Algorithm Description

Bunch sort is an array sorting algorithm. It is similar to linked lists and binary trees in that it requires the assembly of nodes into an organized structure.

Bunch sort works by inserting array elements into a bunch string. The elementary piece of a bunch string is the bunch node. Similar to a binary tree node, the bunch node has three members. Those members are the data member, the primary pointer, and the secondary pointer:

An empty bunch list consists of a single empty node, the head node. Numbers from an unordered array are initially added to the bunch list as they would be if they were inserted into a linked list: the algorithm walks down the string, comparing a new number to the number in each existing node until it finds the proper location to insert the new number. However, if the bunch sort algorithm walks too far, it pauses and "bunches" up the preceding nodes that it just examined. This means that the next time the algorithm has to walk down the list, it will have a shorter path to walk.

Inserting an array of ascending numbers into a bunch list (with a pre-specified maximum "walk" of 4) would look like this:
It should be noted that for the numbers 1 through 4, the algorithm continues appending new numbers to the list just as it would in a linked list. However, when the algorithm inserts 5, it has to "walk" past nodes head-3 to get to node 4. Since the algorithm had to walk past 4 nodes, when it reaches node 4 it pauses, bunches up all the nodes from head to 4, and then resumes walking down the string.

Next, when the algorithm needs to insert the number six, it doesn't have to walk the whole length of the string. The primary pointer in the head node now points to node 4, so the algorithm can jump immediately to four and continue normally. Thus, the algorithm has fewer comparisons to make than if it were to walk down an ordinary linked list.

Bunch sort may encounter the following unique scenarios if it is asked to order an unordered array:

- Current node is the end of the string (primary and secondary ptrs are null):

- Node addressed by primary pointer has a greater data element (DE) than the new number, secondary pointer is null:

- Node at primary pointer has a greater DE, node at secondary pointer has a greater DE:

- Node at primary pointer has a greater DE, node at secondary pointer has a lesser DE:

- Node at primary pointer has a lesser DE:

- Move to the next node, bunching if necessary:
Performance

Just as with a linked list, it is possible to implement a stable version of bunch sort by changing the comparison operator for the next data element from "less than" to "less than or equal to."

Performance for bunch sort is similar to linked list sort in some cases and superior in other cases. For an array that is in reverse order, both bunch sort and linked list will require \( n \) comparisons and will have \( O(n) \) performance because they will only have to repeatedly prepend new numbers to the beginning of the growing list. Thus, a reverse-ordered array is the best case for both bunch sort and linked list.

For ordered arrays, however, bunch sort's performance is markedly superior to that of a linked list. Because bunch sort repeatedly shortens the path to the back of the string, it will never have to perform more than \( w + 1 \) comparisons for a new number, where \( w \) is the number of nodes that the algorithm is allowed to walk past. Because bunch sort will never have to perform more than a fixed number of comparisons for each new number, its performance for ordered lists is also \( O(n) \), which is superior to linked list's \( O(n^2) \).

It is also worth noting that bunch sort's performance is superior to a binary tree's performance for ordered or reverse ordered arrays. For both ordered and reverse ordered arrays, binary trees devolve into linked lists and require \( O(n^2) \) comparisons. This is not the case with bunch sort.

Remaining Questions

The following research questions regarding bunch sort are unanswered at this point and would benefit from further exploration:

- What is the optimum "walking" length for the bunch sort algorithm? What is that optimum length's relationship to the size of the array to be sorted?
- What is the worst case scenario for bunch sort?
- What is the big-O worst case performance for bunch sort?