A doubly linked list – Self Testing

Test your functionalities properly and seriously.

Mark Pass or Fail for each operation. If your implementation and timing should work properly and correctly. If your timing does not match with your code or does not work, you will not get a full credit or even get a penalty for your implementation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Operations | Point | Testing | comments |
| 1 | find, more, less | 0.6 | P/F | Check the code.  find() use one while loop, but not if  Use push command to test find(). |
| 2 | push commands  push() | 0.6 | P/F | Check the code.  push() must use find() and insert(), not more than 3~4 lines of code  Don't add a new node if the position x is not found |
| 3 | pop commands  pop\_all\* | 0.6 | P/F | Test it with over 100,000 samples.  Make sure O(n), not O(n^2)  Use testing method described below |
| 4 | half() and show()\* | 0.6 | P/F | Check the correctness the middle node.  half() is used in shuffle() and show()  Test it with both odd & even number sequences.  Record the timing of half() for 20 million samples displayed using show HEAD/TAIL on exiting.  Method 1: Method 2: .  Method 2 is faster than Method 1 by \_\_\_\_\_\_\_\_\_\_\_\_ % |
| 5 | swap\_pairs | 0.6 | P/F | Check the code. It must go through the list once, not twice nor more.  Test it with both odd & even number sequences. |
| 6 | push\_sorted() | 0.6 | P/F | Use testing method described below |
| 7 | unique()\* | 0.6 | P/F | Test it with over 100,000 samples.  Make sure O(n), not O(n^2)  Use testing method described below |
| 8 | reverse() | 0.6 | P/F | Test it with over 100,000 samples.  Make sure O(n), not O(n^2) |
| 9 | randomize() | 0.6 | P/F | Test it with over 100,000 samples.  Make sure O(n), not O(n^2)  The commands sort & quicksort uses randomize(). |
| 10 | shuffle()\* | 0.6 | P/F | Check the exactness.  Test it with both odd & even number sequences. |
|  | Total | 6.0 | 2.0 | Extra 0.2 p per step for a proper testing |

**Test Hint 1: pop\_all()**

To test pop\_all(), you may need to generate a sequence that has a consecutive numbers of a certain value. You may use "push back N" command option with a negative N provided.

For example, make a sequence with ten thousands and another ten thousands of 7 samples:

* select "push\_back\_N" and enter 10,000 for random samples
* select "push\_back\_N" and enter -10000, then enter "7" for a value.
* run "pop-all" 7.

**Test Hint 2: unique()**

To test unique(), you may also need to generate a sequence that has a consecutive numbers of a certain value. You may use "push back N" command option with a negative N provided.

For example, make a sequence with ten thousands and another ten thousands of 7 samples:

* select "push\_back\_N" and enter 10,000 for random samples
* select "push\_back\_N" and enter -10000, then enter "7" for a value.
* select "randomize" or "sort" if implemented.
* run "unique"

**Test Hint 3: show(), pop(), push\_sorted(), push\_backN(), ...**

Make a sequence of numbers from 1 to 100 as shown below in a fewer steps possible. Then you may need to use all kinds of commands you implemented so far.

