PROJECT DATA STRUCTURE.

**Made by:** Yomna Muhammad AlaaElDin Nouman Eskander.

**ID:** 20201323973.

This report compares Merge sort with selection sort in terms of memory usage and time usage. Firstly, I’ll explain the difference between both sorting methods and essentially, the **performance** of both which is measured by space and time complexities.

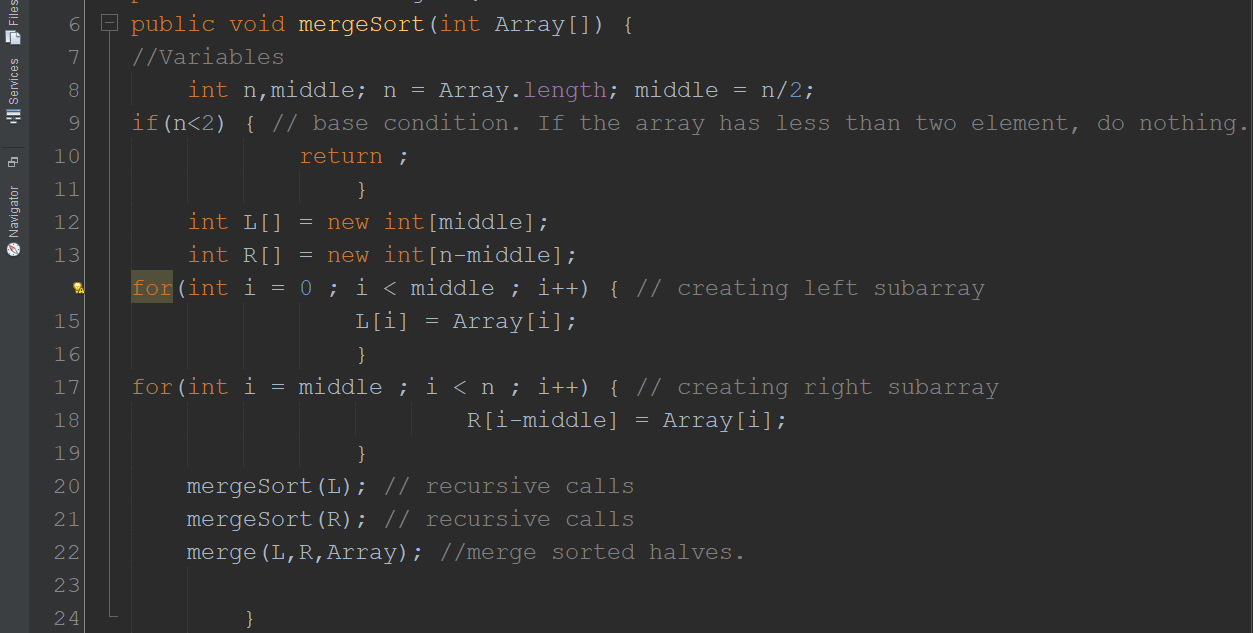
|  |  |  |
| --- | --- | --- |
|  | Selection Sort | Merge Sort |
| Best case | O(n2) | O(nlogn) |
| Worst Case | O(n2) | O(nlogn) |
| Average Case | O(n2) | O(nlogn) |

This table shows the comparison in **time complexities** between both sorting methods. Obviously the merge sort is way better in all the cases as it has o(nlogn) (because we divide the array into two each time) which is faster and the selection sort has two for loops nested so it has o(n2 ).

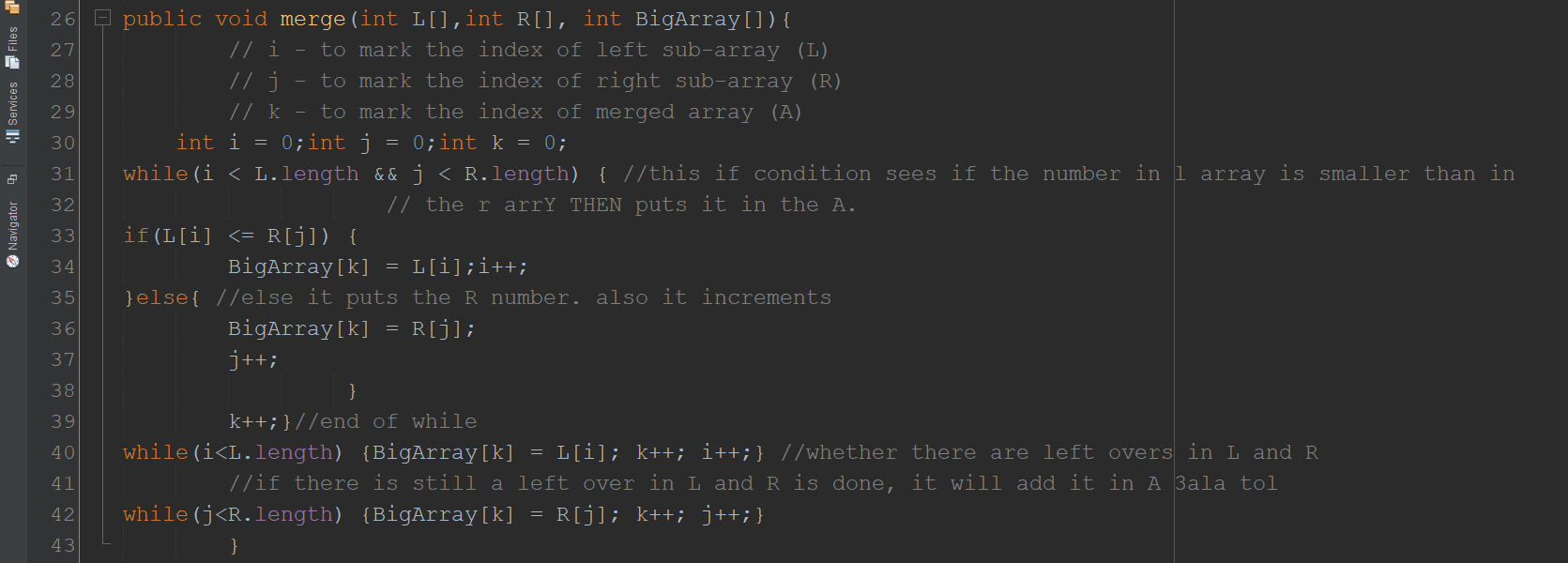
|  |  |  |
| --- | --- | --- |
|  | Selection Sort | Merge Sort |
| Space complexities | O(1) | O(n) |

This table shows the comparison in **space complexities** between both sorting methods. Selection sort is better because it is in-place algorithm as it doesn’t need extra array but merge sort needs an extra one.

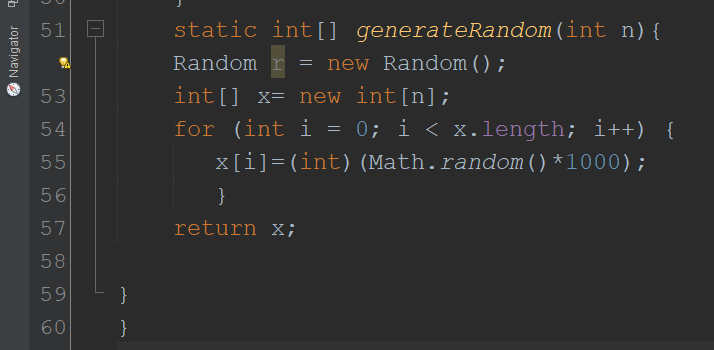
This is the implementation of the sorting algorithms:



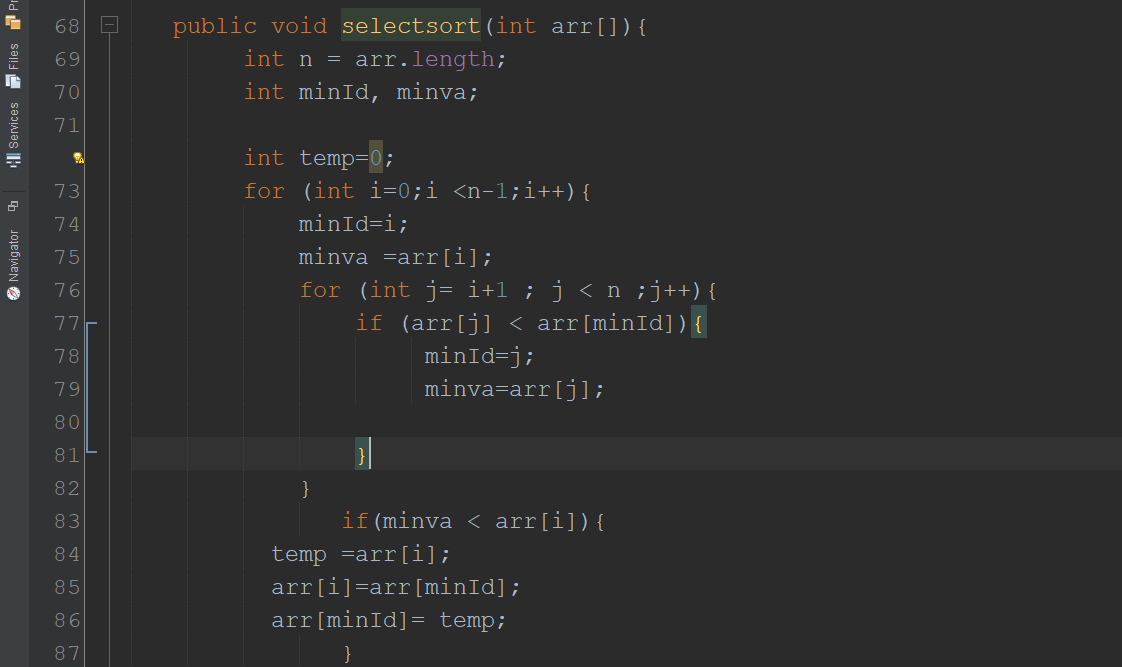
* This method is the one that uses recursion. It first uses an if condition as the base condition for stopping the recursion then there are two for loops that creates the two sub arrays from the array given to the method. They fill them according to which is smaller first. Lastly, it calls it self to sort the left subarray then the second. The merge method merges the already sorted two sub array.



* This is the Merge method which is responsible for merging the two subarrays together by checking which is smaller and putting it first.



* This method generates random integers in an array. This is used to make an array of random integers which I will use to compare the two sorting methods.



* In selection sorting I used nested for loop one of them starts from first index "0" with variable i and anther one start from "i+1" which means second element and start comparing first with all element in the array if there is no less than minva they will be swapped then complete comparing with new one and so on until the loop is sorted

This is implementing the code using three test cases:

**Test case 1: Using an array of 1000 element.**

|  |  |  |
| --- | --- | --- |
|  | Selection Sort | Merge Sort |
| Reversed |  |  |
| Sorted |  |  |
| Unsorted |  |  |

This table proves the above theory, that the merge took more time than select even when the array was reversed and sorted.

**Test case 2: Using an array of 2000 element.**

|  |  |  |
| --- | --- | --- |
|  | Selection Sort | Merge Sort |
| Reversed |  |  |
| Sorted |  |  |
| Unsorted |  |  |

This table also shows the above statements are right as still the merge sort takes less time sorting different arrays.

**Test case 3: Using an array of 3000 element.**

|  |  |  |
| --- | --- | --- |
|  | Merge Sort | Selection Sort |
| Reversed |  |  |
| Sorted |  |  |
| Unsorted |  |  |

Graph Comparison:

* this graph compares between the two sorting algorithms and shows that the merge sort (sorting an unsorted array) takes less time than the selection sort.

Machine specifications: Processor: Intel® core™ i7-8565U CPU @1.80GHz

Installed memory RAM: 16.0 GB.

L1 CACHE: 256KB

L2 CACHE: 1.0MG

L3 CACHE: 8.0MG