Java class java.util.LinkedList provides an implementation of a doubly linked list. The API documentation for it says that:

All of the operations perform as could be expected for a doubly-linked list.  
Operations that index into the list will traverse the list from the beginning or  
the end, whichever is closer to the specified index.

If you are not familiar with doubly linked lists, read the JavaHyperText file on linked lists:

http://www.cs.cornell.edu/courses/JavaAndDS/files/linkedLists.pdf

**A key question about LinkedLists**

Suppose we have declared doubly-linked list ll using this declaration:

LinkedList<Integer> ll= **new** LinkedList <>();

and then added many elements to it. An oft-asked question is to determine the number of basic steps in the algorithm to the right. The key issue is the expression ll.get(k).

// Store in s the sum of the

// values in ll

int s= 0;

for (int k= 0; k < ll.size(); k++)

s= s + ll.get(k);

The first thought is that, surely, this expression is a basic step, taking constant time. Why not? After all, an array reference b[k] is constant time, why not ll.get(k)?

The key to the answer is the statement quoted above: *Operations that index into the list will traverse the list from the beginning* …. For example, evaluation of ll.get(20)requires starting at ll[0] and using the next pointers to visit ll[1], ll[2], …, until node ll[20] is reached; then the value in that node can be returned. Thus, we can assume that evaluation of ll.get(20) takes at least 21 basic steps; more generally, evaluation of ll.get(k)takes at least k+1 basic steps.

There is an added complication. According to the quote above, evaluating ll.get(49) will start at the beginning if there are 100 or more elements in the list but will start at the end if there are fewer. For example, if the list contains exactly 50 elements, ll.get(49) will require looking only at the last node! Below, in evaluating the bound on the basic steps executed in executing the little algorithm above, we will assume that ll.get(k) always searches from the beginning of the linked list.

**Calculating the number of basic steps**

We calculate the number of basic steps in executing the algorithm, assuming that evaluation of ll.get(k) requires k+1 basic steps and that the linked list contains n elements —ll.size() = n.

s= 0; 1 basic step

k= 0; 1 basic step

k < ll.size() n+1 basic steps (evaluated n+1 times and found false once)

k++ n basic steps (executed n times)

s= s + … n basic steps (executed n times)

ll.get(k) when k = 0, 1 basic step. When k = 1, 2 basic steps, ….

when k = n-1, n basic steps. In total: 1 + 2 + … + n = n(n-1)/2 basic steps.

Summing these, we get

n(n-1)/2 + 3n + 3 or

n2/2 + 2.5n + 3 basic steps

**Discussion**

1. The formula 1 + 2 + … + n arises often when analyzing time complexity. Remember that it equals n(n-1)/2.
2. We would call this algorithm a *quadratic* algorithm because the time it takes is proportional to n2. More on this in later discussions of complexity.
3. The number of basic steps executed would appear to be less if we took into account the fact that if the index is large, the search is from the end of the list instead of the beginning. But our calculations even if we took this fact into account would still show that this algorithm is quadratic in n.