A doubly linked list – Testing

Test your functionalities properly and seriously. 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 | Points | Self  Grading | comments |
| 1-1  1-2 | push and pop commands  including  pop\_all\* | 1 | 1 | * No nodes at first * Even/Odd   number of nodes   * Large/Small   number of nodes   * Using Test Hint 1 |
| 2 | half() and show()\* | 1 | 1 | * Even/Odd   number of nodes   * Large/Small   number of nodes   * show all for all N |
| 3 | sort commands  selectionSort()  insertionSort()  sorted() | 1 | 1 | * s/i/q for sorted list * s/i/q for unsorted list * s/i/q for descending order |
| 4-1 | push\_sorted()\* | 1 | 1 | * Large/Small N * Using Test Hint2 |
| 4-2 | push\_sortedN() | 1 | 1 | * Large/Small N * Using Test Hint2 |
| 5 | push\_sortedNlog()\*\* | 1 | 1 | * Large/Small N * Using Test Hint2 |
| 6 | unique()\* | 1 | 1 | * Using Test Hint2 * Nodes with same numbers |
| 7 | reverse() | 1 | 1 | * Large/Small N |
| 8 | shuffle()\*\* | 1 | 1 | * Using Test Hint2 |
| 9 | Timing Test &  Analysis | -2 to +1 | 1 | * Analysis Below. |
|  | Total | 10 | 10 |  |

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

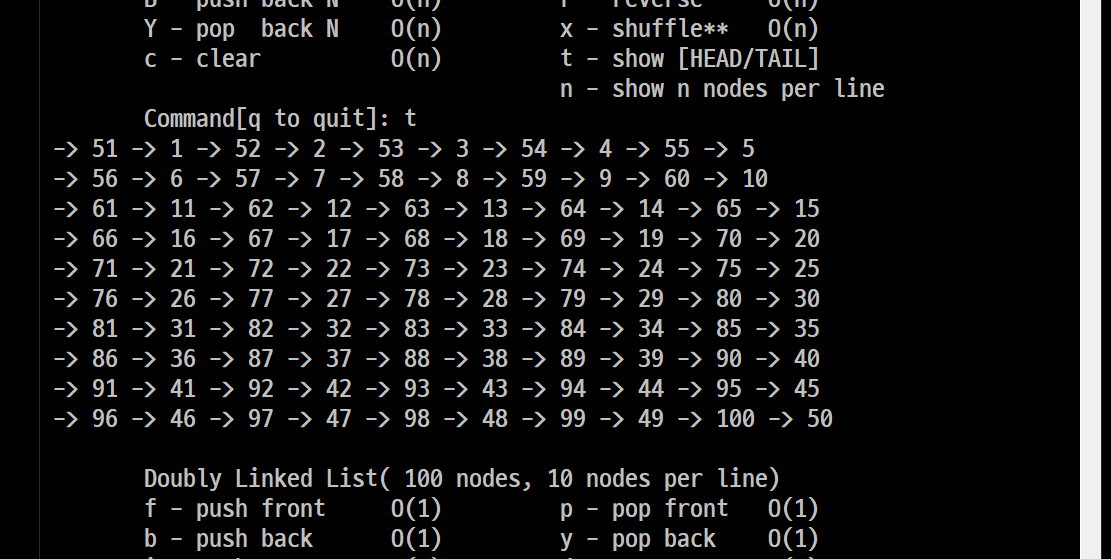
To test pop\_all(), you may need to generate a sequence that has a consecutive numbers of a certain value. You may use "push back N" command option with a negative N provided.

If you want to make a sequence of, for example, 3 2 1 4 5 ... ( ten thousands of 7)

* make 3 2 1 4 5 ... part first,
* select "push\_back\_N" command, enter "-10000" for N, enter "7" for a value.

**Test Hint 2: unique(), sort(),show(),pop(), push\_sorted(), push\_backN(), ...**

Try to 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.



**Step 9:** **Test and Analysis.** You are responsible for present a sort of table or graph with clear description in Korean such that the grader can understand the timing analysis at once.

1. Time it for 100,000 nodes and estimate the elapsed time for 1 million node for following functions. Compare the timing with listdblx.exe.
   1. Selection sort, Insertion sort and quick sort.
      1. Unsorted list case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Listdbl.exe  (my code) | | Listdblx.exe  (prof. code) | |
| Number of Nodes | 100,000 | 1Million | 100,000 | 1Million |
| Selection sort  O(n2) | 29.000(sec) | 2,900(sec)  = 48(min) | 34.936(sec) | 3,493(sec)  = 58.21(min) |
| Insertion sort  O(n2) | 17.200(sec) | 1,720(sec)  = 28.6(min) | 38.265(sec) | 3,826(sec)  = 63.76(min)  = 1.06(hour) |
| Quick sort  O(nlogn) | 0.040(sec) | 0.48(sec) | 0.054(sec) | 0.648(sec) |

Unsorted list의 경우, Selection sort, Insertion sort는 O(n2)의 시간 복잡도를 가지고 있다. 따라서 노드의 개수가 100,000일 때의 Time duration을 측정한 뒤, 100을 곱해주어 노드의 개수가 1,000,000일 때의 Time duration을 구할 수 있다.

Quick sort는 O(nlogn)의 시간 복잡도를 가지고 있다. 따라서 노드의 개수가 100,000일 때의 Time duration을 측정한 뒤, 를 곱해주어 노드의 개수가 1,000,000일 때의 Time duration을 구할 수 있다.

* + 1. Sorted list case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Listdbl.exe  (my code) | | Listdblx.exe  (prof. code) | |
| Number of Nodes | 100,000 | 1Million | 100,000 | 1Million |
| Selection sort  O(n2) | 29.750(sec) | 2,975(sec)  = 49.58(min) | 35.711(sec) | 3,571(sec) |
| Insertion sort  O(n) | 0.004(sec) | 0.04(sec) | 40.067(sec) | 400(sec) |
| Quick sort  O(n2) | 17.631(sec) | 1,763(sec)  = 29.38(min) |  |  |

Selection sort는 여전히 O(n2)의 time complexity를 가지고 있다. 따라서 100,000개의 노드일 때 얻은 time duration에 100을 곱해주면 1Million 개의 노드일 때의 time duration을 구할 수 있다.

Insertion sort는 Sorted List일 때가 Best case이다. 따라서 insertion sort의 time complexity는 O(n)로 감소한다. 따라서 노드의 개수가 100,000개일 때의 time duration에 10을 곱해주면 노드의 개수가 1Million일 때의 time duration을 얻을 수 있다.

Quick sort는 Sorted List일 때가 Worst case다. 따라서 quick sort의time complexity는 O(n2)로 급격하게 증가한다. 따라서 노드의 개수가 100,000개 일 때의 time duration에 100을 곱해주어야 노드의 개수가 1Million일 때의 time duration을 얻을 수 있다.

교수님의 코드에서 insertion sort는 정상적으로 구현되어 있는 것 같지 않았다. 왜냐하면 sorted list와 unsorted list 일 때의 performance가 비슷했기 때문이다. 또한 node의 개수를 100,000개로 quicksort를 시도하였으나 프로그램이 종료되었다. 그래서 걸린 시간은 공란으로 두었다.

1. First, measure time to merge two lists. The first one is a sorted list of 50,000 nodes and the other is a sorted array of 50,000 random numbers. Estimate the elapsed time to merge two lists. Each list has one half million of random numbers sorted: Compare the timing with listdblx.exe.
   1. push\_sortedN()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Listdbl.exe  (my code) | | Listdblx.exe  (prof. code) | |
| Number of Nodes  (total) | 100,000  (50,000 each) | 1Million  (500,000 each) | 100,000 | 1Million |
| Time Duration | 76.179(sec) | 7,617(sec)  = 126.95(min)  = 2.11(hour) | 55.12(sec) | 5,512(sec)  = 91.86(min)  = 1.53(hour) |

push\_sortedN 함수는 O(n2)의 time complexity를 가지고 있다. 따라서 총 노드의 개수가 1Million인 list를 만들기 위해서는 노드의 개수가 100,000개 일 때의 time duration에 100을 곱해주면 된다. 이는 굉장히 느린 operation이기 때문에 B에서 구할 nlogn의 버젼을 사용하는 것이 편리하다.

* 1. push\_sortedNlog()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Listdbl.exe  (my code) | | Listdblx.exe  (prof. code) | |
| Number of Nodes  (total) | 100,000  (50,000 each) | 1Million  (500,000 each) | 100,000 | 1Million |
| Time Duration | 0.047(sec) | 0.564(sec) | 0.062(sec) | 0.744(sec) |

push\_sortedNlog 함수는 quicksort를 사용하여 미리 sorting이 완료된 integer 배열을 가지고 리스트에 삽입한다. 따라서 list를 한 번만 거쳐도 동작이 완료되기 때문에 time complexity가 push\_sortedN에 비해 굉장히 줄어들게 된다.

push\_sortedNlog는 O(nlogn)의 time complexity를 가지고 있기 때문에, 노드의 개수가 100,000일 때의 time duration에 를 곱해주어 노드의 개수가 1Million일 때의 time duration을 구할 수 있다.