Sorting Algorithms Analysis

Following is a tabular representation of the execution time taken by Insertion Sort, Selection Sort and Quick Sort in Various Scenarios(all times are in nanoseconds):

Scenario 1:

**Data Set is Randomly Organized**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 5559 | 2994 | 7270 | Randomly organized Numbers |
| 100 | 564931 | 540983 | 147968 |
| 1000 | 33577310 | 34526704 | 2159656 |
| 10000 | 125988364 | 131496984 | 9821946 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Selection sort starts to work poorer as the dataset gets bigger.
3. For numbers greater than 10 Quick Sort is the best option.

Scenario 2:

**Dataset is Already Sorted**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 4276 | 2566 | 4704 | Already Sorted Dataset |
| 100 | 645331 | 507198 | 111190 |
| 1000 | 21824931 | 20912316 | 1241481 |
| 10000 | 153258622 | 113907973 | 6123588 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Insertion sort work poorer than other sorting algorithms for any size of input data.
3. For numbers greater than 10 Quick Sort is the best option.

Scenario 3:

**Numbers in Descending Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 3849 | 2566 | 5132 | Numbers in Descending Order |
| 100 | 726585 | 570919 | 118033 |
| 1000 | 23343533 | 23049307 | 1499785 |
| 10000 | 134537609 | 122715950 | 7141407 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Insertion sort work poorer than other sorting algorithms for any size of input data.
3. For numbers greater than 10 Quick Sort is the best option.

Scenario 4:

**Numbers are in Nearly Sorted Order**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 20527 | 7270 | 18389 | Nearly Sorted Numbers |
| 100 | 498217 | 397718 | 643620 |
| 1000 | 22223039 | 18430604 | 1034067 |
| 10000 | 112589321 | 111727171 | 6826640 |

**Conclusions:**

1. Quick sort works poor in case of 10 numbers in nearly sorted order.
2. For numbers greater than 10 Quick Sort is the best option.
3. As size of array goes above 10000, Selection sort performs worse than Insertion Sort.

Scenario 5:

**Partially Sorted Array**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 17106 | 11119 | 27370 | Partially Sorted Numbers |
| 100 | 533285 | 490091 | 182608 |
| 1000 | 16563039 | 16909438 | 4062712 |
| 10000 | 87509553 | 69339390 | 10838890 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Selection Sort works poorly in case of 1000 numbers.
3. Insertion sort work poorer than other sorting algorithms for any size of input data.
4. For numbers greater than 10 Quick Sort is the best option.

Scenario 6:

**Pivot of Quick Sort at Max Element**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 16679 | 11119 | 29508 | Pivot at Max Element |
| 100 | 894224 | 373770 | 161653 |
| 1000 | 19995817 | 18873654 | 1938128 |
| 10000 | 78152484 | 75452704 | 9340391 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Selection Sort works poorly in case of 1000 numbers.
3. For numbers greater than 10 Quick Sort is the best option.

Scenario 7:

**Pivot of Quick Sort at Min Element**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 15396 | 11119 | 29936 | Pivot at Min Element |
| 100 | 536279 | 515751 | 165502 |
| 1000 | 21097027 | 19584842 | 2110473 |
| 10000 | 78140082 | 80868942 | 11382439 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. Selection Sort works poorly in case of 1000 numbers.
3. For numbers greater than 10 Quick Sort is the best option.

Scenario 8:

**Range of Number is varied**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 14540 | 11119 | 28653 | Number Range is Big |
| 100 | 403278 | 510619 | 98360 |
| 1000 | 15207376 | 15753062 | 1712754 |
| 10000 | 76695039 | 64033914 | 8550087 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. For numbers greater than 10 Quick Sort is the best option.
3. For 1000 numbers in the array, selection sort performs poorly than Insertion and Quick Sort algorithms.

Scenario 9:

**All Elements in the array are same**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number of Elements | Insertion Sort Time | Selection Sort Time | Quick Sort Time | Data Behavior |
| 10 | 18817 | 11119 | 25231 | All Elements Are The Same |
| 100 | 481111 | 455879 | 138988 |
| 1000 | 15703454 | 16073375 | 1128578 |
| 10000 | 108932879 | 66965482 | 5929422 |

**Conclusions:**

1. Quick sort works poor in case of smaller numbers.
2. For numbers greater than 10 Quick Sort is the best option.
3. For 1000 numbers in the array, selection sort performs poorly than Insertion and Quick Sort algorithms.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Number of Elements** | **Insertion Sort Time** | **Selection Sort Time** | **Quick Sort Time** | **Data Behavior** |
| 10 | 5559 | 2994 | 7270 | **Randomly organized Numbers** |
| 100 | 564931 | 540983 | 147968 |
| 1000 | 33577310 | 34526704 | 2159656 |
| 10000 | 125988364 | 131496984 | 9821946 |
| 10 | 4276 | 2566 | 4704 | **Already Sorted Dataset** |
| 100 | 645331 | 507198 | 111190 |
| 1000 | 21824931 | 20912316 | 1241481 |
| 10000 | 153258622 | 113907973 | 6123588 |
| 10 | 3849 | 2566 | 5132 | **Numbers in Descending Order** |
| 100 | 726585 | 570919 | 118033 |
| 1000 | 23343533 | 23049307 | 1499785 |
| 10000 | 134537609 | 122715950 | 7141407 |
| 10 | 20527 | 7270 | 18389 | **Nearly Sorted Numbers** |
| 100 | 498217 | 397718 | 643620 |
| 1000 | 22223039 | 18430604 | 1034067 |
| 10000 | 112589321 | 111727171 | 6826640 |
| 10 | 17106 | 11119 | 27370 | **Partially Sorted Numbers** |
| 100 | 533285 | 490091 | 182608 |
| 1000 | 16563039 | 16909438 | 4062712 |
| 10000 | 87509553 | 69339390 | 10838890 |
| 10 | 16679 | 11119 | 29508 | **Pivot at Max Element** |
| 100 | 894224 | 373770 | 161653 |
| 1000 | 19995817 | 18873654 | 1938128 |
| 10000 | 78152484 | 75452704 | 9340391 |
| 10 | 15396 | 11119 | 29936 | **Pivot at Min Element** |
| 100 | 536279 | 515751 | 165502 |
| 1000 | 21097027 | 19584842 | 2110473 |
| 10000 | 78140082 | 80868942 | 11382439 |
| 10 | 14540 | 11119 | 28653 | **Number Range is Big** |
| 100 | 403278 | 510619 | 98360 |
| 1000 | 15207376 | 15753062 | 1712754 |
| 10000 | 76695039 | 64033914 | 8550087 |
| 10 | 18817 | 11119 | 25231 | **All Elements Are Same** |
| 100 | 481111 | 455879 | 138988 |
| 1000 | 15703454 | 16073375 | 1128578 |
| 10000 | 108932879 | 66965482 | 5929422 |

**Overall Execution Summary:**

**Analysis of Quick Sort Algorithm:**

**Conclusions:**

1. When the array is nearly sorted, quick sort algorithm works worst when there are 100 elements in the list.
2. When the pivot is chosen at the minimum value of the array, the time taken for sorting 10 elements is the maximum.
3. For 100 elements in an array, Quick sort algorithm takes the maximum time for the Nearly Sorted array.
4. The worst time taken for 1000 elements is when the array is Partially Sorted.

**Analysis of Insertion Sort Algorithm:**

1. This is an algorithm where we perform swaps of elements
2. This algorithm works better than Quick Sort in case the input array has a smaller length
3. The worst performance of this algorithm is noted when the array is already sorted or nearly sorted
4. This algorithm works better than Selection sort when the input data is randomly stored
5. The worst case time complexity of this algorithm is O(n2)
6. The space complexity of this algorithm is O(n)

**Analysis of Selection Sort Algorithm:**

1. This is an algorithm where we perform swaps of elements
2. This algorithm works better than Quick Sort in case the input array has a smaller length
3. The worst performance of this algorithm is noted when the array is partially sorted or the number range is huge or data is randomly organized
4. This algorithm works better than Insertion sort in most of the cases
5. The worst case time complexity of this algorithm is O(n2)
6. The space complexity of this algorithm is O(n)

**Time Complexity Analysis of the three Algorithms:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Scenario** | **Insertion Sort** | **Selection Sort** | **Quick Sort** |
| **Worst Case** | **O(n2)** | **O(n2)** | **O(n2)** |
| **Average Case** | **O(n2)** | **O(n2)** | **O(nlogn)** |
| **Best Case** | **O(n)** | **O(n2)** | **O(nlogn)** |