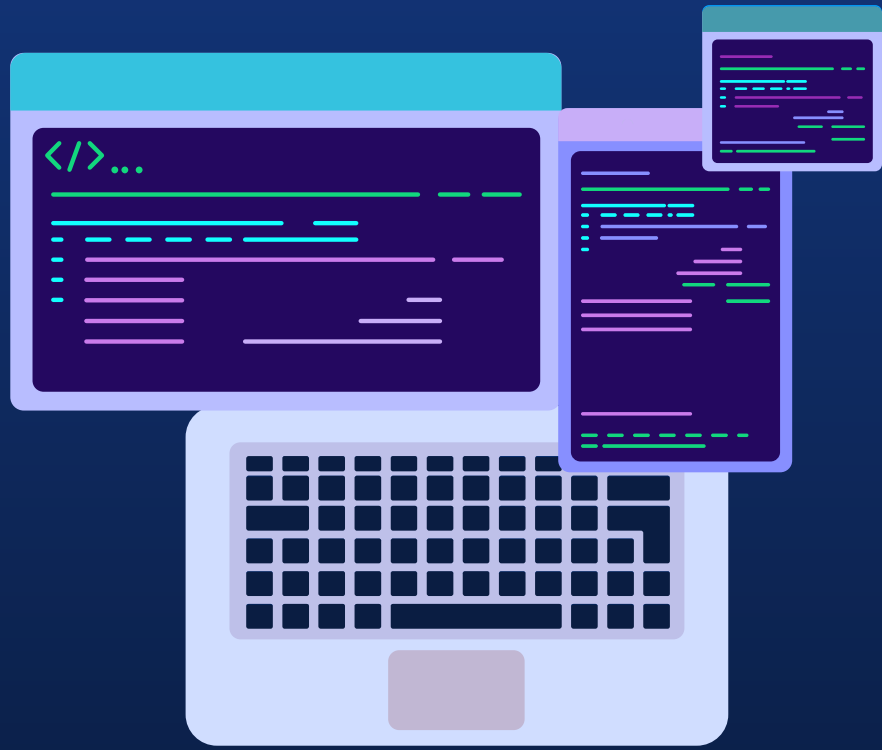
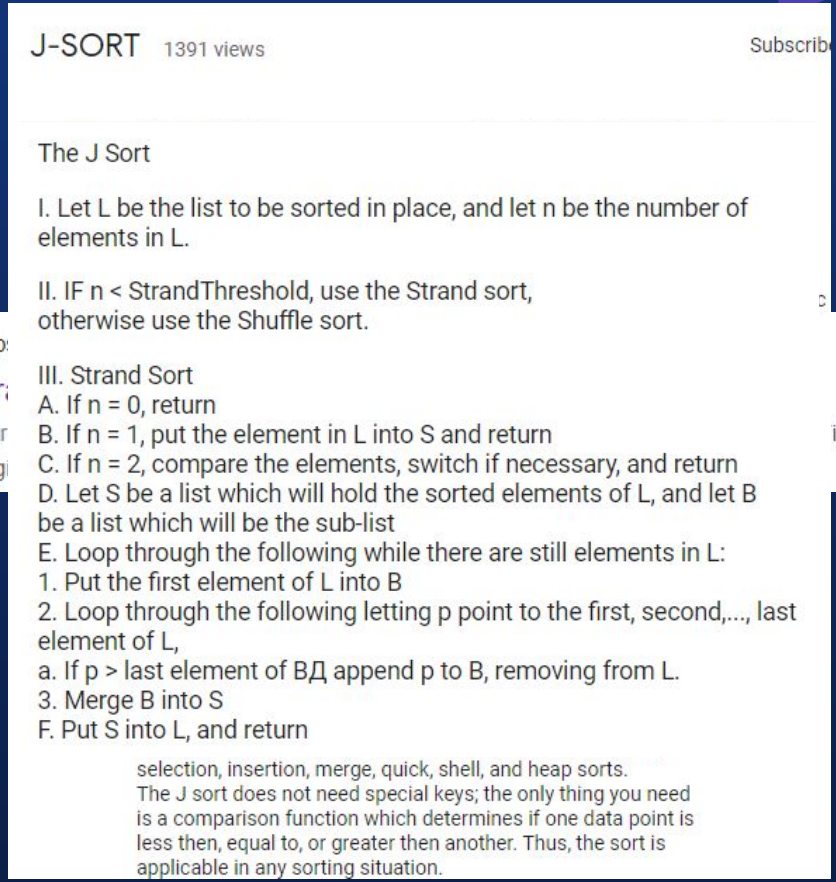
STRAND SORT

Team Summer



Creation of Strand Sort

- Creator and year are both unknown.
- Earliest website posting was on March 15, 2006.
- Earliest mention was on November 26, 1997, in an email that contains John Cohen proposing the J-Sort.



J-SORT 1391 views Subscrib

The J Sort

I. Let L be the list to be sorted in place, and let n be the number of elements in L.

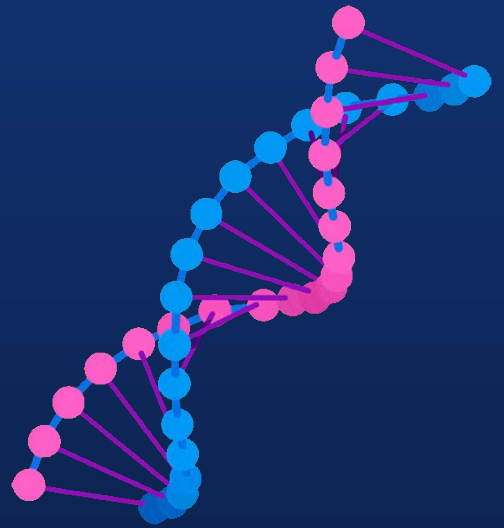
II. IF n < StrandThreshold, use the Strand sort,
otherwise use the Shuffle sort.

III. Strand Sort
A. If n = 0, return
B. If n = 1, put the element in L into S and return
C. If n = 2, compare the elements, switch if necessary, and return
D. Let S be a list which will hold the sorted elements of L, and let B be a list which will be the sub-list
E. Loop through the following while there are still elements in L:
1. Put the first element of L into B
2. Loop through the following letting p point to the first, second,..., last element of L,
a. If p > last element of BД append p to B, removing from L.
3. Merge B into S
F. Put S into L, and return

selection, insertion, merge, quick, shell, and heap sorts.
The J sort does not need special keys; the only thing you need is a comparison function which determines if one data point is less then, equal to, or greater then another. Thus, the sort is applicable in any sorting situation.

Strand Sort

- It is a comparison sorting algorithm that uses recursion to sort items in a list in ascending order.
- It repeatedly pulls out a series of elements from the unsorted list into a sublist to be sorted, then merges them into the result array.
- It is derived from Selection Sort.



Complexity: Time and Space

O(n)

Best

When the list is already sorted

O(n log n)

Average

Average Case

O(n^2)

Worst

When the the order of the list is sorted in reverse

Internet Variations

Internet Variations of Strand Code include:

- Using Linked List as the Data Structure
- Using Stack as the Sub List
- Using Queue as the Sub List

Simulation for Strand Sort

There are 3 lists used in Strand Sort:

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1. Move the first element of the unsorted list to the sub list

Unsorted List:

10	34	2	17	56	43	29	11	6

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1. Move the first element of the unsorted list to the sub list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

34	2	17	56	43	29	11	6	

Sub List:

10								

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

2	17	56	43	29	11	6		

Sub List:

10	34							

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

2. Traverse the unsorted list. For every element, compare and check if it is greater than is greater than the last inserted element of the sub list. If yes, remove the element from the unsorted list and append of the sublist. If no, move to the next element.

Unsorted List:

2	17	43	29	11	6			

Sub List:

10	34	56						

Sorted/Output List:

Simulation for Strand Sort

3. Merge the sub list into the sorted/output list.

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

3. Merge the sub list into the sorted/output list.

Unsorted List:

Sub List:

Merge

Sorted/Output List:

Simulation for Strand Sort

3. Merge the sub list into the sorted/output list.

Unsorted List:

2	17	43	29	11	6			

Sub List:

Sorted/Output List:

10	34	56						

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

1st Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

29	11	6						

Sub List:

2	17	43						

Merge

Sorted/Output List:

10	34	56						

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

Process of Merging

Sub List:

Sorted/Output List:

Merged List:

Simulation for Strand Sort

2nd Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

29	11	6						

Sub List:

Sorted/Output List:

2	10	17	34	43	56			

Simulation for Strand Sort

3rd Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

Simulation for Strand Sort

4th Recursive Call

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

6								

Sub List:

Sorted/Output List:

2	10	11	17	29	34	43	56	

Simulation for Strand Sort

4. Recur for the remaining elements in the unsorted list and current elements in the sorted/output list

Unsorted List:

Sub List:

Sorted/Output List:

2	6	10	11	17	29	34	43	56

Internet Code

```
void strandSort(list<int> &ip, list<int> &op)
{
    if (ip.empty())
        return;

    list<int> sublist;
    sublist.push_back(ip.front());
    ip.pop_front();

    for (auto it = ip.begin(); it != ip.end(); ) {

        if (*it > sublist.back()) {
            sublist.push_back(*it);
            it = ip.erase(it);
        }

        else
            it++;
    }

    op.merge(sublist);

    strandSort(ip, op);
}
```

Streamline Code

```
void StrandSort(int A[],int len) {
    List mirA,*trav,sub,*travSub,solFirst=NULL,solSec,*travSol,temp;
    int i;

    for (trav=&mirA,i=0 ; i<len ; i++){
        *trav = (List)malloc(sizeof(node));
        if (*trav!=NULL){
            (*trav)->val = A[i];
            trav = &(*trav)->link;
        }
    }
    *trav = NULL;
    while (mirA!=NULL){
        sub = mirA;
        mirA = mirA->link;
        for (trav=&mirA,travSub=&sub ; *trav!=NULL ; ){
            if ((*trav)->val>=(*travSub)->val){
                travSub = &(*travSub)->link;
                *travSub = *trav;
                *trav = (*trav)->link;
            }else {
                trav = &(*trav)->link;
            }
        }
        (*travSub)->link = NULL;
        for (travSol=&solSec ; sub!=NULL && solFirst!=NULL ; travSol=&(*travSol)->link){
            if (sub->val<solFirst->val){
                *travSol = sub;
                sub = sub->link;
            }else {
                *travSol = solFirst;
                solFirst = solFirst->link;
            }
        }
        *travSol = sub!=NULL ? sub : solFirst;
        solFirst = solSec;
    }
    for (i=0 ; solFirst!=NULL ; i++,solFirst=solFirst->link,free(temp)){
        A[i] = solFirst->val;
        temp = solFirst;
    }
}
```

End of Presentation :)

References

Kadam, P., & Kadam, S. (2014). Root to Fruit (2): An Evolutionary Approach for Sorting Algorithms. Oriental Journal of Computer Science and Technology, 7(3), 369-376.

Pandey, R. C. (2008). Study and Comparison of various sorting algorithms (Doctoral dissertation).

tutorialspoint.dev. (n.d.). Strand Sort. Retrieved from https://tutorialspoint.dev/algorithm/sorting-algorithms/strand-sort?fbclid=IwAR1N76NwevYHHuJZfgB7G4-SpZ_8ZyVaUHcW8ttv0SzxYW-jTnko4ujZz-M

Erokhin, S. (1997, November 26). J-SORT. Retrieved from https://groups.google.com/g/fido7.ru.algorithms/c/dL4SAH96OPY/m/s4oABNtMCCYJ?pli=1&fbclid=IwAR3IuUS5YS3X2w3DZfBEqyj6golkr3yrkggwJphYGxlWBTL8oN8y-C1JUhY