Assignment 2: CSCI 4310/6323

Instructor: Dr. Bin Fu. Due March 11, 2021 (Thursday).
Please type your solution in MS word format.

Problem 1. (25%) You are given a string of n characters $s[1 \cdots n]$, which you believe to be a corrupted text document in which all punctation has vanished (so that it looks something like "itwasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function dict[.]: for any given string w, $dict(w) = true$ if w is a valid word, and $dict(w) = false$ otherwise.

(a) Given a dynamic programming algorithm that determines whether the string $s[.]$ can be reconstructed as a sequence of valid words. The running time should be $O(n^2)$, assuming calls a dict take a unit time.

(b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

Problem 2. (25%) This problem is about binomial heap:

(a)Suppose a binomial heap H has a total of n nodes. Prove that H has at most $\lfloor \log n \rfloor + 1$ binomial trees.

(b) Prove that the union operation of two binomial heaps takes $O(\log n)$ steps.

(c) The binomial heap H_1, which starts from empty, is generated by inserting the odd numbers from 11 to 40 sequentially, and the heap H_2, which starts from empty also, is generated by inserting the even numbers from 11 to 40 sequentially. (1)Show the structure of H_1 and H_2. (2) Show the structure of the union of H_1 and H_2.

Problem 3. (50%) The knapsack problem is that given a set of positive integers $\{a_1, \cdots, a_n\}$, and a knapsack of size s, find a subset A of $\{a_1, \cdots, a_n\}$ such that the sum of elements in A is the largest, but at most s.

Part 1. Use the dynamic programming method to design an algorithm for the knapsack problem. Prove the correctness of your algorithm. Show the computational time of your algorithm carefully.

Part 2. Use C++ to implement the function below int
knapsack(
int *a, //the input integers
int n, //the number of input integers
int s, //knapsack size
int *subset, //subset elements
int &size_of_subset //the number of items in the subset
)

Test your program for the following knapsack problem:

Input list: 5, 23, 27, 37, 48, 51, 63, 67, 71, 75, 79, 83, 89, 91, 101, 112, 121, 132, 137, 141, 143, 147, 153, 159, 171, 181, 190, 191 with knapsack size 762. Print out a subset with the sum of its elements so that the sum has the closest distance to 762. Also print out your source code.