Hw3

Thursday, October 5, 2023 2:58 PM

Don't print the starting list - start printing after first iteration of outer loop
DoublyLL()
InsertAtTail()
Print()

Insertion()
- Check if list only has one item
  - Print and return
- Set up 3 pointers: sorted (start of sorted list), unsorted (start of unsorted), insert (start of unsorted)
  - Unsorted = sorted->next (always next)

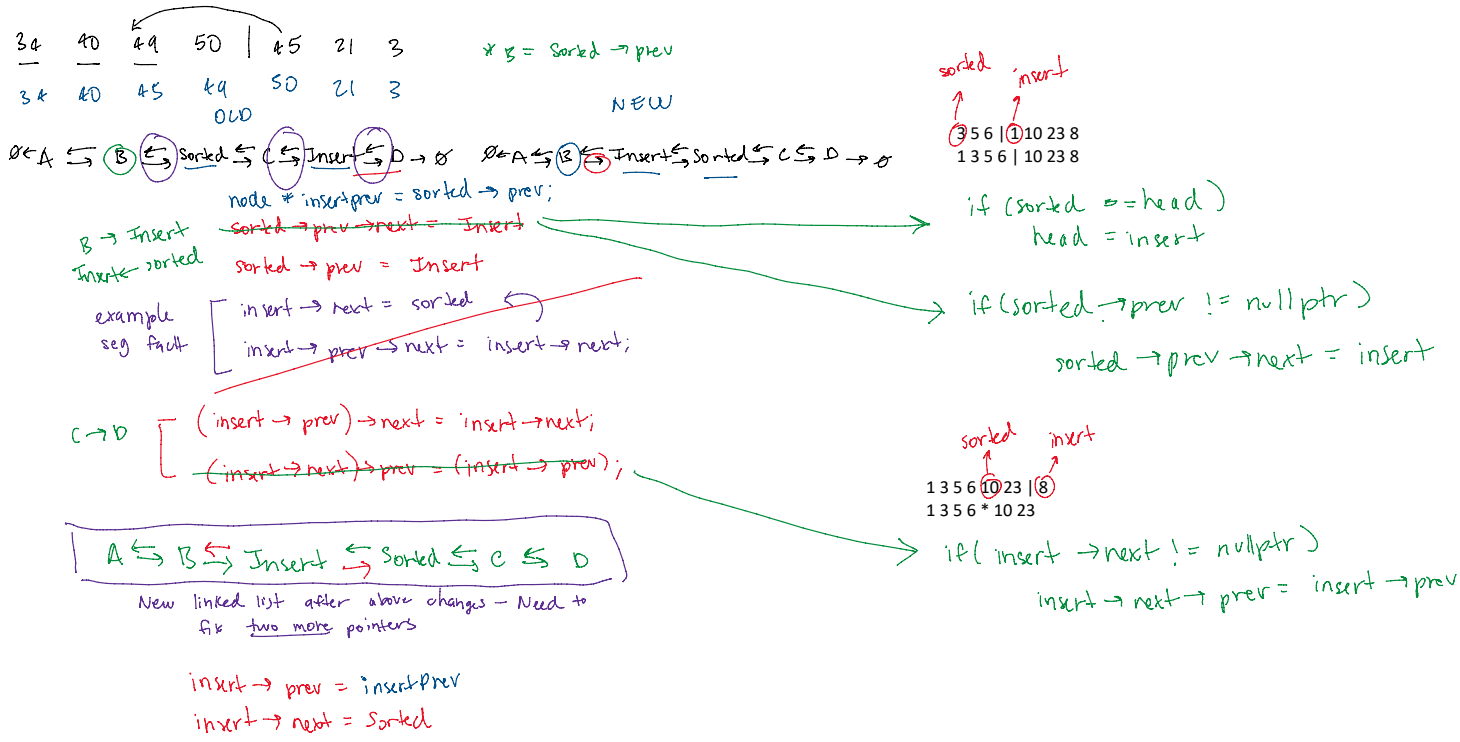
Case 1 (rearrange pointers)
34 40 49 | 45 21 1 3
34 40 45 49 | 21 1 3

Case 2 (doesn't rearrange pointers)
34 40 49 | 50 21 1 3
34 40 45 50 | 21 1 3

While(unsorted != nullptr) OUTERLOOP
- Calls print at the end of OUTER while loop

While (sorted != unsorted) INNERLOOP
- Keep traversing sorted untill **sorted->data > insert->data**
- Condition 1: Need to add *insert* node *into sorted* list
- Condition 2: *Insert* is in correct position, so move up unsorted list

Condition 1:
- Note: sorted is the position we want to insert the **insert** node at
- Need to traverse unsorted BEFORE insert's pointers are changed
  - A --> B --> Sorted --> C --> Insert --> D **becomes** A --> B --> Insert --> Sorted --> C --> D
  - What nodes do we need to change?
  - What is an edge case? (if sorted is head, if insert is at tail)
  - For simplicity, store sorted->prev
  - sorted->prev->next = insert; IF sorted != head
  - Sorted->prev = insert
    Etc...
- Set insert = unsorted after insertion is complete



```
insertionSort(){

	//check if list has one node
	If(head ->next == nullptr)
		Print()
		Return;

	Node* sorted = head;
	Node* unsorted  = sorted->next;
	Node* insert = unsorted; //insert keeps track of the unsorted's head


	//outer loop
	While( unsorted != nullptr){

		//inner loop - just to traverse sorted
		While ( sorted != unsorted ){
			//condition to check where to stop
			//sorted is at the correct position
			Sorted = sorted->next
		}

		If(sorted != unsorted){
			//traverse unsorted before insert's pointers are changed
			Unsorted = unsorted ->next;

			//pointer arrangement

			//update insert
			Insert = unsorted;

		}
		Else{

			//traverse
			Unsorted = unsorted ->next;
			Insert = unsorted;

		}

		Sorted = head;
		Print(head)
	}

}
```

Selection()

- Min based selection sort
- **Node* min**
  - Print at outer loop
  - Traverse min at outer loop
- GetSize and have outer for loop
- Several cases to account for
  - SWAPPING TWO NODES IN A LINKED LIST
    - Check if one or more nodes are head or tail
    - Check if nodes are adjacent