**Data Structure\_2071035 Lee Somin**

**Technical Report – MergeList.cpp**

*Theorical Explanation of Functions in ‘MergeList.cpp’*

**Time Complexity:**

Let’s say that n is the sum of number of elements of listA and listB.

There is An operation when assigning the elements to the lists, Bn operation when merging two lists, and Cn operation when displaying the list. (A, B, C is constant value.)

T(n) = (A+B+C)n + *c*

∴T(n)=O(n)

The code has time complexity of n.

**typedef element**

This typedef defines the data type of the elements of list.

**typedef struct ListNode**

This structure contains the actual information of node, the data and the link.

**typedef struct ListInfo**

This structure contains the double pointer to the head of the list and double pointer to the end of the list. This is for accessing to the node of head and tail easily, and for determining a list from another.

**init**

Input: ListInfo\* list

Return: non

Time complexity: O(1)

This function allocates the memory to the phead and plast, and initializes the list by assigning NULL to \*phead and \*plast.

**insert\_last**

Input: ListInfo list, element new\_data

Return: non

Time complexity: O(1)

The function allocates the memory to new node, assigns the given data ‘element new\_data’ to ‘new\_node->data’, and set the link of the new node to NULL since it is inserted to last. If the list is blank, add new node to the headpointer and ‘\*plast’ as the first node. Otherwise, add the node at the end of the list by updating last node’s link to new node and the address of the last node to new node.

**display\_list**

input: ListInfo list

return: non

time complexity: O(n)

This function displays the linked list until the last node. During the while loop, the data of ‘current\_node’ is printed and the ‘current\_node’ is updated by it’s link which points to the next node. “ “is only printed when current node is not the last node(updated link in not NULL).

**merge\_list**

input: ListInfo listA, ListInfo listB

return: ListInfo listMerged

time complexity: O(n)

This function preforms the merge operation with given two list ‘listA’ and ‘listB’. It declares the list ‘listMerged’ and initializes it using ‘init()’. Then the ‘cntA’ and ‘cntB’ is defined which points to the next node that should be added to the merged list from each list. The while loop continues until both ‘cntA’ and ‘cntB’ reaches the end of each list. When all elements of one of the two lists are added to the 'listMerged', all the nodes left in the other list are all inserted to the end of 'listMerged' in ascending order. Otherwise, the node with the bigger data is inserted at the end of the ‘listMerged’ and updates itself by its link. In this way, the elements of the ‘listMerged’ is sorted in ascending order. When the while loop ends, the function returns the list ‘listMerged’.

**main()**

input: non

Return: non

The main function first declares and initializes the two list ‘listA’ and ‘listB’ using ‘init()’. Then, two array of element of two lists are defined. Through for loop, each element is assigned to ‘listA’ and ‘listB’ as a node in ascending order. Then, the ‘mergedList’ is assigned as the merged list of ‘listA’ and ‘listB’, by ‘merge\_list()’. Then, ‘listA’, ‘listB’, ‘mergedList’ is printed out by ‘display\_list()’.