Assignment #3

COP-3530, Summer C 2019

Release: June 25, 2019

Due date: July 3, 2019

Purpose:

This assignment has **4 problems**. The problems 2, 3, and 4 are relate to some of the important topics we studied in the Module 5 and the first part of Module 6.

The purpose of this assignment is:

- To apply the basic methods for the **collision resolution** in **hash tables**.
- Value the use of the different **quadratic sorting algorithms**.
- To apply basic techniques of algorithm analysis to implement efficient algorithms.

Submitting Your Assignment:

- **Assignments must be turned in via Canvas.**
- Please follow these steps for every assignment:

1. You are allowed to upload only a single file. Since programs will consist of multiple files, your upload must be a compressed "ZIP" file (with extension **.ZIP**).

   ☞ ***No other kinds of compressed files will be accepted!***

2. Please name your submission as **3_XXXXXX.ZIP**, where **XXXXXXX** is your seven-digit Panther ID number.
3. **Before uploading the zip file**, extract the individual files and examine them to make sure you have included all that is required. In your .ZIP you must include **ONLY** **Java files (.java)** and **PDF files** (generated using text editors like MS Word, WordPerfect, TextMaker, LaTex, etc). **Photos and scanned images, as well as solutions in a different format that the ones stated above will be not accepted**.
4. Please include the following header for each Java program:

```
/*****************************************************************
  Purpose/Description: <a brief description of the program>
  Author's Panther ID: <your Panther ID number>
  Certification:
    I hereby certify that this work is my own and none of it is the work of
    any other person.
*****************************************************************/
```

5. Please make sure that you do not include any other personal information in your submission (besides the **Panther ID** in the name of the **ZIP** file and in the headers of your Java files as explained above). For example, no date of birth or name should be found in the document(s) you submit.
6. **Submissions turned in after the due date and/or which don't meet the established formatting rules will not be accepted.**

Failure to follow these simple directions may result in a loss of credit for the assignment.

Problem 1: **(20 pts)**

Given an **unsorted** array of n integers numbers (with possible duplicates) in range [1..k].

Implement in **Java** an algorithm that preprocesses the input array in O(n+k) running time and then returns how many integer numbers there are in the range [left..right] in O(1) running time for any given left and right, 1 ≤ left ≤ right ≤ k.

Example:

Input:

```
int [ ] a = {3, 8, 2, 4, 1, 9, 11, 4, 15}; //input array
int k = 15;                                // max element in a
int left = 2;                              // lower value in range
int right = 9;                             // upper value in range
```

Output:

```
Total number in range [2..9] is 6
```

Important Notes:

- You must add the main method in your program in Java in order to test your implementation.
- There are no data errors that need to be checked as all the data will be assumed correct.
- You can use the array of the previous example to test your program, however, I suggest that you also use other input arrays to validate the correctness and efficiency of your solution.
- Your program MUST be submitted only in source code form (.java file).
- A program that does not compile or does not run loses all correctness points.

Problem 2: **(30 pts)**

2.1. Consider the problem of inserting the keys 10, 22, 31, 4, 15, 28, 17, 88, and 59 into a **hash table** of length 11 (**TableSize = 11**) using **open addressing** with the standard hash function **h(k) = k mod TableSize**. Illustrate the result of inserting these keys using:

(a) **Linear probing**.

(b) **Quadratic probing** with quadratic probe function **$c(i) = 3i^2 + i$**.

(c) **Double hashing** with **u(k) = k** and **v(k) = 1 + (k mod(TableSize − 1))**.

2.2. (Consider two sets of integers, S = {s_1, s_2, ..., s_m} and T = {t_1, t_2, ..., t_n}, m ≤ n. Propose a O(n+m) running time complexity algorithm (only pseudo-code) that uses a hash table of size m to test whether S is a subset of T.

Important Notes:

- For this problem, you don't need to submit any implementation in **Java**.
- You must justify your results and draw the hash tables using the appropriate graphics tools at your convenience. Hash tables that were drawn by hand will be not accepted.

<u>Problem 3:</u> (20 pts)

(a) Given a sorted array of N distinct integers that has been rotated an unknown number of times. Implement (in Java) an efficient algorithm that finds an element in the array.

(b) What is the running time complexity of your algorithm?

Note: You may assume that the array was originally sorted in increasing order.
Example:

> **Input:** find 5 in array (15 16 19 20 25 1 3 4 5 7 10 14)
>
> **Output:** 8 (the index of 5 in the array)

Important Notes:

- You must add the main method in your program in order to test your implementation.
- There are no data errors that need to be checked as all the data will be assumed correct.
- You can use the array of the previous example to test your program, however, I suggest that you also use other input arrays to validate the correctness and efficiency of your solution.
- Your program MUST be submitted only in source code form (.java file).
- A program that does not compile or does not run loses all correctness points.

<u>Problem 4:</u> (30 pts)

(a) Given an unsorted array A[1...n] of distinct integers numbers and is given a non-negative integer number, k, k < n. You need to find an element from A such that its rank is k, i.e., there are exactly (k-1) numbers less than or equal to that element.

Example:

> **Input:** A = [1, -3, 4, 3, 12, 20, 30, 7, 14, -1, 0] and k =8.
>
> **Output:** 12, since 7, -3, -1, 0, 4, 3, 1 (8-1=7 numbers) are all less than or equal to 12

Suggest a sub-quadratic running time complexity algorithm (only pseudo-code) to solve this problem. *Justify.*

(b) Given an array A of n + m elements. It is know that the first n elements in A are sorted and the last m elements in A are unsorted.

Suggest an algorithm (only pseudo code) to sort A in O(mlogm +n) worst case running time complexity. *Justify*.

(c) The processing time of an algorithm is described by the following recurrence equation (c is a positive constant):

$T(n) = 3T(n/3) + 2cn; T(1) = 0$

What is the running time complexity of this algorithm? Justify.

(d) You decided to improve insertion sort by using binary search to find the position p where the new insertion should take place.

(d.1) What is the worst-case complexity of your improved insertion sort if you take account of only the comparisons made by the binary search?

(d.2) What is the worst-case complexity of your improved insertion sort if only swaps/inversions of the data values are taken into account?

Important Notes:

- For this problem, you don't need to submit any implementation in **Java**.