CG1103 GLab 5 and overall justification Report Group C02-5

Choice of linked list with justification

* This doesn’t really matter to us. This is because we use linked list just to “hang” the products. When we do sorting, we copy the items to array of structures (consist of the product pointer and its current position in list).
* If we were to justify, for our case, linked list is “slightly” better. This is due to whenever we add an item ( a product pointer), the contents of the nodes are less( no prev pointer, etc).,but this shouldn’t have significant effect.
* To sort a linked list we need to used an iterator class.(which Uncle Soo discouraged us to used as he mentioned in one of the tutorials). IF we were REALLY to sort using iterator, DLL is of course faster, since it can travel in both directions.
* Another justification- we copy to array, and then do sorting. This doesn’t change the position of the product pointer in the list. This is very important. This is because raw data positions are MEANT not to be changed. And this is one of the requirements in lab also. (see Testing 1st point on page 2 of labsheet). We only sort the position of pointers in the array. We fulfill the requirement.

Choice of sorting algorithm with justification

* We chose merged sort.
* It is fast, and the speed is constant, since it doesn’t have worst case and best case. This is suitable to sort a large amount of data items. Well, HYPERMARKET is meant to be BIG, right? And whenever a list is given, we never know whether it is sorted beforehand or not. Merge sort is “play safe+ fast”.
* We used an array of structures to store pointers and its position. Thus we are able to sort by comparison, through the pointers copied.
* Complexity of merge sort itself (for an array) is n\*logn. Again, to re-mention that we do not have the knowledge of using iterator class, so we do not used linked list. If we are to sort according to linked list ***without iterator class***, the algorithm for the sorting itself is n^2\*logn. This is because the TOTAL cost of traversal of linked list is already N^2. And we do it logn times. So is N^2\* logn.
* Complexity of copytostruct() is n. This is because in ListBase class we have an array of template, T[]. Meaning, whatever you store in list, the array will be filled with the list. In this case our list store product pointers. So the array contains product pointers. Then in Statistic Class I create a structure to store one pointer and its position. We declare the structure in private, and make an array of those particular structures. Copy from ***T[]*** to ***struct[].productpointers*** need N times. So complexity is N. *Note: T[] is created dynamically.*
* Complexity of returnpointer() is also N. Since copytostruct() is a method in statistic class, there is no way that hypermarket can access the private attribute in statistic. So we need to recopy the pointers. So it takes another N.
* So total complexity to “ sort and get result” is N+N + N\*logN, which is N.
* Anyway the sorting itself is only n+ nlogn.

Complexity to sort by worth.

* Procedure to sort by worth is
  1. Copytostruct(); cost = N;
  2. Sortsetstockrank(); cost =Nlog N
  3. Sortsetpricerank(); cost =NlogN
  4. Sortsetworthrank(); cost= NlogN
  5. Returnpointers(); cost =N

So the total complexity to get the final result is 3NlogN +2N;

Which is still, N

Time

* Xl-1430ms
* L -120ms
* M -10ms
* S-2 ms
* Xs-0ms.

System run algorithm.

UI call myhypermarket. Myhypermarket calls mystatistics. Statistics class will do the sorting. Once the sorting is done, an array of pointers is created to store all the pointers with respects to their position. Hypermarket then can use Product\*\* to hook to this array, and do other stuffs.

Some additional verdict, from our group:

Correct us if we are wrong, but based on the knowledge we have, array is the fastest in terms of accessing. Due to this feature using array in comparison sorting is the best. Nevertheless, if we have an iterator class in linked list, which is to achieve the ***“O(1) in accessing 1 item”*** will produce the same result as using an array.

Another advantage of using an array is the data are always intact. That means in the memory in the computer itself it is always “linked” together. This is not so for the case of linked list. What we learnt is pointers are REALLY dangerous (and that’s why java tries to hide it). So if we were given to choose, we will choose array too.

Bad thing about using a dynamic array is wastage of memory. Well in our group opinions, if you plan your programming properly, there is no need to create a dynamic array with size larger than it is required. Unless it is on-going added list, then you need to re-declare the array and keep track of the size, and update when necessary.

Use array to mergesort consumes memory too, i.e not in place sorting. Well there is no perfect sorting algorithm; we need to sacrifice something to gain some other thing.