**Please follow the instructions provided below to submit your assignment.**

**Worth:** 10% **Due:** By 11:59 pm on Friday Oct 20

- You must submit your assignment as a single PDF file through the MarkUs system.

  https://markus.teach.cs.toronto.edu/csc263-2017-09/

  The filename must be the assignment's name followed by "sol", e.g. your submission for the assignment "A2" must be "A2sol.pdf". Any group of two students would submit a single solution through Markus. Your PDF file must contain both team member's full names.
- Make sure you read and understand "POLICY REGARDING PLAGIARISM AND ACADEMIC OFFENSE" provided in the course information sheet.
- The PDF file that you submit must be clearly legible. To this end, we encourage you to learn and use the LaTex typesetting system, which is designed to produce high-quality documents that contain mathematical notation. You can find a latex template in Piazza under resources. You can use other typesetting systems if you prefer. Handwritten documents are acceptable but not recommended. The submitted documents with low quality that are not clearly legible, will not be marked.
- You may not include extra descriptions in your provided solution. The maximum space limit for each question is 2 pages. (using a reasonable font size and page margin)
- For any question, you may use data structures and algorithms previously described in class, or in prerequisites of this course, without describing them. You may also use any result that we covered in class, or is in the assigned sections of the official course textbook, by referring to it.
- Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision, and conciseness of your presentation.
- For describing algorithms, you may not use a specific programming language. Describe your algorithms clearly and precisely in plain English or write pseudo-code.

1. A binary search tree is *weight balanced* if for each node in the tree, the number of nodes in its two subtrees differ by at most 1. Recall that for an AVL tree, it is the height of the two subtrees that must differ by at most one.

   (a) (10 MARK) Give an algorithm to build a weight balanced BST given a sorted array of n elements. Your algorithm should run in $O(n)$ time. Analyze the time complexity of your algorithm.

   (b) (10 MARK) Prove that the height of a weight balanced BST is $O(logn)$.

   (c) (5 MARK) Does the same rebalance operation for AVL tree insert work for inserting to weight balanced BST? Justify your answer by a proof or a counter example.

2. Bob has $m$ dollars and wants to buy two items with an exact total cost of $m$. Assume that you are given a binary search tree that stores the price of $n$ available items and no two items have the same price.

   (a) (15 MARK) Write an $O(n)$ algorithm to find two items with total price of $m$.

   (b) (5 MARK) Analyze the time complexity of your algorithm.

3. Consider the following opertation in a binary tree that stores integer keys:

   - CLOSEST-PAIR($r$): returns the minimum difference of the keys among all pairs in the subtree rooted at $r$. In other words, Closest-pair($r$) returns the minimum value of $|x.key - y.key|$ among all elements $x, y$ rooted at $r$.

   We want to is augment AVL tree to support this operation in $O(1)$ time such that the time complexity of *Insert* and *Delete* remains $O(\log n)$.

   (a) (6 MARK) What extra information do we need to store at each node?

   (b) (8 MARK) Explain how to perform CLOSEST-PAIR in $O(1)$?

   (c) (16 MARK) Explain why Insert and Delete can still be performed in $O(\log n)$ time?

4. A *cryptographic hash function* is a particular class of hash functions that have special properties which make them suitable for use in cryptography. The ideal cryptographic hash function has five main properties:

   1) It is deterministic so the same message always results in the same hash.

   2) It is quick to compute the hash value for any given message.

   3) It is infeasible to generate a message from its hash value except by trying all possible messages.

   4) A small change to a message should change the hash value so extensively that the new hash value appears. uncorrelated with the old hash value

   5) It is infeasible to find two different messages with the same hash value

   In this question, you will see the algorithms for two simple string hash functions. For both algorithms, the input is a string of ASCII characters and the output is a hash code.

- **Algorithm 1**

  This algorithm is not the best possible algorithm, but it has the merit of extreme simplicity.

```
function HASH($S$)
    $hash = 0$
    for each characters $c$ in $S$ do
        $hash += \text{ASCII}(c)$
    end for
    return hash
end function
```

- **Algorithm 2**

  This algorithm is one of the best algorithms known for string hash.

```
function HASH($S$)
    $hash = 5381$
    for each characters $c$ in $S$ do
        $hash = hash * 33 + \text{ASCII}(c)$
    end for
    return hash
end function
```

(a) (7 MARK) What is the hash code of the string "abcd" using Algorithm 1 and Algorithm 2?

(b) (5 MARK) For each 5 properties of cryptographic hash functions mentioned above, discuss if Algorithm 1 has the property. Briefly justify your answer for each or give a counter example.

(c) (5 MARK) Why do you think Algorithm 2 is a better algorithm? (Specify at least one property in which you think algorithm 2 performs better than Algorithm 1. Justify your answer.)

(d) (8 MARK) Name two well-known cryptographic hash functions and specify the hash code of the string "abcd" using each of them. (You do not need to describe the algorithms and the computation process. You might need a quick Google search or running a command line program.)