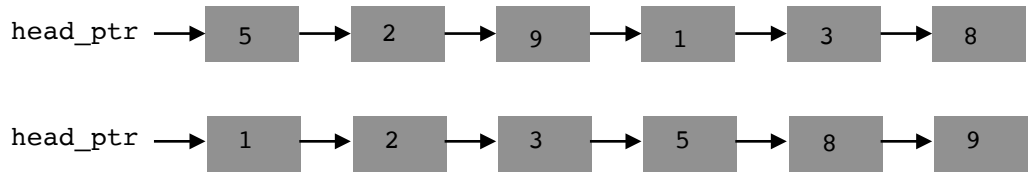
Project 3B: Recursive Selection Sort

For this project you will add another method to the **singly-linked list** class that will sort the items in the list in increasing order. This method **must be recursive** and **should not make copies of the elements/nodes in the list**.

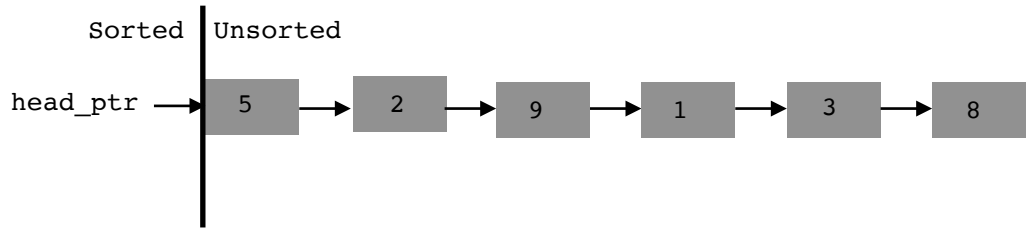


The idea: Again, you want to sort **in place (O(1) extra space)** by re-linking the chain instead of moving the items.

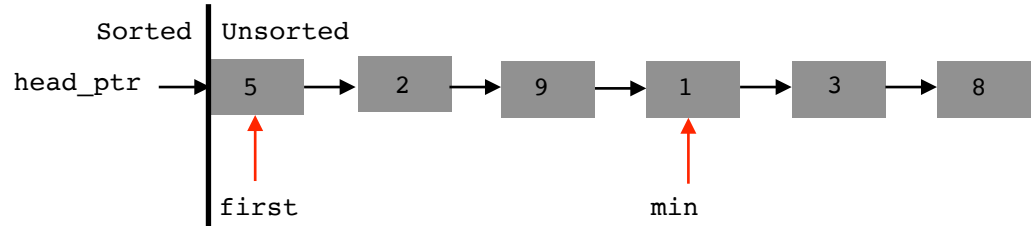
You will implement **Selection Sort**: *find the smallest item and swap it with the first unsorted item. SelectionSort the rest.*

To do so recursively, you can think of the solution to the problem as:

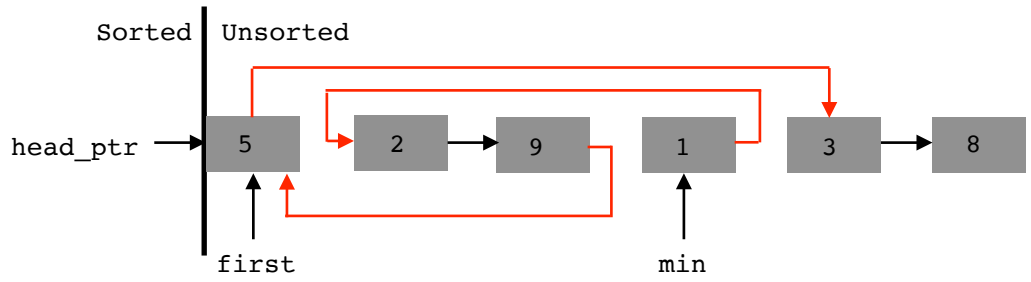
- **Consider the list to be subdivided into sorted and unsorted sublists (initially the sorted sublist is empty)**



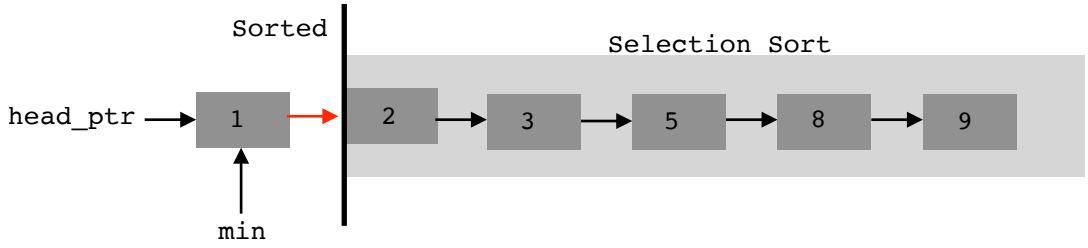
- **Find the minimum item in the unsorted sublist.**



- **Swap the min with the first in the unsorted sublist by re-linking the chain. Beware not to disconnect the chain, you will need more pointers.**



- **SelectionSort the rest of the (unsorted) list and then connect the sorted sublist to it.**



Implementation:

In order to practice safe programming and not expose the structure of the list to clients of the class via pointer parameters, you will write a **public non-recursive method `selectionSort`** with no parameters that calls a **private recursive method `recursiveSelectionSort`.**
You **must** implement these methods in **in place** (**O(1) extra space, you cannot use any additional list or other containers and cannot make copies of the nodes**).
Should you write a solution that uses iteration instead of recursion for recursiveSelectionSort and /or uses additional containers or copies of the items to sort the list, I reserve the right to take away points even if Gradescope gives you full credit. Moreover, if your name is not in your submission's comment preamble you will lose 10 points.

```
// A  wrapper to a recursive selection sort method that sorts the list in
increasing order
// @post the contents of the list are sorted in increasing order such that
//       the item at position 1 <= the item at at position 2 <= ...
//       <= item at position item_count_-1
void selectionSort();


//recursive selection sort, used for safe programming to avoid
//exposing pointers to list in public methods
//To sort the list, it relies on re-linking the chain rather than copying
data items
// @post the contents of the list are sorted in increasing order such that
//       the item at position 1 <= the item at at position 2 <= ...
//       <= item at position item_count_-1
Node<T>*  recursiveSelectionSort(Node<T>* current_first_ptr);
```

You may want to break up the work into additional helper functions. Any additional methods should be private.

Beware of special cases when re-linking the chain!

Project Notes: The List class will issue some **warnings** for `getEntry()`, because an uninitialized item is returned if the function is called with an invalid `position`. This is so because we have not discussed exception handling yet. We will be able to fix this later in the semester.

Submission:

You will find the Node and LinkedList class on Blackboard under Course Materials/ Project3. You must modify and **submit `LinkedList.hpp`** and **`LinkedList.cpp`** (**2 files**)

Your project must be submitted on Gradescope. The due date is Friday April 12 by 6pm. No late submissions will be accepted.

Have Fun!!!!!