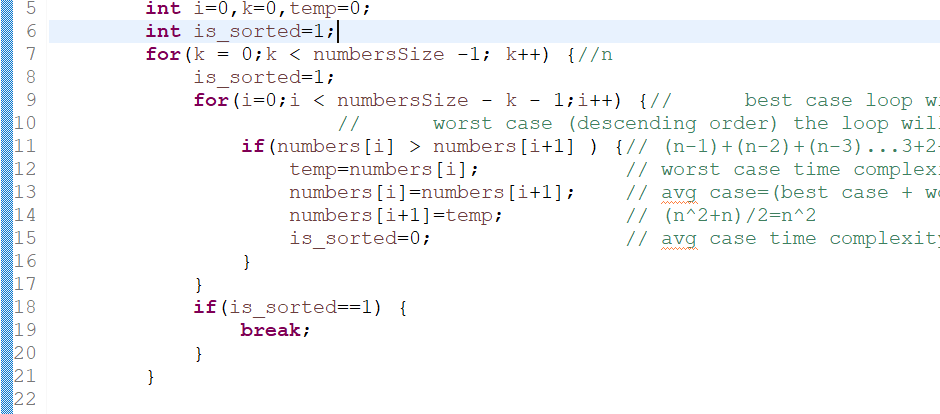
|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Best Case*** | ***Average Case*** | ***Worst Case*** |
| ***Bubble Sort*** | O(*n*) | *O*(*n*2) | *O*(*n*2) |
| ***Selection Sort*** | *O*(*n*2) | *O*(*n*2) | *O*(*n*2) |
| ***Insertion Sort*** | O(*n*) | *O*(*n*2) | *O*(*n*2) |
| ***Merge Sort*** | O (*n* log(*n*)) | O (*n* log(*n*)) | O (*n* log(*n*)) |

***Explanation:***

Bubble Sort

***Experiment Description:***

An experiment has been conducted by taking an array of input size 10,000, 20,000, 30,000 till 10,00,000 with integers in descending, ascending order and sorting them using Bubble Sort. The time taken for execution has been recorded and the values has been plotted on a graph, with input size on the x-axis and time taken in Nano-seconds in y-axis.



***Worst Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **14** | **12** | **1** | **0** |

* For the worst case the time complexity is to *O*(*n*2).
* The inner For loop at line number 9 in the above picture runs n-1 times for the 1st iteration.
* For second iteration the inner loop runs for n-2 and it continues as an arithmetic progression with difference of -1 till 1.
* Thus, the summation of n-1 + n-2 + ….. 2 + 1 = n\*(n-1) / 2
* The bigger exponent of the above equation is *n*2 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of bubble sort in worst case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the Bubble sort grows at the rate of K2 in the worst case.

***Average Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **14** | **13** |

* The time complexity of bubble sort in average case is *O*(*n*2).
* In average case the list is partially sorted, but still the comparisons have to be done for all the elements till the end.
* The Inner For loop iterates in the arithmetic progression with sequence n-1 + n-2 + .. 2 + 1 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of bubble sort in average case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the Bubble sort grows at the rate of K2 in the average case.

***Best Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |

* The time complexity of bubble sort in best case is O(*n*).
* Best case implies the array is already sorted.
* The is\_sorted flag in line 8 avoids entering into the inner for loop at line number 9 which makes the array search and swap operations only search the N number of items in the array. Hence the complexity is O(*n*).

***Growth Order Function:***

The Time complexity of bubble sort in best case is O(*n*)..

The complexity for O(2N) is 2*n*

The complexity for O(KN) is K*n*

By Dividing the above two coefficients,

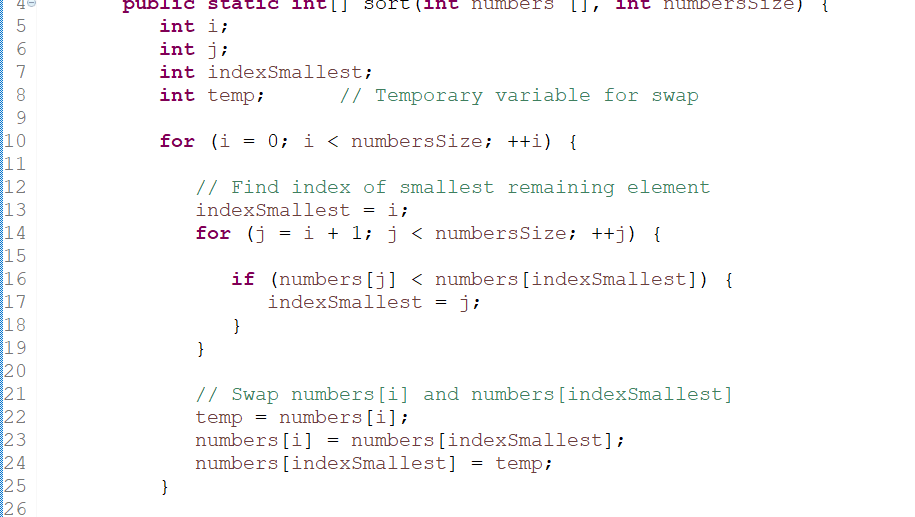
Order of Growth Function O(KN) / O(2N) = K

Hence the Bubble sort grows at the rate of Kin the best case.

Selection Sort

***Experiment Description:***

An experiment has been conducted by taking an array of input size 10,000, 20,000, 30,000 till 10,00,000 with integers in descending, ascending order and sorting them using Selection Sort. The time taken for execution has been recorded and the values has been plotted on a graph, with input size on the x-axis and time taken in Nano-seconds in y-axis.



***Worst Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **14** | **12** | **1** | **0** |

* For the worst case the time complexity is to *O*(*n*2).
* The inner For loop at line number 14 in the above picture runs n-1 times for the 1st iteration.
* For second iteration the inner loop runs for n-2 and it continues as an arithmetic progression with difference of -1 till 1.
* Thus, the summation of n-1 + n-2 + ….. 2 + 1 = n\*(n-1) / 2
* The bigger exponent of the above equation is *n*2 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of Selection sort in worst case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the Selection sort grows at the rate of K2 in the worst case.

***Average Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **14** | **13** |

* The time complexity of selection sort in average case is *O*(*n*2).
* In average case the list is partially sorted, but still the comparisons have to be done for all the elements till the end.
* The Inner For loop iterates in the arithmetic progression with sequence n-1 + n-2 + .. 2 + 1 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of selection sort in average case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the selection sort grows at the rate of K2 in the average case.

***Best Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |

* The time complexity of selection sort in best case is *O*(*n*2).
* Best case implies the array is already sorted.
* Though the array is sorted, the elements are still compared and the inner for loop is executed which leads to running time of n-1 + n-2 … + 2 + 1, which leads to the time complexity of *O*(*n*2).

***Growth Order Function:***

The Time complexity of selection sort in best case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

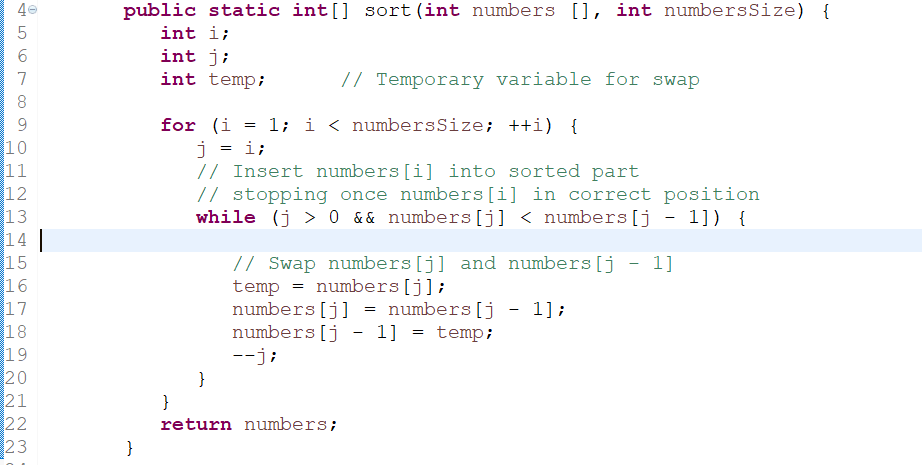
Order of Growth Function O(KN) / O(2N) = K2

Hence the selection sort grows at the rate of K2 in the best case.

Insertion Sort

***Experiment Description:***

An experiment has been conducted by taking an array of input size 10,000, 20,000, 30,000 till 1,00,000 with integers in descending, ascending order and sorting them using Insertion Sort. The time taken for execution has been recorded and the values has been plotted on a graph, with input size on the x-axis and time taken in Nano-seconds in y-axis.



***Worst Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **14** | **12** | **1** | **0** |

* For the worst case the time complexity is to *O*(*n*2).
* The inner For loop at line number 13 in the above picture runs 1 time for the 1st iteration.
* For second iteration the inner loop runs for 2 times and it continues as an arithmetic progression with difference of +1 till n-1.
* Thus, the summation of 1 + 2 + … n-1 = n\*(n-1) / 2
* The bigger exponent of the above equation is *n*2 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of Insertion sort in worst case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the Insertion sort grows at the rate of K2 in the worst case.

***Average Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **14** | **13** |

* The time complexity of Insertion sort in average case is *O*(*n*2).
* In average case the list is partially sorted, but still the comparisons have to be done for all the elements till the end.
* The Inner For loop iterates in the arithmetic progression with sequence 1 + 2 … n-1 and hence the time complexity is *O*(*n*2).

***Growth Order Function:***

The Time complexity of insertion sort in average case is *O*(*n*2).

The complexity for O(2N) is 4*n*2

The complexity for O(KN) is K2 *n*2

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(2N) = K2

Hence the insertion sort grows at the rate of K2 in the average case.

***Best Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |

* The time complexity of insertion sort in best case is *O*(*n*).
* Best case implies the array is already sorted.
* Though the array is sorted, the elements are still compared and the inner for loop is executed which leads to running time of 1 + 2 … + n-1, which leads to the time complexity of *O*(*n*).

***Growth Order Function:***

The Time complexity of insertion sort in best case is *O*(*n*).

The complexity for O(2N) is 4*n*

The complexity for O(KN) is K*n*

By Dividing the above two coefficients,

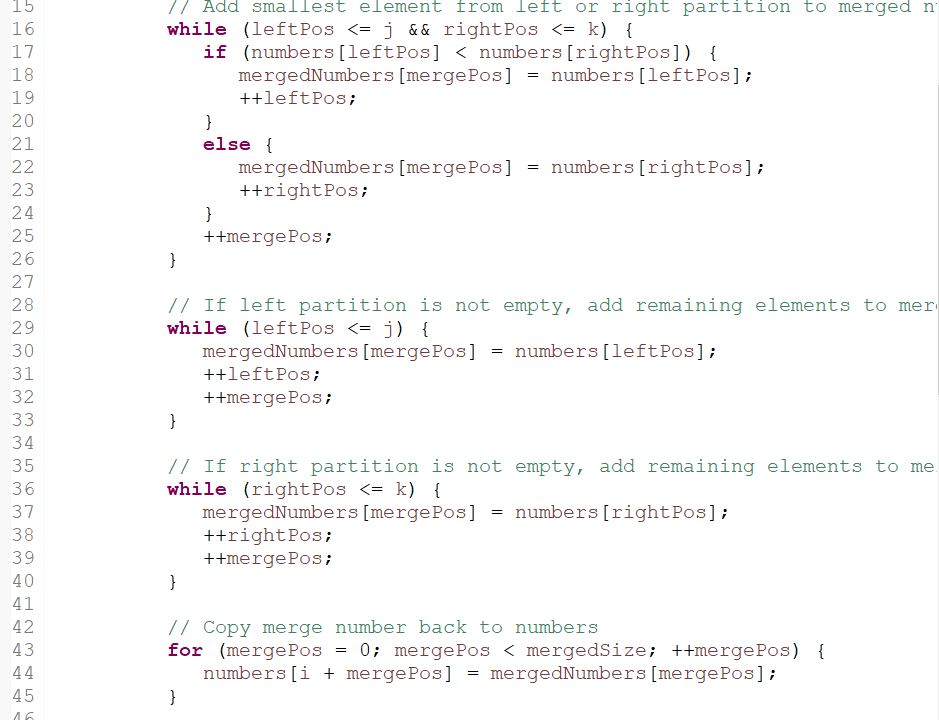
Order of Growth Function O(KN) / O(2N) = K

Hence the insertion sort grows at the rate of Kin the best case.

Merge Sort

***Experiment Description:***

An experiment has been conducted by taking an array of input size 10,000, 20,000, 30,000 till 1,00,000 with integers in descending, ascending order and sorting them using Merge Sort. The time taken for execution has been recorded and the values has been plotted on a graph, with input size on the x-axis and time taken in Nano-seconds in y-axis.



***Worst Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **14** | **12** | **1** | **0** |

* For the worst case the time complexity is to O (*n* log(*n*)).
* Merge-sort is a divide and conquer algorithm and is O (log(*n*)) because the input is repeatedly halved and with n items and hence O (*n* log(*n*)).

***Growth Order Function:***

The Time complexity of Insertion sort in worst case is O (*n* log(*n*)).

The complexity for O(2N) is O (2*n* log(2*n*))

The complexity for O(KN) is O (K*n* log (K*n*))

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(N) = K [Log K+ 1]

Hence the Insertion sort grows at the rate of K [Log K+ 1]in the worst case.

***Average Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **14** | **13** |

* The time complexity of Insertion sort in average case is O (*n* log(*n*)).

***Growth Order Function:***

The Time complexity of Insertion sort in average case is O (*n* log(*n*)).

The complexity for O(2N) is O (2*n* log(2*n*))

The complexity for O(KN) is O (K*n* log (K*n*))

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(N) = K [Log K+ 1]

Hence the Insertion sort grows at the rate of K [Log K+ 1]in the average case.

***Best Case:***

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | **2** | **3** | **4** |

* The time complexity of insertion sort in best case is O (*n* log(*n*)).
* Best case implies the array is already sorted.

***Growth Order Function:***

The Time complexity of Insertion sort in best case is O (*n* log(*n*)).

The complexity for O(2N) is O (2*n* log(2*n*))

The complexity for O(KN) is O (K*n* log (K*n*))

By Dividing the above two coefficients,

Order of Growth Function O(KN) / O(N) = K [Log K+ 1]

Hence the Insertion sort grows at the rate of K [Log K+ 1]in the best case.