Sample Solutions for Sample Past Exam 2 Questions

Q1 ***(a)*** The *linked list implementation* would be more efficient for building the list.

Inserting a new item into a sorted list of items implemented using an array in general requires that some if not all existing items be shifted to make room for the new item – ***insertion anomaly***. The array may also need to be resized (an ***O(n)*** operation) first if it is already full – ***resizing woe***. Only some pointers need to be changed when inserting a new item into a sorted list of items implemented using linked list. A new node has to be created each time but the operation is ***O(1)***.

(b) The *array implementation* would be more efficient for answering questions of the type "What is the `i`th highest score in the class?".

With a sorted array, the `i`th highest score (for any random but valid `i`) is contained in the element whose index is `(used - i)`, which can be efficiently accessed in ***O(1)*** time using the usual array notation – ***random access***. With a sorted linked list, the `i`th highest score is contained in the `(many_nodes - i + 1)`th node, the access of which requires that the list be traversed starting from the first node until the desired node is reached (an ***O(n)*** operation) – ***no random access***.

Q2

```
bool IsEqual(Node* head1, Node* head2)
{  // 1st 2 if statements can be excluded
   if (head1 == 0 && head2 == 0) return true;
   if (head1 == 0 || head2 == 0) return false;
   while (head1 != 0 && head2 != 0)
   {
      if (head1->data != head2->data) return false;
      head1 = head1->link;
      head2 = head2->link;
   }
   return head1 == 0 && head2 == 0;
}
```

Q3

```
void SwapGivenNodeWithNextNode(Node* head, Node* givenNodePtr)
{  // assume: list has @ least 3 nodes & given node isn't head node or tail node
   while (head->link != givenNodePtr)
      head = head->link;
   head->link = givenNodePtr->link;
   givenNodePtr->link = givenNodePtr->link->link;
   head->link->link = givenNodePtr;
}
```

Q5

```
// 3 stacks s1, s2 & s3: - s1 gets relevant characters
//                       - s2 buffer that helps construct s3
//                       - s3 gets relevant characters in reverse order
while (there's still character left in input to read)
   read one character c
   if ( isalpha(c) )
      c = toupper(c)
      s1.push(c)
      s2.push(c)
while ( ! s2.empty() )
   s3.push( s2.top() )
   s2.pop()
while ( ! s1.empty() )
   if ( s1.top() != s3.top() )
      output "is not palindrome" message & return
   s1.pop()
   s3.pop()
output "is palindrome" message
```

Q4

Symbol	Stack	Comments
(	(	
2	(	2
+	(+	2
3	(+	2 3
*	(+ *	2 3
4	(+ *	2 3 4
)		2 3 4 * +
*	*	2 3 4 * +
(	* (	2 3 4 * +
5	* (	2 3 4 * + 5
*	* (*	2 3 4 * + 5
(	* (* (	2 3 4 * + 5
7	* (* (	2 3 4 * + 5 7
-	* (* (-	2 3 4 * + 5 7
6	* (* (-	2 3 4 * + 5 7 6
)	* (*	2 3 4 * + 5 7 6 -
+	* (+	2 3 4 * + 5 7 6 - *
9	* (+	2 3 4 * + 5 7 6 - * 9
)	*	2 3 4 * + 5 7 6 - * 9 +
-	-	2 3 4 * + 5 7 6 - * 9 + *
8	-	2 3 4 * + 5 7 6 - * 9 + * 8
		2 3 4 * + 5 7 6 - * 9 + * 8 -

Symbol	Stack	Comments
2	2	
3	2 3	
4	2 3 4	
*	2 12	3 * 4 = 12
+	14	2 + 12 = 14
5	14 5	
7	14 5 7	
6	14 5 7 6	
-	14 5 1	7 - 6 = 1
*	14 5	5 * 1 = 5
9	14 5 9	
+	14 14	5 + 9 = 14
*	196	14 * 14 = 196
8	196 8	
-	188	196 - 8 = 188
		188 --> desired result

Q6 **432101234**
