CS325: Analysis of Algorithms, Fall 2017

Group Assignment 1[*]

Due: Tue, 10/10/17

Homework Policy:

1. Students should work on group assignments in groups of preferably three people. Each group submits to TEACH a zip file that includes their source code and their *typeset* report. Each group, also, hands in a printed hard copy of the report in class or slides the hard copy under my door before the midnight of the due day. The hard copy will be graded, and the codes submitted to teach will be tested.
2. The goal of the homework assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
3. You are allowed to discuss the problems with other groups, and you are allowed to use other resources, but you *must* cite them. Also, you must write everything in your own words, copying verbatim is plagiarism.
4. *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a completely wrong answer will receive zero.
5. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.
6. More items might be added to this list. ☺

Finding the kth smallest element of a *sorted* array $A[1 \ldots N]$ is easy. In fact, there is an $O(1)$ time algorithm: return $A[k]$. If A is not sorted, however, we need to spend $O(N)$ time to examine all elements of A. But, what if A is "somewhat" sorted? For example, can we do better than $O(N)$ if A is composed of large sorted subarrays?

Specifically, let $A_1[1 \ldots n], A_2[1 \ldots n], \ldots, A_m[1 \ldots n]$ be m sorted arrays, each of length n. Also, let $1 \le k \le mn$. We would like to find the kth smallest element in the union of all these arrays, $A_1 \cup A_2 \cup \ldots \cup A_m$. In particular, if $m = 1$, we have only one sorted array, so we can use the $O(1)$ time algorithm mentioned above. Also, if $n = 1$, we have an unsorted array, so we need to examine all its elements. But, what should we do in intermediate cases?

In this assignment, you describe and analyze an algorithm to find the kth smallest number of all numbers in m sorted arrays, each of length n. For example, given $m = 2$, $n = 3$, $k = 5$, $A_1 = [1, 6, 7]$ and $A_2 = [2, 2, 2]$ your algorithm must return 6. For full credit, the running time of your algorithm should be $O(m^{c_1} \log^{c_2} n)$ for a small constants c_1 and c_2. Note the polynomial dependency on m and the logarithmic dependency on n. What is the smallest c_1 and c_2 that you can get?

[*]The problem is from Jeff Erickson's lecture notes. Looking into similar problems from his lecture notes on recursion is recommended.

Report (60%). In your report, include the description of your algorithm, and provide running time analysis. Algorithms should be explained in plain english. You can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.

Code (40%). Write a program to find the kth smallest element of the array. Your program will be tested against several test cases, for correctness and efficiency. For each test case, the program will be automatically stopped after 20 seconds if it is not done in that time. In this case, the group will miss the points of that test case. Your program must be written in one of the following languages: Python, C++, or Java.

Input/Output Your program reads from multiple txt and binary input files. The input file "input.txt" contains three numbers $1 \le m \le 10$, $1 \le n \le 10^8$, and $1 \le k \le mn$ in this order, separated by commas. There are m binary files, "1.dat", "2.dat", ..., "m.dat", each composed of exactly $4n$ bytes with each four consecutive bytes representing an integer. Integers are stored in big endian format: the most significant byte is at the lowest address.

Your output must be written in "output.txt". The output is a single number: the value of the kth smallest number in all the binary files "1.dat", "2.dat", ..., "m.dat".

Sample Input (1):
"Input.txt": 2,2,3
"1.dat": 0000 0001 0000 0007 # 1, 7
"2.dat": 0000 0002 0000 0006 # 2, 6

Sample Output (1):
"output.txt": 6

Sample Input (2):
"input.txt": 3,5,14
"1.dat": 0000 0001 0000 0004 0000 0007 0000 000a 0000 000d # 1, 4, 7, 10, 13
"2.dat": 0000 0002 0000 0005 0000 0008 0000 000b 0000 000e # 2, 5, 8, 11, 14
"3.dat": 0000 0003 0000 0006 0000 0009 0000 000c 0000 000f # 3, 6, 9, 12, 15

Sample Output (2):
"output.txt": 14

In the examples above, the binary files are shown in hex format. The list of numbers after "#" are not in input files, they are given to improve readability of the examples. As mentioned each binary file is composed of exactly $4n$ bytes.

Submission Each group submits to TEACH a zip file that includes their source code (*which must be just one file with name "select.cpp", "select.java", or "select.py"*) and their report in pdf format. This file can be submitted by any member of the group, but all names must be listed in the submitted report. Each group, also, hands in a printed hard copy of the report in class or slides the hard copy under the door of my office before the midnight of the due day. The hard copy will be graded, and the submitted code to teach will be tested.

Your codes will be tested *automatically.* So, you need to carefully follow all formatting requirements mentioned above. To summarize:

(1) Your source code file should be named "select.cpp", "select.java", or "select.py".

(2) It reads from files "input.txt", "1.dat", "2.dat", ..., "m.dat" in the current folder. The format of the input files will be exactly as specified above.

(3) It writes to the file "output.txt" in the current folder. It should write exactly one number into "output.txt", with no extra symbol.

Test your code with the sample test files (`http://web.engr.oregonstate.edu/~nayyeria/CS325/Fall17/hws/test1.zip`) before submitting them, to make sure there is no formatting error.

Comments/hints:

(1) Note that the binary files may be very big. Hence, it is not possible to read the entire file into an array, because of the time constraint. To overcome this issue, you need to use random access binary files. You can open a file in binary mode, and use the command "seek" to access different addresses in the file.

(2) For designing the algorithm, first, consider the case that $m = 2$.