 * @param  start  starting bit position (bits are numbered from 0)
 * @param  n the size of the bit-field.
 *
 * @return  the integer value in bits [start..start+n-1] of x.
 */

private int getbits(int x, int start, int n) {
    return ((x >>> start) & ((1<<n) - 1));
}
```

2 Extra Credit 1 (10 points)

Convert your radixsort program so that it sorts an array of long types. Note that you want to make sure that the generated random input array is still all positive numbers. The function Random.nextLong() gives you 64-bit random long integers but they are not always positive, so you would need to take the absolute value.

Note that if you do this part, make sure your code can do both 32-bit and 64-bit sorting. You may use method overloading to provide two different methods.

3 Extra Credit 2 (10 points)

If your time is better than the time for `Arrays.sort` on the same input, then you can get this extra credit.

4 Submitting the assignment

All of your work should be in one subdirectory. Let's say you have created a directory `cs342` inside your home directory. Then create a directory `p2` in the directory `cs342` and `cd` to this directory. All your programs, script files should be in this directory. The timing results and any other comments should be in a file called `README`. If this file isn't a text file, then make sure it has an appropriate suffix (like `.odt` or `.doc` or `.pdf`) You should have a Makefile that compiles and generates your program. The name of the class must be `RadixSort`.

The program should take three arguments, one for the number of bits to use in the counting sort, the seconds for the size of the input and the last one for a random seed.

```
java RadixSort <bits> <input-size> <random seed>
```

The program should print out the sorting times for radixsort and `Arrays.sort` in seconds (with 2 digits decimal precision)a. Here is an example of typical output.

```
java RadixSort 11 5000000 11123
Radixsort time = 1.61
Arrays.sort time = 2.15
```

Once you are ready and in the directory ∼`/cs342/p2`, you can submit your assignment as follows:

```
submit amit cs342 2
```

The command will copy the contents of your current directory (recursively) and make a copy somewhere in the instructor's home directory that only the instructor can access. If there is a problem, then you will get an error message.