Homework 2

Arrays

CS 5060 Intensive Programming, Fall 2012

80 points

Due: 3:59 pm September 14, 2012

Dynamic array implementation (25 pts).

Problem 0: Dynamic array

Implement a dynamic array that holds integer values and that grows when necessary to make room for new elements. Make sure your `DynamicArray` class implements the `DynamicArrayHw2` interface, available in the Files section on Canvas. You should have a function to add an element to the end of the array, and a function to remove an element from any position in the array. *The order of the elements is not important, so when an element is removed, replace it with the last element in the array (unless it is the last element).* The initial size of the array should be less than or equal to 4.

Note: All input must be read from the standard input stream, and all output must be written to the standard output stream. For this assignment, assume the input is correct.

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000000$) followed by a list of n operations. There are two types of operations: append, and delete. An append operation is represented a letter 'a' followed by a number m ($0 \leq m \leq 1000000$) to append. A delete operation is represented by a letter 'd' followed by the index i of the element to remove ($0 \leq i \leq$ array size $- 1$).

Output: For each test case output the elements of the array after all the operations have been performed. Consecutive elements should be separated by a single space.

Example:

Input:

```
3
4 a 5 a 0 a 6 a 0
6 a 3 a 0 a 6 a 5 d 0 a 0
8 a 1 a 2 a 3 a 4 d 1 a 5 a 6 d 2
```

Output:

```
5 0 6 0
5 0 6 0
1 4 6 5
```

Grading:

- Code: 5 pts
- Correct results ($n \leq 1000$): 10 pts (note that for each test case the result is unique)
- Stress test ($n \leq 1000000$): 10 pts
  - The grader will measure the time T required to solve the problem on his computer (it takes about 1 second on the instructor's computer to solve all the test cases).
  - If your code takes time $t \leq 2T$, you get 10 pts;
  - if your code takes time $t \leq 3T$, you get 7 pts;
  - if your code takes time $t \leq 4T$, you get 4 pts;
  - if your code takes time $t \leq 5T$, you get 1 pt;
  - otherwise you get 0 pts.

Solve the following problems (55 pts).

Each problem is worth 10 points. All of the problems can be solved using arrays. There is 1 extra point for each problem that you solve using the dynamic array implemented in problem 0.

Note: All input must be read from the standard input stream, and all output must be written to the standard output stream. For this assignment, assume the input is correct.

Problem 1: Reverse array

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n numbers m ($1 \leq m \leq 1000000$) separated by spaces.

Output: For each test case output the numbers in reverse order.

Example:

Input:

```
2
5 1 2 3 4 5
5 7 7 8 7 8
```

Output:

```
5 4 3 2 1
8 7 8 7 7
```

Problem 2: Split sum

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n numbers m ($1 \leq m \leq 1000000$) separated by spaces.

Output: For each test case determine if the list can be split into two in such a way that the sum of the numbers on the left is equal to the sum of the numbers on the right.

Example:

Input:

```
3
3 1 2 3
4 1 2 3 4
13 1 0 0 1 1 1 1 0 0 0 1 0 0
```

Output:

```
YES
NO
YES
```

Problem 3: Run count

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each of the next t lines starts with a number n ($1 \leq n \leq 1000$) followed by a list of n numbers m ($1 \leq m \leq 1000000$) separated by spaces.

Output: A run is a series of one or more consecutive elements with the same value. For each test case output the number of runs in the list.

Example:

Input:

```
3
5 1 2 3 4 5
6 1 2 2 3 3 3
13 1 0 0 1 1 1 1 0 0 0 1 0 0
```

Output:

```
5
3
6
```

Problem 4: Compact array

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each test case has three lines. The first line has a number n ($1 \leq n \leq 1000$). The second line has a list of n numbers m ($1 \leq m \leq 1000000$) separated by spaces. The third line contains n numbers, zero or one, where a zero means that the corresponding element of the list in the previous line is invalid, and a one means that the corresponding element is valid.

Output: For each test case output the list after removing all the invalid elements.

Example:

Input:

```
2
5
1 2 3 4 5
1 0 1 0 1
13
1 2 2 3 3 3 4 4 4 4 5 5 5
1 0 0 1 1 1 1 0 0 0 1 0 0
```

Output:

```
1 3 5
1 3 3 3 4 5
```

Problem 5: Array merge

Input: The input begins with the number t of test cases in a single line ($t \leq 100$). Each test case has two lines. The first line has a number n_1 ($1 \leq n_1 \leq 1000$) followed by a list of n_1 numbers m ($1 \leq m \leq 1000000$) separated by spaces. The second line has a number n_2 ($1 \leq n_2 \leq 1000$) followed by a list of n_2 numbers m ($1 \leq m \leq 1000000$) separated by spaces. Both lists are sorted in increasing order.

Output: For each test case merge both lists into one list that keeps all the elements in sorted order. Output the merged list. *Do not use sorting to solve this problem. There is a 5 pts deduction if you use sorting.*

Example:

Input:

```
2
3 1 3 5
3 2 4 6
4 2 7 32 45
5 3 4 5 21 38
```

Output:

```
1 2 3 4 5 6
2 3 4 5 7 21 32 38 45
```

Submission.

Submit a `zip` file with six files:

1. A code file `DynamicArray.java` with your dynamic array implementation and a code file `DynamicArrayTest.java` with the solution to problem 0 (do not submit the `DynamicArrayHw2.java` file).

2. Five code files `ReverseArray.java`, `SplitSum.java`, `RunCount.java`, `CompactArray.java` and `ArrayMerge.java` with the solutions to problems 1, 2, 3, 4 and 5, respectively.

Include your name and A number at the top of each source file. Name the `zip` file `hw02_firstName_lastName.zip`. For example, if your name is John Smith, name the file `hw02_John_Smith.zip`.