CSCI 201 – Computer Science 1

Homework 8. Due date: Sunday March 24

Objectives: *Employ the property of sortedness to make array processing more efficient. Design a nested* `for` *loop for the Matching-sum problem when the data is sorted. Measure the performance improvement obtained by using sortedness.*

IMPORTANT. *Use the* `store` *and* `binary search` *functions from Lab 6 and Homework 6 to implement the algorithm.*

Keeping data sorted often results in query processing being more efficient. We saw this in the case of searching, where the binary search algorithm is exponentially faster than sequential search. In this homework, we look at another strategy for the matching sum problem that *exploits the non-decreasing order of items in A*. We shall then compare the two strategies by plotting graphs that show how many array accesses were needed by each of the strategies to find the answer.

Question 1. (Measuring scalability of the nested for loop.) Create 5 input files containing sorted sequences of numbers (integers). The files should contain 20, 40, 60, 80 and 100 numbers respectively. For each file, identify (manually) a number, x, for which a matching sum cannot be found (note that this gives us the maximum number of array accesses, i.e., the worst-case scenario). Run your program from the lab, on all the five test files, using the number, x, you have identified, and record the number of array accesses in a table.

New Strategy. In the sample run given above, we had the numbers `2 3 3 4 7 9 13 16 16` and were looking for 23. We started with the first number, 2, and then checked all the other numbers to see if the sum was 23. Instead, if we observe that 23 - 2 = 21, all we have to do is look for 21, which can be done very quickly with binary search. More formally, if *A[i]* and *A[j]* form a matching sum for x, then $A[j]$ equals $(x - A[i])$. In the program written in the lab, the *(i + 1)*-th iteration of the outer loop examines the pairs
`(i, i+1), (i, i+2), ..., (i, n-1)`
to see if any of them equals x. Note that this is the same as checking if any item in $A[(i+1)\ldots(n-1)]$ is equal to $(x - A[i])$. In other words what we are doing is searching the slice $A[(i+1)\ldots(n-1)]$ for $(x - A[i])$. Since the array A is sorted, instead of searching all these items sequentially as we did in the lab, we can search for the value $(x - A[i])$ in the slice *A[(i + 1) . . . (n - 1)]* using a **binary search.**

Question 2. Modify the C++ program written in the lab to implement the above strategy for finding a matching sum. This modification requires two things:
(i) adding the `binSearch` function (re-use from the earlier homework).
(ii) replacing the inner loop in the main program by a call to `binSearch`.
The function call will search for for $x - A[i]$ in the slice $A[(i+1)...(n-1)]$. Your program should also keep track of the number of array accesses using a global variable. Compile and test in a script session.

Question 3. Run the program that uses the binary search strategy on the data files that you created for Question 1. Record the number of array accesses in a table.

Question 4. Plot the number of array accesses versus the size of the input file, using the same X and Y axes, for both strategies. Using your intuition and prior knowledge, try to find functions that best match these graphs.

What to turn in. Upload the script file and answers for Questions 1, 3 and 4 to CourseFiles.