java.util.Collections.disjoint() Method

<http://www.java2s.com/Tutorials/Java/java.util/Collections/Java_Collections_disjoint_Collection_lt_gt_c1_Collection_lt_gt_c2_.htm>

java.util.Collections.frequency() Method

<https://www.tutorialspoint.com/java/util/collections_frequency.htm>

### Sorting algorithms[[edit](https://en.wikipedia.org/w/index.php?title=Best,_worst_and_average_case&action=edit&section=5" \o "Edit section: Sorting algorithms)]

*See also:*[*Sorting algorithm § Comparison of algorithms*](https://en.wikipedia.org/wiki/Sorting_algorithm#Comparison_of_algorithms)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Data structure** | **Time complexity:Best** | **Time complexity:Average** | **Time complexity:Worst** | **Space complexity:Worst** |
| Quick sort | Array | O(*n* log(*n*)) | O(*n* log(*n*)) | O(*n*2) | O(1) |
| Merge sort | Array | O(*n* log(*n*)) | O(*n* log(*n*)) | O(*n* log(*n*)) | O(n) |
| Heap sort | Array | O(*n* log(*n*)) | O(*n* log(*n*)) | O(*n* log(*n*)) | O(1) |
| Smooth sort | Array | O(*n*) | O(*n* log(*n*)) | O(*n* log(*n*)) | O(1) |
| Bubble sort | Array | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Insertion sort | Array | O(*n*) | O(*n*2) | O(*n*2) | O(1) |
| Selection sort | Array | O(*n*2) | O(*n*2) | O(*n*2) | O(1) |

### Data structures[[edit](https://en.wikipedia.org/w/index.php?title=Best,_worst_and_average_case&action=edit&section=6)]

*See also:*[*Search data structure § Asymptotic amortized worst-case analysis*](https://en.wikipedia.org/wiki/Search_data_structure#Asymptotic_amortized_worst-case_analysis)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Data structure** | **Time complexity: Avg: Indexing** | **Time complexity: Avg: Search** | **Time complexity: Avg: Insertion** | **Time complexity: Avg: Deletion** | **Time complexity: Worst: Indexing** | **Time complexity: Worst: Search** | **Time complexity: Worst: Insertion** | **Time complexity: Worst: Deletion** | **Space complexity: Worst** |
| Basic Array | O(1) | O(*n*) | – | -– | O(1) | O(*n*) | – | – | O(*n*) |
| Dynamic array | O(1) | O(*n*) | O(*n*) | – | O(1) | O(*n*) | O(*n*) | – | O(*n*) |
| Singly linked list | O(*n*) | O(*n*) | O(1) | O(1) | O(*n*) | O(*n*) | O(1) | O(1) | O(*n*) |
| Doubly linked list | O(*n*) | O(*n*) | O(1) | O(1) | O(*n*) | O(*n*) | O(1) | O(1) | O(*n*) |
| Hash table | - | O(1) | O(1) | O(1) | – | O(*n*) | O(*n*) | O(*n*) | O(*n*) |
| Binary search tree | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | – | O(*n*) | O(*n*) | O(*n*) | O(*n*) |
| B-tree | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | O(*n*) |
| Red-black tree | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | O(*n*) |
| AVL tree | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | – | O((log *n*)) | O((log *n*)) | O((log *n*)) | O(*n*) |

# Why is selection sort said to be better than bubble sort though both are of order n square?

By first scanning the entire list before locating the exact pair of numbers to swap, only two writes to memory are performed by Selection Sort for each O(n) scan, whereas Bubble Sort does writes on each and every comparison. So Selection Sort does O(n) writes to memory whereas Bubble Sort does O(n^2) writes.

Need to implement below later

1. <http://www.geeksforgeeks.org/find-the-maximum-of-minimums-for-every-window-size-in-a-given-array/> (stack)