**CoGS assignment**

**Lili Feng**

***Problem 1: Efficient implementation***

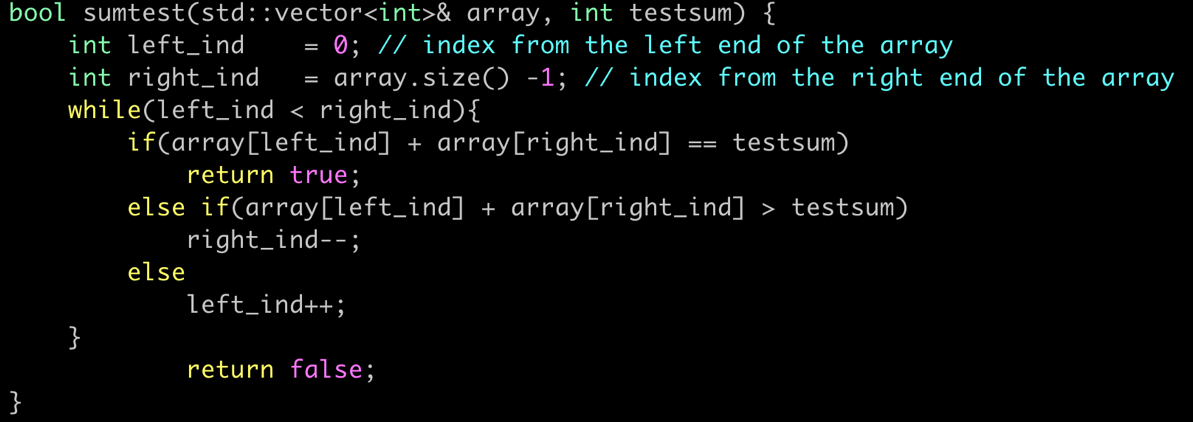
1. Write a function “sumtest(array of ints, int)” that tests whether any combination of two elements in the array sums up to a testsum and returns a boolean if this is so. Assume that the array of integers is sorted. For example,

sumtest([1,5,7,9], 6) should return true because 1 + 5 = 6

sumtest([1,5,7,9], 7) should return false.

Because this problem is about efficient implementation, so I choose to use C++ rather than Python to solve the problem.

The function is in the cpp file with path “./problem\_1/sumtest.cpp” and below is a screenshot of the function.



***Figure 1. Screenshot of the sumtest function***

**Instructions to compile and run the code**

The code has been successfully compiled with gcc 4.8.5 (the default gcc version comes with conda install -c anaconda gcc) on Linux and MacOS machines.

To compile the code:

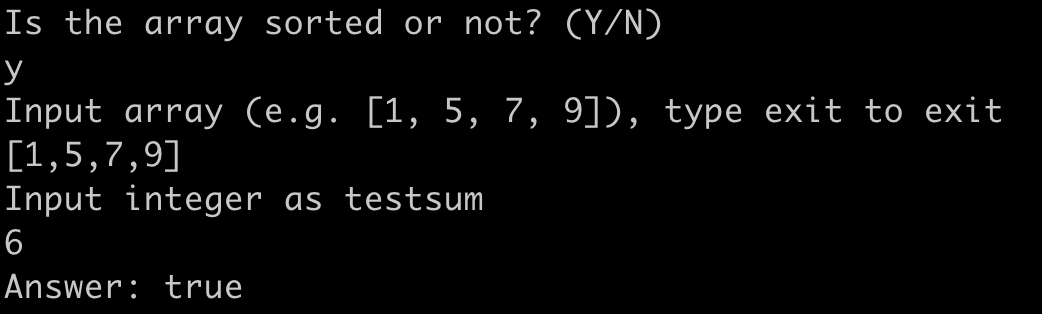
$ cd ./problem\_1

$ ./compile.sh

To run the code:

$ ./sumtest

And below is a screenshot of an example of sorted array



1. What is the fastest implementation that you can construct?

***Figure 1. The “shrink” algorithm***

Fig. 1 illustrates the algorithm that I implement in the function. There are two indices start with pointing to the left and right ends of the input array. The left index will move to the right if the sum of the two elements from the array is too small, while the right index will move to the left if the sum of the two elements from the array is too large.

1. What is its runtime as function of the length of the array N?

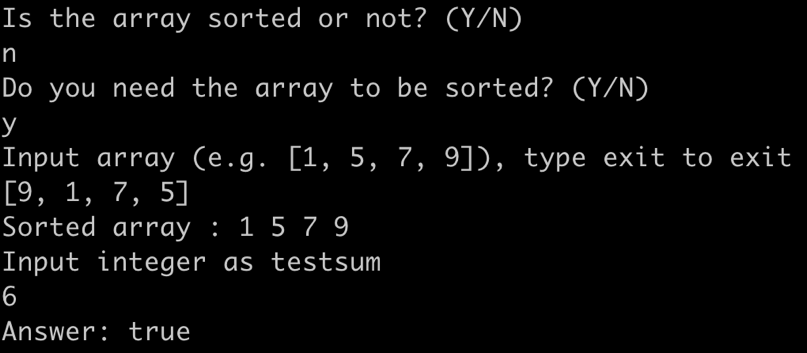
My implementation has a time complexity of O(N), with the worst case that the whole array has to be scanned.

1. How does the runtime change if the array needs to be sorted first?

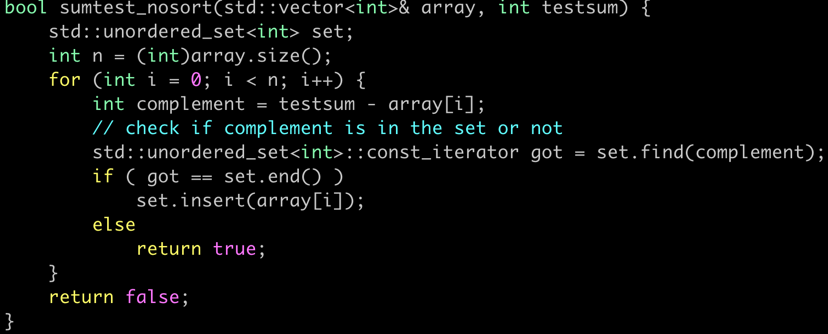
If the array is not sorted, there are two ways to solve the problem.

* 1. Sort the array at first, and then use the same sumtest function mentioned above.

In c++, there is a sort function can be applied to sort the input array. The time complexity is O(Nlog(N)) for sorting. And running the time complexity for running the sumtest is O(N). As a sum, the time complexity will be O(Nlog(N)) + O(N) ~ O(Nlog(N)). Below is a screenshot of running the code with an un-sorted array.



* 1. In fact, **there is no need to sort the array**. Below is the sumtest\_nosort function for un-sorted array



The function uses unordered\_set. Because the time complexity for operation of unordered\_set is O(1), so the time complexity for the sumtest\_nosort is O(N). Below is a screenshot of running the code with an un-sorted array without sorting it.

