Sorting and Searching Algorithms

Michael Karimizadeh & Oliver Mo

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial State  (Bubble) | 2 | 9 | 4 | 8 | 0 | # of comparisons | accesses |
| i = 0 | 2 | 4 | 8 | 0 | 9 | 1111 4 | 111 |
| i = 1 | 2 | 4 | 0 | 8 | 9 | 1111 4 | 1 |
| i = 2 | 2 | 0 | 4 | 8 | 9 | 1111 4 | 1 |
| i = 3 | 0 | 2 | 4 | 8 | 9 | 1111 4 | 1 |
| i = 4 | 0 | 2 | 4 | 8 | 9 | 1111 4 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial State  (Insertion) | 2 | 9 | 4 | 8 | 0 | # of comparisons | accesses |
| i = 1 | 2 | 9 | 4 | 8 | 0 | 1 1 | 0 |
| i = 2 | 2 | 4 | 9 | 8 | 0 | 11 2 | 1 |
| i = 3 | 2 | 4 | 8 | 9 | 0 | 11 2 | 1 |
| i = 4 | 0 | 2 | 4 | 8 | 9 | 11111 5 | 1111 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial State  (Selection) | 2 | 9 | 4 | 8 | 0 | # of comparisons | accesses |
| i = 0 | 0 | 9 | 4 | 8 | 2 | 1111 4 | 1 |
| i = 1 | 0 | 2 | 4 | 8 | 9 | 111 3 | 1 |
| i = 2 | 0 | 2 | 4 | 8 | 9 | 11 2 | 0 |
| i = 3 | 0 | 2 | 4 | 8 | 9 | 1 1 | 0 |

|  |  |
| --- | --- |
| Bubble Sort Advantages | Bubble Sort Disadvantages |
| * Simple to write * Easy to understand * Good for sorting small quantities * Stable * Doesn’t create additional temporary storages | * Time inefficient - runs in *O*(*n*2) * Bad at sorting large amounts of data * Suitable for academic purposes but not real world applications |

|  |  |
| --- | --- |
| Insertion Sort Advantages | Insertion Sort Disadvantages |
| * Simple implementation * Efficient for small data sets * Minimal space requirement because it uses a in-place sorting algorithm * Easily understood | * Not as efficient as other sorting algorithms * Usually only useful when sorting small amounts of data |

|  |  |
| --- | --- |
| Selection Sort Advantages | Selection Sort Disadvantages |
| * Performs well on a small list * Performance can be faster depending on initial sorting of data * No additional temporary storage needed because it is an in-place sorting algorithm | * Not the most efficient when dealing with large amounts of data * Requires *n*2 number of steps for sorting *n* elements |

**Bubble Sort Pseudocode**

If the first in the array is greater than the second, switch them

If the second is greater than the third, switch them

If the third is greater than the fourth, switch them

If the fourth is greater than the fifth, switch them

If the fifth is greater than the sixth, switch them

Repeat until second last has been compared with last one

Check array

Display Array

If any number is less than the one previous to it, repeat whole code

If no numbers are less than the one previous to it stop checking

**Insertion Sort Pseudocode**

If second in array is less than first in array, switch them

Display Array

If third in array is less than second, switch them

If second in array is less than first, switch them

Display Array

If fourth in array is less than third, switch them

If third in array is less than second, switch them

If second in array is less than first, switch them

Display Array

Repeat this pattern until end of array has been checked

Display Array

**Selection Sort Pseudocode**

If the second in the array is less than the first, change lowest index

If third in array is less than number at lowest index, change lowest index

If fourth in array is less than number at lowest index, change lowest index

If fifth in array is less than number at lowest index, change lowest index

Repeat this pattern until last in array has been checked

Switch the first number with the number at lowest index

Display Array

If this code has repeated as many times as there are numbers in the array, stop checking

If code hasn’t repeated as many times as necessary, repeat code

**Execution time & Worst Case Performance**

O(n2)

The definition of this type of time is when the number of operations is proportional to the size of the task squared. This type of time is used when there are loops within loops being used.

Bubble sort uses O(n2) due to the basic code including nested for loop within a do while loop.

Selection sort uses O(n2) because of its code being comprised of nested for loops within for loops.

Insertion sort also uses O(n2) because of the nested while loops within for loops in order to function.

**Ordered Array**

7 12 14 15 16 19 21 22 24 32 32 32 37 44 45 45 47 56 58 64 65 65 65 69 69 79 82 85 87 98

Best Case Scenario (Nanoseconds)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Insertion | Selection | Bubble |
| 5 | 3640 | 4368 | 3276 |
| 10 | 4004 | 6188 | 3641 |
| 15 | 4369 | 8737 | 4005 |
| 20 | 4733 | 12013 | 4369 |
| 25 | 5096 | 17110 | 4369 |
| 30 | 5461 | 22207 | 4733 |

**Random Array**

85 45 69 32 14 24 22 15 7 98 65 65 47 12 32 44 58 69 56 32 21 64 79 82 19 37 45 65 87 16

Random Scenario (Nanoseconds)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Insertion | Selection | Bubble |
| 5 | 4368 | 4369 | 5461 |
| 10 | 5824 | 6189 | 9101 |
| 15 | 8009 | 8919 | 15290 |
| 20 | 9829 | 12924 | 20023 |
| 25 | 12377 | 17474 | 33128 |
| 30 | 16017 | 22571 | 47326 |

**Backwards Array**

98 87 85 82 79 69 69 65 65 65 64 58 56 47 45 45 44 37 32 32 32 24 22 21 19 16 15 14 12 7

Worse Case Scenario (Nanoseconds)

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Insertion | Selection | Bubble |
| 5 | 4369 | 4369 | 5097 |
| 10 | 6188 | 6188 | 9465 |
| 15 | 9829 | 9101 | 18566 |
| 20 | 14917 | 12742 | 30216 |
| 25 | 20023 | 17474 | 45870 |
| 30 | 26939 | 22935 | 65892 |