**Administrative**

**Today’s session**

Homework 2 key

Linear search

Binary search

Insertion sort

Selection sort

Bubble sort

**Session Topics**

**Homework 2 key**

● The Homework 2 key is available on Blackboard.

**Linear search**

● A **linear search** searches an array for a value by starting at the *beginning* of the array.

● If the value is not found at the current element, the search moves to the next element in the array.

● The search continues until the value is found or the end of the array is reached.

● A linear search may be done on an **unsorted** array, an array with data in any order.

● See **Array Searching** sample application on Blackboard.

**Binary search**

● A **binary search** searches an array for a value by starting at the *middle* of the array.

● If the value is not found, it cuts the array in half, discarding one half and searching the other half.

● It continues to search until the value is found or there is no array to cut in half.

● A binary search may only be done on a **sorted** array.

● A binary search may use iteration or recursion to perform the search.

● See **Array Searching** sample application on Blackboard.

● Here is a comparison of linear and binary searches for an array of N elements:

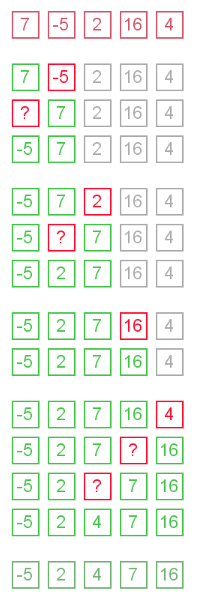
|  |  |  |
| --- | --- | --- |
|  | Linear search | Binary search |
| Requirement | None, data may be in any order. | Data must be sorted. |
| Average search time | N / 2 | log2 N |
| Worst-case search time | N | log2 N |

|  | **Average time** | | **Worst-case time** | |
| --- | --- | --- | --- | --- |
| **N** | **Linear search** | **Binary search** | **Linear search** | **Binary search** |
| **5** | 2.5 | 1.3 | 5.0 | 2.3 |
| **10** | 5.0 | 2.3 | 10.0 | 3.3 |
| **20** | 10.0 | 3.3 | 20.0 | 4.3 |
| **30** | 15.0 | 3.9 | 30.0 | 4.9 |
| **40** | 20.0 | 4.3 | 40.0 | 5.3 |
| **50** | 25.0 | 4.6 | 50.0 | 5.6 |
| **75** | 37.5 | 5.2 | 75.0 | 6.2 |
| **100** | 50.0 | 5.6 | 100.0 | 6.6 |
| **150** | 75.0 | 6.2 | 150.0 | 7.2 |
| **200** | 100.0 | 6.6 | 200.0 | 7.6 |
| **300** | 150.0 | 7.2 | 300.0 | 8.2 |
| **400** | 200.0 | 7.6 | 400.0 | 8.6 |
| **500** | 250.0 | 8.0 | 500.0 | 9.0 |
| **750** | 375.0 | 8.6 | 750.0 | 9.6 |
| **1,000** | 500.0 | 9.0 | 1,000.0 | 10.0 |
| **1,500** | 750.0 | 9.6 | 1,500.0 | 10.6 |
| **2,000** | 1,000.0 | 10.0 | 2,000.0 | 11.0 |

**Insertion sort**

● The **insertion sort** splits the array into sorted and unsorted portions.

● In the unsorted part of the array, it takes the first element and finds the spot where it fits in the sorted part of the array.



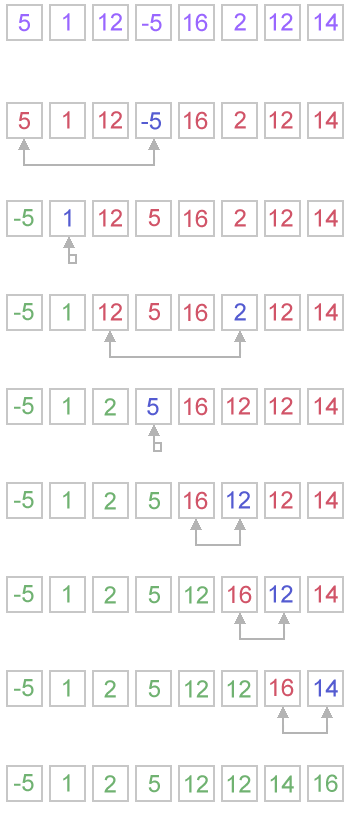
● A visualization of the insertion sort is available at [www.cs.usfca.edu/~galles/visualization/ComparisonSort.html](http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html).

**Selection sort**

● The **selection sort** splits the array into sorted and unsorted portions.

● In the unsorted part of the array, it compares the first element against all the other elements.

● In each comparison, if the array element values are out of order, they are swapped.



● A visualization of the selection sort is available at [www.cs.usfca.edu/~galles/visualization/ComparisonSort.html](http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html).

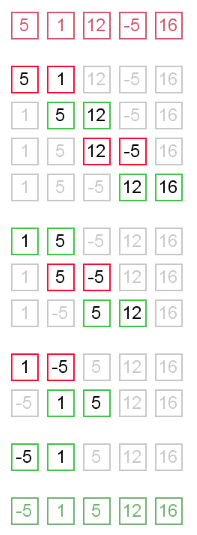
**Bubble sort**

● The **bubble sort** splits the array into sorted and unsorted portions.

● In the unsorted part of the array, it bubbles the lowest value to the top (beginning) or the highest value to the bottom (end).

● In each comparison, if the array element values are out of order, they are swapped.

● This example "bubbles" the highest value to the bottom (end) of the array:



● A visualization of the bubble sort is available at [www.cs.usfca.edu/~galles/visualization/ComparisonSort.html](http://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html).

● See **Array Sorting** sample application on Blackboard.

● Here is a comparison of the three sorting methods for an array of N elements:

|  |  |  |  |
| --- | --- | --- | --- |
| Sort method | Best case | Average case | Worst case |
| Insertion | N | N2 | N2 |
| Selection | N2 | N2 | N2 |
| Bubble | N | N2 | N2 |

● The **best (fastest) cases** for these sort methods is when the data is already sorted.

● The **worst (slowest) cases** for these sort methods is when the data is reverse sorted.