Fall 2015 Biostat 615 Homework #2 (Total 40 pts)

Due by Wednesday October 14th, 2015 noon as a compressed file "hw2.tar.gz" containing the required C++ source code files for the following four problems. Please read the instruction carefully before you start to work on your homework.

Instruction

- The additional grading will be executed on every day from October 6th to October 13th. The final grading will be executed exactly at 12:10pm, Wednesday October 14th, 2015.
- **(Note: changed from homework #1)** Your program might get input from `cin` or `argv` instead of only `argv`, according to the specific description given each problem.
- Make your algorithm as efficient as you can, as some of the test cases may be computationally challenging and your program will be terminated if it does not finish after running for 10 seconds and you will lose the points for those test cases. However, you should always turn in a program even if it is not very efficient so that you can at least get partial points.
- All your programs (e.g., if it is named as `program.cpp`) will be compiled and tested on the server `scs.itd.umich.edu` using the following command:

```
g++ -O -o program program.cpp
```

  where "program" will be replaced by the corresponding program name for each problem.
- For this homework, you are NOT allowed to use your own header files in your submission.

Problem 1 - Compute Simple Quantiles (10 pts)

Suppose that values $x_1, \ldots, x_n$ have been realized from a random variable X. Simple estimators of the quantiles for X can be obtained from the order statistics: i.e. the data arranged in numerically ascending order. If $x_{(1)} \leq \ldots \leq x_{(n)}$ denote the order statistics, the uth quantile ($0 \leq u \leq 1$) is defined to be

$$\widetilde{Q}(u) = \begin{cases} x_{(nu)}, & \text{if } nu \text{ is a positive integer} \\ x_{(\lfloor nu \rfloor + 1)}, & \text{otherwise} \end{cases}$$

Write a C++ program `simpleQuantiles.cpp` to compute a number of uth quantiles of a sequence of input numbers. Which quantiles to be computed are specified using the program augment list `argv`. Your program should obtain the input numbers using `cin` from one line in the console. Your program should print the quantile values separated by white spaces in a new line. No error handling for malformed argument is needed. Example runs of valid input arguments are

```
user@host:~/Private/biostat615/hw2$./simpleQuantiles 0.5
1 2 3 4 5
3
user@host:~/Private/biostat615/hw2$./simpleQuantiles 0.25 0.5 0.75
-1.86  0.38  0.16  0.16 -0.48  1.07  0.55  0.61  0.76
0.16 0.38 0.61
```

Problem 2 - Minimum Sub-Sum (10 pts)

Write a C++ program `minSubSum.cpp` which, given a sequence of integers (from `cin`), computes and prints the minimum sum of a contiguous subsequence. Empty subsequence (with a sum of 0) is also allowed. Example runs of valid input arguments are

```
user@host:~/Private/biostat615/hw2$ ./minSubSum
-1 1 -1 1
-1
user@host:~/Private/biostat615/hw2$ ./minSubSum
-1 -2 -1 2 -1
-4
```

Problem 3. One-sided Fisher's Exact Test (10 pts)

Write a C++ program `oneSidedFastFishersExactTest.cpp`, as a modified version of the fast fishers exact test presented in the class. Instead of calculating the p-value $p_{two-sided}(a,b,c,d) = \sum_x \Pr(x) I[\Pr(x) \leq \Pr(a)]$ as in the two-sided tests given in the class, your problem should perform a one-sided test by calculating the following p-value

$$p_{one-sided}(a,b,c,d) = \sum_{x<a} \Pr(x)$$

and output "significant" if the calculated p-value is smaller than 0.05, and "not significant" otherwise. Example runs of valid input arguments are

```
user@host:~/Private/biostat615/hw2$ ./oneSidedFastFishersExactTest 2 7 8 2
significant
user@host:~/Private/biostat615/hw2$ ./oneSidedFastFishersExactTest 7 2 2 8
not significant
```

Problem 4 - Sort Intervals (10 pts)

Write a C++ program `sortIntervals` to sort a set of integer intervals in a non-decreasing order by their lower bounds. The intervals with the same lower bounds are sorted by their upper bounds. The sorting algorithm should be is stable. Note that your C++ source code file must be named as `sortIntervals.cpp`. The input values will be $2n$ integers, denoted, $l_1, u_1, l_2, u_2, \ldots, l_n, u_n$. They represent the input intervals

$$[l_1, u_1], [l_2, u_2], \ldots, [l_n, u_n]$$

The output of the program has two lines: the first line prints the sorted intervals, that is

$$l_{\tau_1}\ u_{\tau_1}\ l_{\tau_2}\ u_{\tau_2}\ \ldots\ l_{\tau_n}\ u_{\tau_n}$$

with $l_{\tau_1} \leq l_{\tau_2} \leq \ldots \leq l_{\tau_n}$ and $u_{\tau_i} \leq u_{\tau_j}$ when $l_{\tau_i} = l_{\tau_j}$ for any $1 \leq i < j \leq p$. The second line prints the corresponding input order, i.e.

$$\tau_1\ \tau_2\ \ldots\ \tau_n$$

No error handling for malformed argument is needed. Example runs of valid input arguments are

```
user@host:~/Private/biostat615/hw2$./sortIntervals
5 6 1 5 1 2 1 2 3 5
1 2 1 2 1 5 3 5 5 6
3 4 2 5 1
user@host:~/Private/biostat615/hw2$./sortIntervals
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
1 2 3 4
```