

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

COMP2521 25T1

Sorting Algorithms (II)

Elementary Sorting Algorithms

Kevin Luxa

`cs2521@cse.unsw.edu.au`

selection sort

bubble sort

insertion sort

Selection Sort

Example

Implementation

Analysis

Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Method:

- Find the smallest element, swap it with the first element
- Find the second-smallest element, swap it with the second element
- ...
- Find the second-largest element, swap it with the second-last element

Each iteration improves the “sortedness” of the array by one element.

Selection Sort

Example

Implementation

Analysis

Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Example

4	1	7	3	8	6	5	2
---	---	---	---	---	---	---	---

Selection Sort

Example

Implementation

Analysis

Properties

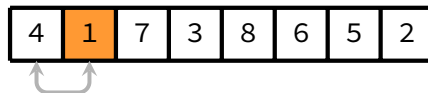
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

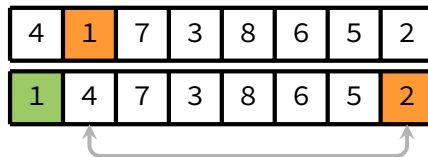
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

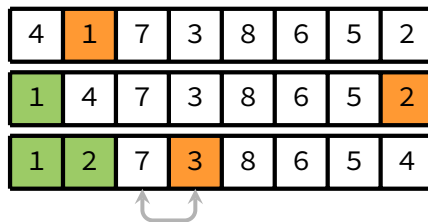
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

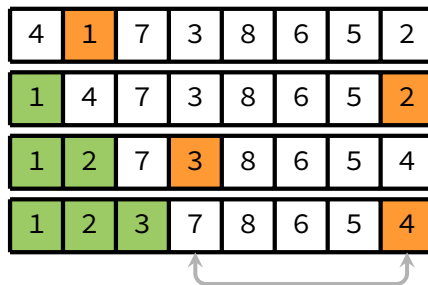
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

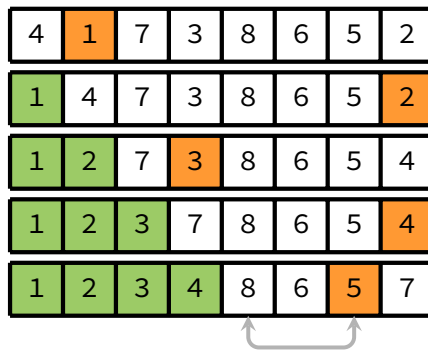
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

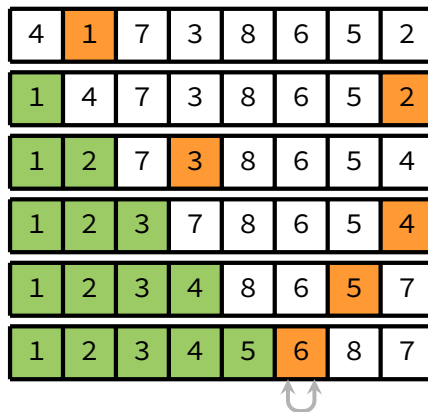
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

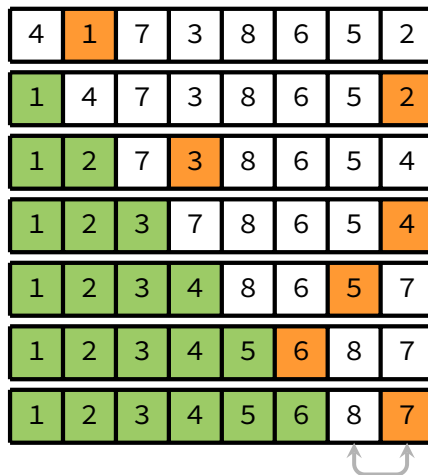
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

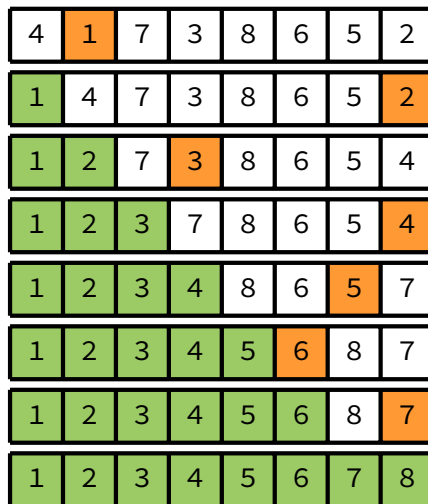
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix



Selection Sort

Example

Implementation

Analysis

Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

```
void selectionSort(Item items[], int lo, int hi) {  
    for (int i = lo; i < hi; i++) {  
        int min = i;  
        for (int j = i + 1; j <= hi; j++) {  
            if (lt(items[j], items[min])) {  
                min = j;  
            }  
        }  
        swap(items, i, min);  
    }  
}
```

Selection Sort

Example

Implementation

Analysis

Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Cost analysis:

- In the first iteration, $n - 1$ comparisons, 1 swap
- In the second iteration, $n - 2$ comparisons, 1 swap
- ...
- In the final iteration, 1 comparison, 1 swap
- $C = (n - 1) + (n - 2) + \dots + 1 = \frac{1}{2}n(n - 1) \Rightarrow O(n^2)$
- $S = n - 1$

Cost is the same, regardless of the sortedness of the original array.

Selection Sort

Example

Implementation

Analysis

Properties

Bubble Sort

Insertion Sort

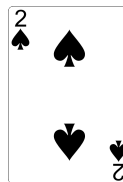
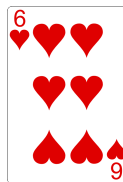
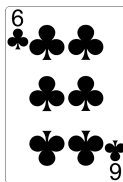
Summary

Sorting Lists

Appendix

Selection sort is unstable

- Due to long-range swaps
- For example, sort these cards by value:



Unstable

Due to long-range swaps

Non-adaptive

Performs same steps, regardless of sortedness of original array

In-place

Sorting is done within original array; does not use temporary arrays

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Method:

- Make multiple passes from left (low) to right
- On each pass, swap any out-of-order adjacent pairs
- Elements “bubble up” until they meet a larger element
- Stop if there are no swaps during a pass
 - This means the array is sorted

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Example

4	3	6	1	2	5
---	---	---	---	---	---

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

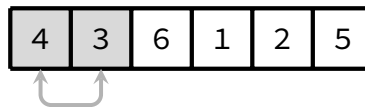
Insertion Sort

Summary

Sorting Lists

Appendix

First pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

First pass

4	3	6	1	2	5
3	4	6	1	2	5

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

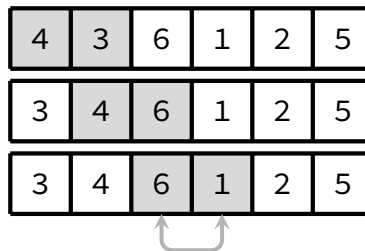
Insertion Sort

Summary

Sorting Lists

Appendix

First pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

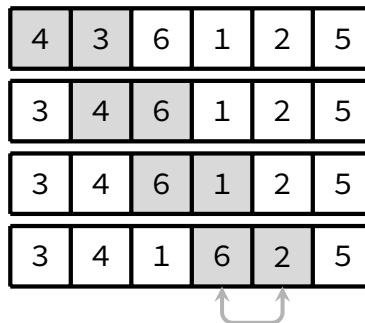
Insertion Sort

Summary

Sorting Lists

Appendix

First pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

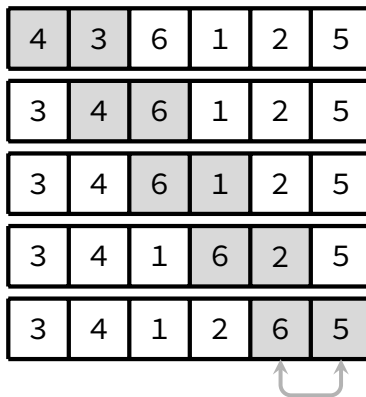
Insertion Sort

Summary

Sorting Lists

Appendix

First pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

First pass

4	3	6	1	2	5
3	4	6	1	2	5
3	4	6	1	2	5
3	4	1	6	2	5
3	4	1	2	6	5
3	4	1	2	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

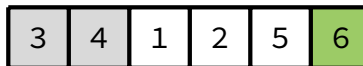
Insertion Sort

Summary

Sorting Lists

Appendix

Second pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

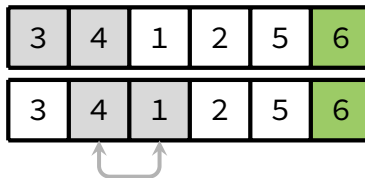
Insertion Sort

Summary

Sorting Lists

Appendix

Second pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

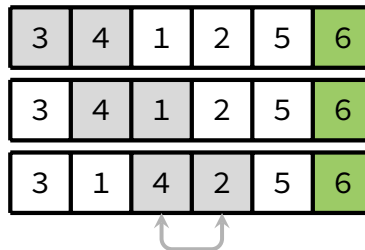
Insertion Sort

Summary

Sorting Lists

Appendix

Second pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Second pass

3	4	1	2	5	6
3	4	1	2	5	6
3	1	4	2	5	6
3	1	2	4	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Second pass

3	4	1	2	5	6
3	4	1	2	5	6
3	1	4	2	5	6
3	1	2	4	5	6
3	1	2	4	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

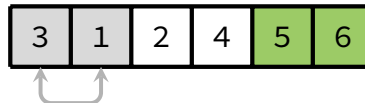
Insertion Sort

Summary

Sorting Lists

Appendix

Third pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

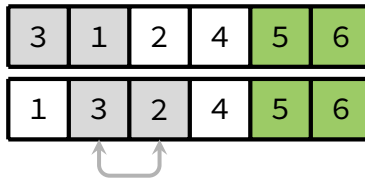
Insertion Sort

Summary

Sorting Lists

Appendix

Third pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Third pass

3	1	2	4	5	6
1	3	2	4	5	6
1	2	3	4	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Third pass

3	1	2	4	5	6
1	3	2	4	5	6
1	2	3	4	5	6
1	2	3	4	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

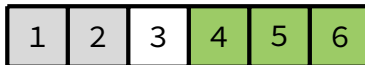
Insertion Sort

Summary

Sorting Lists

Appendix

Fourth pass



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Fourth pass

1	2	3	4	5	6
1	2	3	4	5	6

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

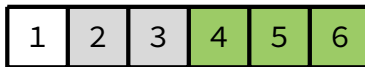
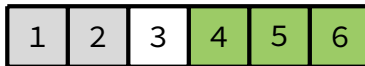
Insertion Sort

Summary

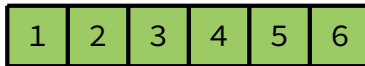
Sorting Lists

Appendix

Fourth pass



No swaps made; stop



Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

```
void bubbleSort(Item items[], int lo, int hi) {  
    for (int i = hi; i > lo; i--) {  
        bool swapped = false;  
        for (int j = lo; j < i; j++) {  
            if (gt(items[j], items[j + 1])) {  
                swap(items, j, j + 1);  
                swapped = true;  
            }  
        }  
        if (!swapped) break;  
    }  
}
```

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Best case: Array is sorted

- Only a single pass required
- $n - 1$ comparisons, no swaps
- Best-case time complexity: $O(n)$

1	2	3	4	5	6
---	---	---	---	---	---

Worst case: Array is reverse-sorted

- $n - 1$ passes required
 - First pass: $n - 1$ comparisons
 - Second pass: $n - 2$ comparisons
 - ...
 - Final pass: 1 comparison
- Total comparisons: $(n - 1) + (n - 2) + \dots + 1 = \frac{1}{2}n(n - 1)$
- Every comparison leads to a swap $\Rightarrow \frac{1}{2}n(n - 1)$ swaps
- Worst-case time complexity: $O(n^2)$

6	5	4	3	2	1
---	---	---	---	---	---

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Average-case time complexity: $O(n^2)$

- It can be proven that for a randomly ordered array, bubble sort needs to perform $\frac{1}{4}n(n-1)$ swaps on average $\Rightarrow O(n^2)$
 - See appendix for details
- Can show empirically by generating random sequences and sorting them

Selection Sort

Bubble Sort

Example

Implementation

Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix

Stable

Comparisons are between adjacent elements only
Elements are only swapped if out of order

Adaptive

Bubble sort is $O(n^2)$ on average, $O(n)$ if input array is sorted

In-place

Sorting is done within original array; does not use temporary arrays

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Method:

- Take first element and treat as sorted array (of length 1)
- Take next element and insert into sorted part of array so that order is preserved
 - This increases the length of the sorted part by one
- Repeat for remaining elements

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

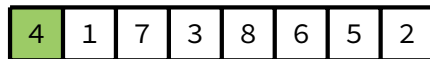
Example

4	1	7	3	8	6	5	2
---	---	---	---	---	---	---	---

Selection Sort

Bubble Sort

Insertion Sort



Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

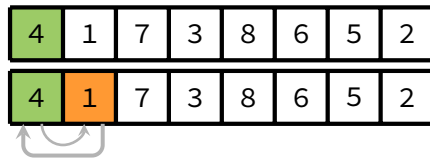
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

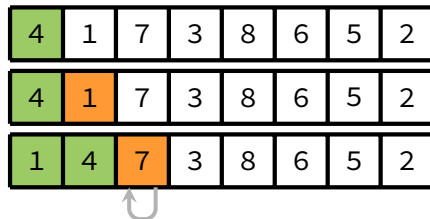
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

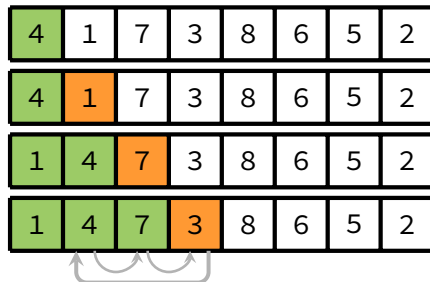
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

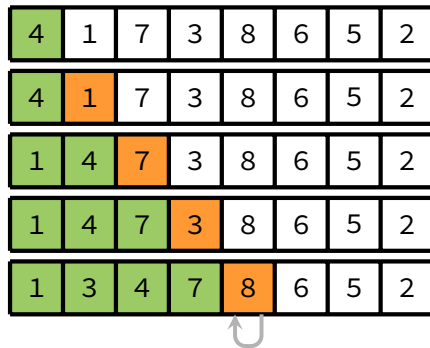
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

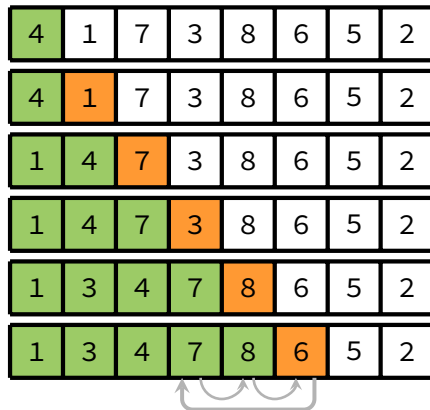
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

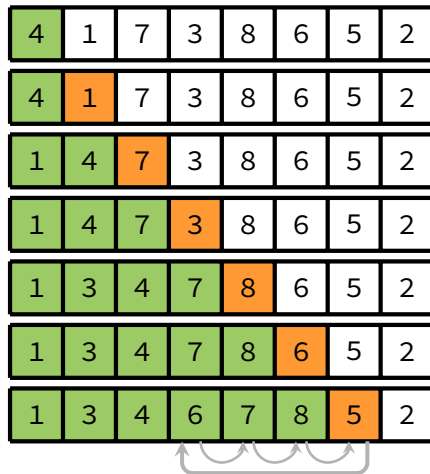
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

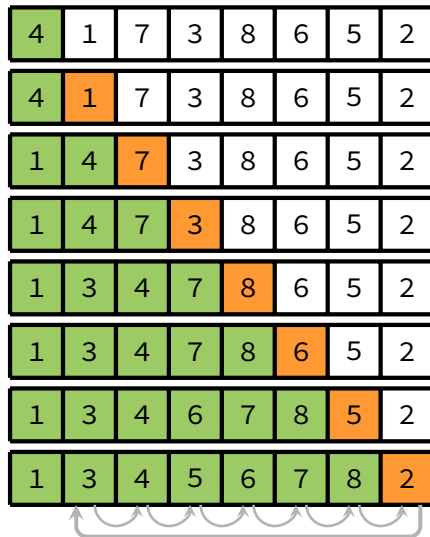
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

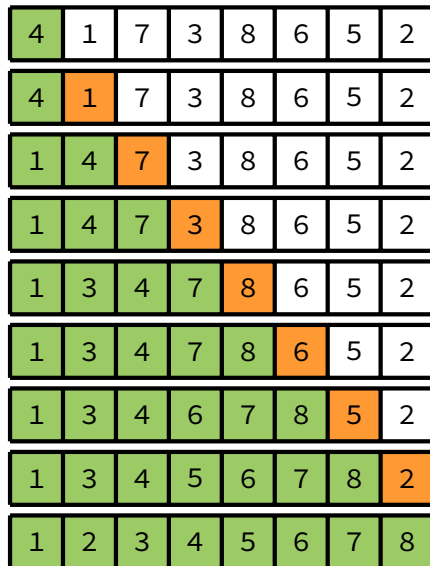
Analysis

Properties

Summary

Sorting Lists

Appendix



Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Best case: Array is sorted

- Inserting each element requires one comparison
- $n - 1$ comparisons
- Best-case time complexity: $O(n)$

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Worst case: Array is reverse-sorted

- Inserting i -th element requires i comparisons
 - Inserting index 1 element requires 1 comparison
 - Inserting index 2 element requires 2 comparisons
 - ...
- Total comparisons: $1 + 2 + \dots + (n - 1) = \frac{1}{2}n(n - 1)$
- Worst-case time complexity: $O(n^2)$

8	7	6	5	4	3	2	1
---	---	---	---	---	---	---	---

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Average-case time complexity: $O(n^2)$

- Same reason as for bubble sort
- Can show empirically by generating random sequences and sorting them

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation

Analysis

Properties

Summary

Sorting Lists

Appendix

Stable

Elements are always inserted to the right of any equal elements

Adaptive

Insertion sort is $O(n^2)$ on average, $O(n)$ if input array is sorted

In-place

Sorting is done within original array; does not use temporary arrays

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

	Time complexity			Properties	
	Best	Average	Worst	Stable	Adaptive
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	No	No
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	Yes	Yes

Selection sort:

- Let L = original list, S = sorted list (initially empty)
- Repeat the following until L is empty:
 - Find the node V containing the largest value in L , and unlink it
 - Insert V at the front of S

Bubble sort:

- Traverse the list, comparing adjacent values
 - If value in current node is greater than value in next node, swap values
- Repeat the above until no swaps required in one traversal

Insertion sort:

- Let L = original list, S = sorted list (initially empty)
- For each node in L :
 - Insert the node into S in order

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

<https://forms.office.com/r/2BW7BasQ77>



Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
case

Insertion sort
walkthrough

Appendix

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

New concept: **inversion**

An inversion is a pair of elements from a sequence where the left element is greater than the right element.

For example, consider the following array:

4	2	1	5	3
---	---	---	---	---

The array contains 5 inversions:

$(4, 2)$, $(4, 1)$, $(4, 3)$, $(2, 1)$, $(5, 3)$

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

Observation:

- In bubble sort, every swap reduces the number of inversions by 1

The goal of the proof: Show that the average number of inversions in a randomly sorted array is $O(n^2)$.

- This implies the number of swaps required by bubble sort is $O(n^2)$...
- Which implies that the average-case time complexity of bubble sort is $O(n^2)$ or slower
 - (but we know that it can't be slower than $O(n^2)$ since the worst-case time complexity of bubble sort is $O(n^2)$)

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

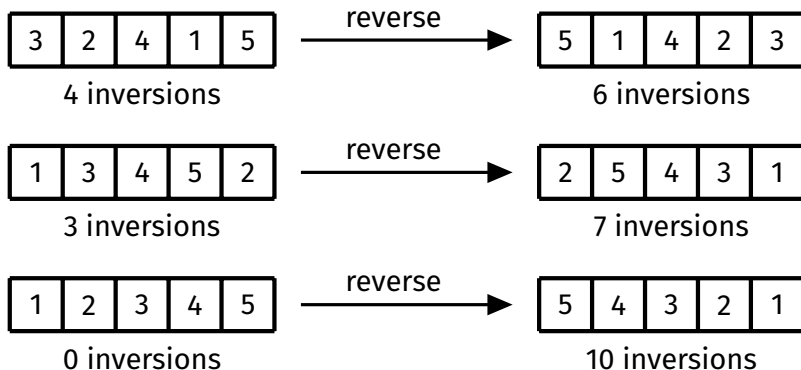
Bubble sort average
caseInsertion sort
walkthrough

In a randomly sorted array:

- The minimum possible number of inversions is 0 (sorted array)
- The maximum possible number of inversions is $\frac{1}{2}n(n-1)$ (reverse-sorted array)

Let k be the number of inversions in a random permutation.
By reversing this permutation, one can obtain a permutation with $\frac{1}{2}n(n-1) - k$ inversions.

For example, suppose $n = 5$:



Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

Thus, if we take all the possible permutations of an array and pair each permutation with its reverse, the total number of inversions in each pair is $\frac{1}{2}n(n-1)$.

This implies that the average number of inversions across all permutations is $\frac{1}{4}n(n-1)$, which is $O(n^2)$.

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

Sort the following array:

4	2	1	5	3
---	---	---	---	---

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

[0]	[1]	[2]	[3]	[4]
lo				hi
4	2	1	5	3

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

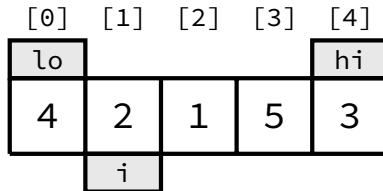
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

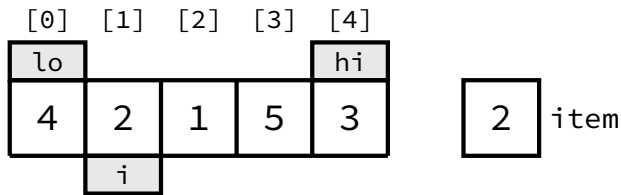
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

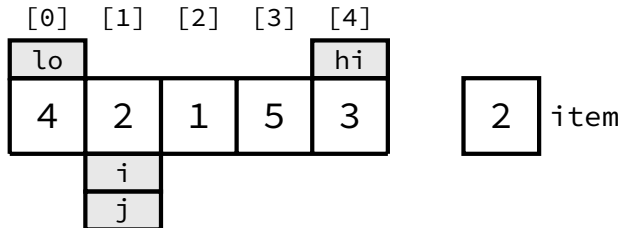
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

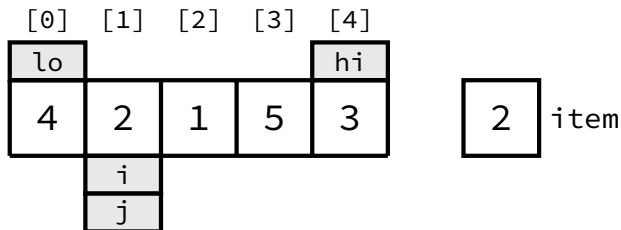
Bubble Sort

Insertion Sort

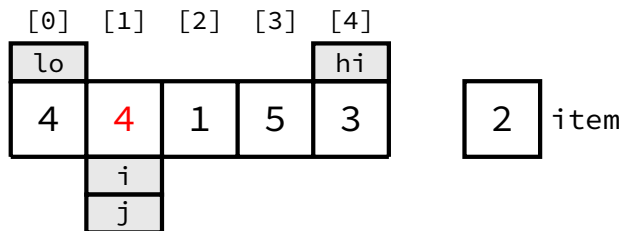
Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```



```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```


Selection Sort

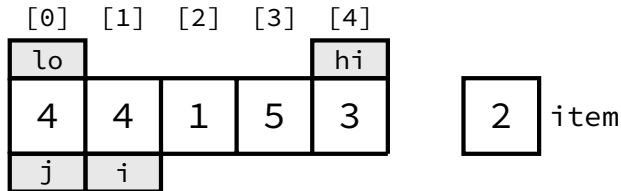
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

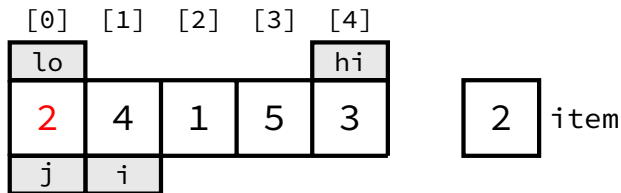
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

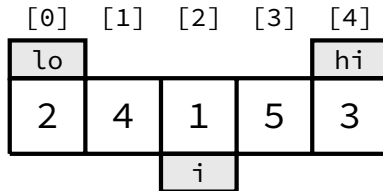
Bubble Sort

Insertion Sort

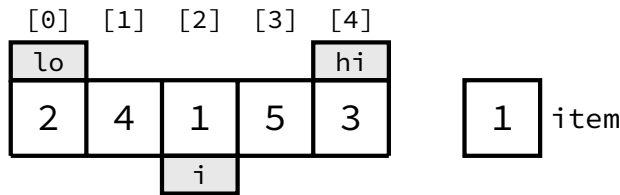
Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```



```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

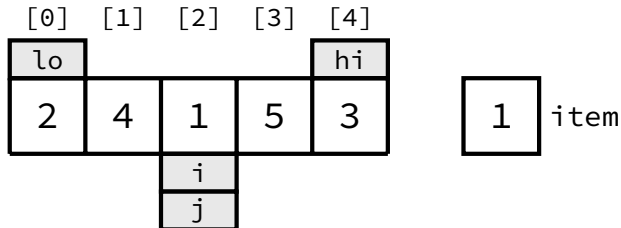
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

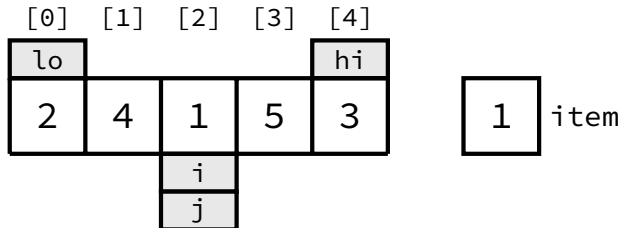
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

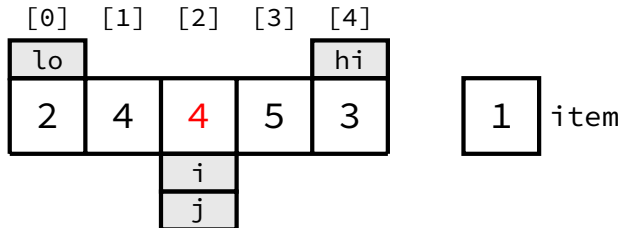
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

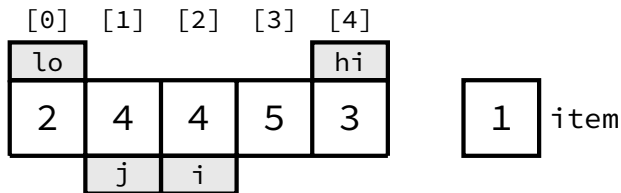
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```


Selection Sort

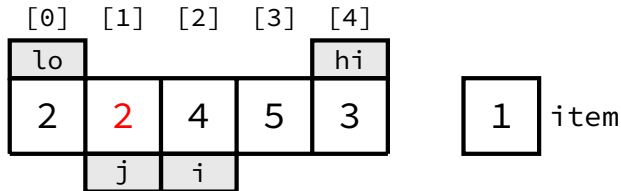
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

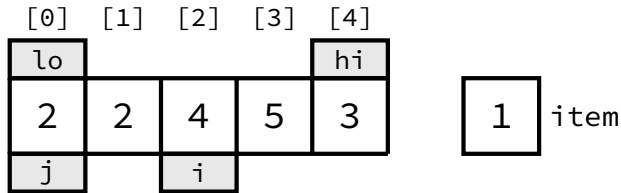
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

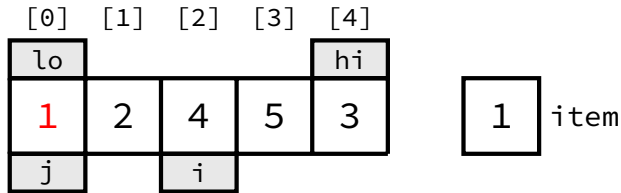
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

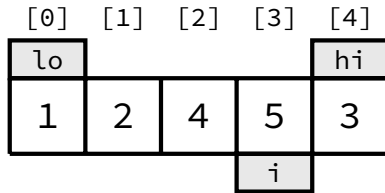
Bubble Sort

Insertion Sort

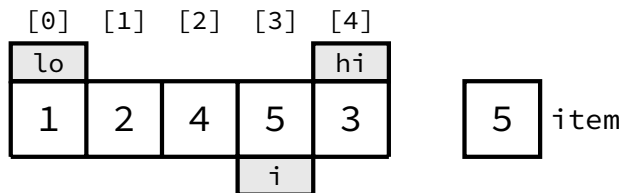
Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```



```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

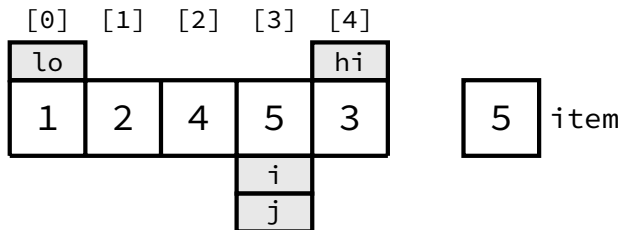
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

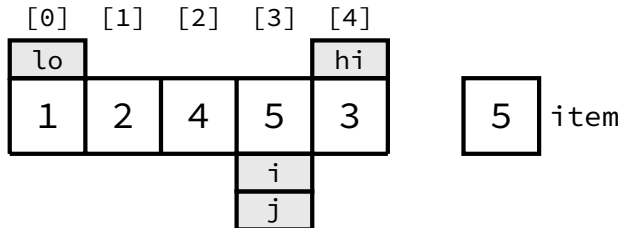
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

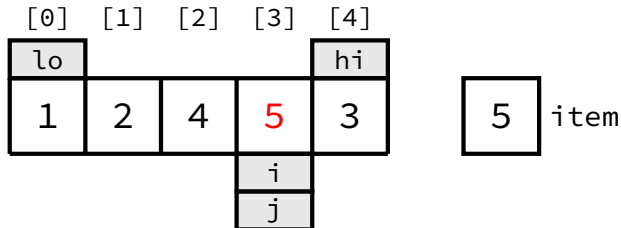
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```


Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

[0]	[1]	[2]	[3]	[4]
lo				hi
1	2	4	5	3
				i

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

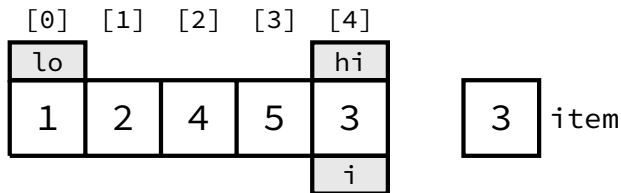
Bubble Sort

Insertion Sort

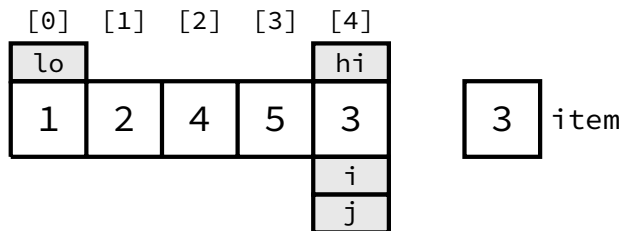
Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```



```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

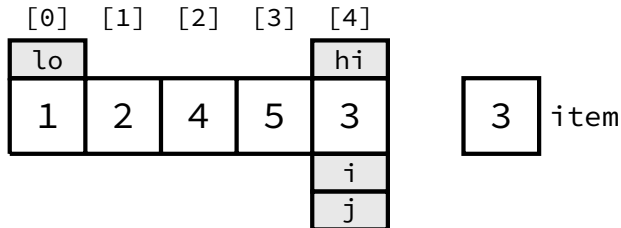
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

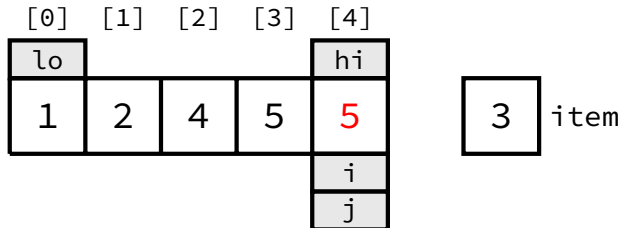
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

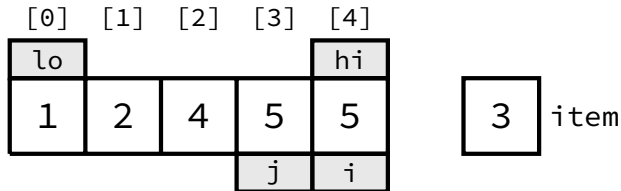
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

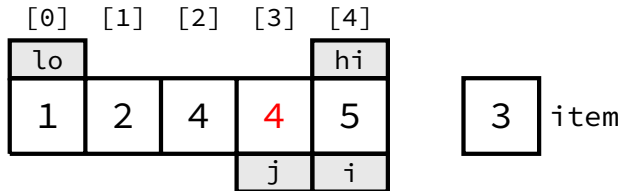
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

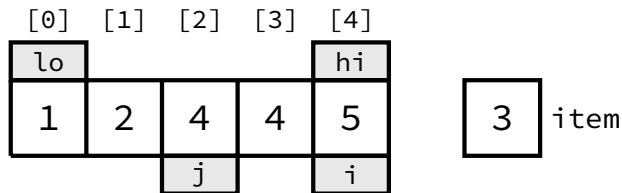
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```


Selection Sort

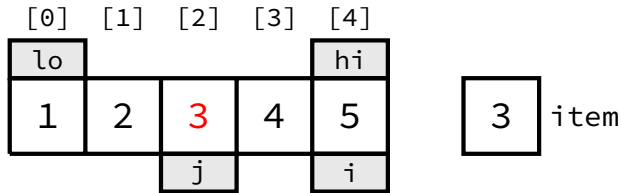
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

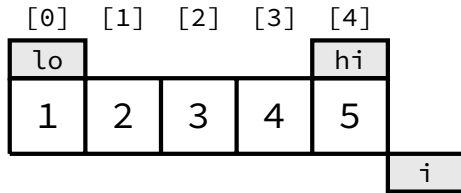
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```

Selection Sort

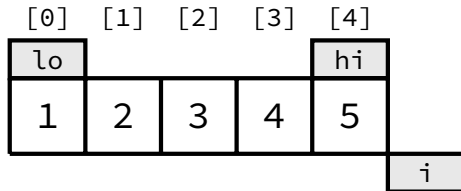
Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average
caseInsertion sort
walkthrough

```
void insertionSort(Item items[], int lo, int hi) {  
    for (int i = lo + 1; i <= hi; i++) {  
        Item item = items[i];  
        int j = i;  
        for (; j > lo && lt(item, items[j - 1]); j--) {  
            items[j] = items[j - 1];  
        }  
        items[j] = item;  
    }  
}
```