ListFLType\* KWayMerge(ListFLType\*\* listArray, int index, int k, int n, Operation\* opAttr, Operation\* opComp)

{

/\*

\* ~ How it works?

\* Interclasarea listelor consta in "scoaterea" celui mai mic element (la nivelul tutudor listelor) din heap

\* si adaugarea lui intr-o lista finala ce pastreaza elementele din toate listele, sortate crescator

\*

\* La extragerea minimului din heap are loc, de fapt, extragerea efectiva a elementului din prima lista din heap si trecerea la

\* elementul urmator

\* Daca lista se goleste se inlocuieste varful heapului cu ultimul element si se reface structura de heap

\*

\*/

ListFLType\* sortedList = (ListFLType\*)malloc(sizeof(ListFLType));

sortedList->first = sortedList->last = NULL;

buildHeap(listArray, k, opAttr, opComp); // O(k) ( nu e influentat de n pentru ca initial heapul se construieste

//tinand cont doar de primul element din fiecare lista

while (k > 0)

{

ListType\* min = extractMinHeap(listArray, &k, opAttr, opComp); // O(log k) (pentru un singur apel) \* n ( numarul total de apeluri facute)

if (min != NULL) // ==> O(n log k)

{

insertLast(&sortedList, min->key, min->index, opAttr, opComp); //O(1)

}

}

if (isListSorted(sortedList->first) == false) // Test aditional, nu tine de algoritm :)

{

printf("\nEroare pentru KWayMerge! Sirul nu e sortat.\n");

return NULL;

}

return sortedList;

// Complexitate totala: O(nlog k) + O( k )

}

ListFLType\* TwoWayMerge(ListFLType\* list1, ListFLType\* list2, int index, int n)

{

ListFLType\* sortedList = (ListFLType\*)malloc(sizeof(ListFLType));

sortedList->first = sortedList->last = NULL;

Operation justDemo = p.createOperation("justDemo", 0);

while (list1->first != NULL && list2->first != NULL)

{

if (list1->first->key < list2->first->key)

{

insertLast(&sortedList, list1->first->key, list1->first->index, &justDemo, &justDemo);

list1->first = list1->first->next;

}

else if(list1->first->key > list2->first->key)

{

insertLast(&sortedList, list2->first->key, list2->first->index, &justDemo, &justDemo);

list2->first = list2->first->next;

}

else

{

insertLast(&sortedList, list1->first->key, list1->first->index, &justDemo, &justDemo);

list1->first = list1->first->next;

insertLast(&sortedList, list2->first->key, list2->first->index, &justDemo, &justDemo);

list2->first = list2->first->next;

}

}

while (list1->first != NULL)

{

insertLast(&sortedList, list1->first->key, list1->first->index, &justDemo, &justDemo);

list1->first = list1->first->next;

}

while (list2->first != NULL)

{

insertLast(&sortedList, list2->first->key, list2->first->index, &justDemo, &justDemo);

list2->first = list2->first->next;

}

if (isListSorted(sortedList->first) == false)

{

printf("\nEroare pentru TwoWayMerge! Sirul nu e sortat.\n");

return NULL;

}

return sortedList;

}

void doKMerge(int n, int k, int caseK)

{

int\* arr = (int\*)malloc(sizeof(int) \* n);

ListFLType\*\* listArray = (ListFLType\*\*)malloc(k \* sizeof(ListFLType\*));

ListFLType\* unsortedList = (ListFLType\*)malloc(sizeof(ListFLType));

unsortedList->first = NULL;

unsortedList->last = NULL;

int rest = n-k;

/\*

fiecare lista are cel putin 1 element deci vor ramane n-k elemente de distribuit random prin liste

\*/

printf("Cele %d liste sunt formate din urmatoarele elemente: \n",k);

printf("Index\tSize\t Array\n");

int i;

Operation justDemo = p.createOperation("justDemo", 0);

int size;

int stop = 0;

srand(time(NULL));

rest = n - k;

for (i = 0; i < k - 1 && rest; i++)

{

size = rand() % (rest - 0 + 1) + 0;

rest -= size;

size++;

FillRandomArray(arr, size, 0, 100, false, ASCENDING);

listArray[i] = (ListFLType\*)calloc(size, sizeof(ListFLType));

for (int j = 0; j < size; j++)

{

insertLast(&listArray[i], arr[j], i, &justDemo, &justDemo);

insertLast(&unsortedList, arr[j], i, &justDemo, &justDemo);

}

printf("%d\t%d\t: ", i, size);

showList(listArray[i]->first);

}

for (i = i; i < k; i++)

{

size = rest + 1;

FillRandomArray(arr, size, 0, 100, false, ASCENDING);

listArray[i] = (ListFLType\*)calloc(size, sizeof(ListFLType));

for (int j = 0; j < size; j++)

{

insertLast(&listArray[i], arr[j], i, &justDemo, &justDemo);

insertLast(&unsortedList, arr[j], i, &justDemo, &justDemo);

}

printf("%d\t%d\t: ", i, rest + 1);

showList(listArray[i]->first);

}

ListFLType\* sortedList;

if (caseK != 2)

{

Operation justDemo = p.createOperation("justDemo", 0);

sortedList = KWayMerge(listArray, 0, k, n, &justDemo, &justDemo);

}

else

{

sortedList = TwoWayMerge(listArray[0], listArray[1], 0, n);

}

printf("\n\_\_\_\_\_\_\_\_\n");

printf("Before Merge: ");

showList(unsortedList->first);

printf("\nAfter Merge: ");

showList(sortedList->first);

}