Data structures - final exam

By handing in this exam the student confirms that she/he has not used any material other than the exam folder itself and her/his otherwise empty programming environment.

You can always make helper classes, helper functions, helper variables or helper parameters with default values, etc. Anything that does not change the way the base functionality is called.

Any python functionality can be used unless explicitly disallowed.
Limitations when doing array problems should be known to students.

Use the exam base to solve the exam, then re-zip that folder structure and submit it back into the assignment.

25% Multiple choice.

- The questions are in a separate assignment/quiz on Canvas.

75% Programming problems.

1. **15%**
   Implement the ADT **Set** using *arrays*.
   Finish implementing the class ***ArraySet***.
   *Normal limitations on the use of a python list apply. In short you can only use the [ ] operator with a single integer (a statement that returns an integer such as [i+n] is OK, as it sends only a single integer to the [ ] operator) and initialize with [None]*n.*
   The operations you must implement are:
   - ***add(value)***: adds the value to the set
   - ***__str__***: returns a string with all values that have been added to the set, but each unique value only once. The items should be **in order** from lowest to highest with a single space between them.

   There must be no limitations on how many items can be added to the set but a newly initialized set must also not allocate an excessive amount of memory.
   The values added to a single instance of ArraySet can be assumed to be numeric values of the same type.
   Weight: **5%** adding, **5%** __str__ and **5%** handling of duplicates.

2. **10%**
    - **5% :** Implement the operation ***print_odd*** that takes an integer value and prints every odd number from 0 up to and including the given value.
      Full marks for *recursive solution*
    - **5% :** Implement the operation ***sum_of_items*** that takes a singly-linked list with numeric values and returns the sum of the values in the list.
      Full marks for *recursive solution*

3. **15%**
   Finish implementing the following operations in the class **DLL**.
    - **3%**: move_to_prev(): moves the current location one closer to the head of the list.
      Does nothing if already at the first item.
    - **3%**: move_to_next(): moves the current location one closer to the tail of the list.
      Does nothing if already at the last item.
    - **9%**: remove(): removes the item at the current location from the list.
      Does nothing if the current location doesn't hold an item.
      The current item reference moves to the item that takes the location left by the former current item.

   Make sure that all *currently implemented operations* continue to work and return correctly.

4. **10%**
   Implement the class HashMap so that the test code works. Implement the ***__setitem__ (4%)***, ***__getitem__ (4%)*** and ***__len__ (2%)*** operations. Use any built-in python object for the buckets or implement your own bucket class.
   The hash table can be fixed at 16 buckets.
   When __getitem__ is called for item that doesn't exist, raise a *NotFoundException*.

5. **10%**
   Implement the private functionality ***_remove_node*** in the ***BSTSet*** class, so that the ***remove*** operation works correctly.

6. **15%**
   Implement the class ***DoubleKeyContainer***. It should have the following operations:
    - **5% : *add_contact(id, name, phone)***: adds this information into the collection.
      If id already in the collection, update the information.
    - **2% : *get_name_by_id(id)***: returns the ***name*** connected with this ***id***
      If id not in collection, return ***None***.
    - **2% : *get_name_by_phone(phone)***: returns the ***name*** connected with ***phone***
      If id not in collection, return ***None***.
    - **6% : *remove(id)***: removes all data connected with this id
      If id not in collection, do *nothing*.