Selection Sort Bubble Sort Insertion Sort Summary Sorting Lists Appendix

COMP2521 25T3

Sorting Algorithms (II) Elementary Sorting Algorithms

Sim Mautner

cs2521@cse.unsw.edu.au

selection sort bubble sort insertion sort

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.

Example

Analysis Properties

Bubble Sort

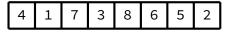
Insertion Sort

Summary

Sorting Lists

Appendix

Example



Example

Selection Sort

Example

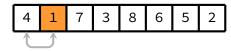
Analysis Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists



Example

Selection Sort

Example

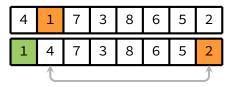
Analysis Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists



Example

Selection Sort

Example

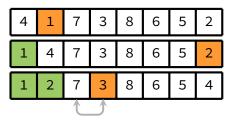
Analysis

Bubble Sort

Insertion Sort

Summary

Sorting Lists



Example

Selection Sort

Example

Analysis

Bubble Sort

Insertion Sort

Summary

Sorting Lists

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

Example

Selection Sort

Example

Analysis

Bubble Sort

Insertion Sort

Summary

Sorting Lists

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

Example

Selection Sort

Example

Analysis

Bubble Sort

Insertion Sort

Summary

Sorting Lists

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

Example

Selection Sort

Example

Analysis Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

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

Example

Selection Sort

Example

Analysis Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

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

C Implementation

```
Selection Sort
Example
Implementation
Analysis
Properties
```

Bubble Sort
Insertion Sort

Summary

Sorting Lists

```
void selectionSort(Item items[], int lo, int hi) {
    for (int i = lo; i < hi; i++) {</pre>
        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:

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

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

Selection Sort Properties

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:







Selection Sort
Example
Implementation
Analysis

Properties

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

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

Bubble Sort

Example Implmentation Analysis

Insertion Sort

Summary

Sorting Lists

Appendix

Method:

- Make multiple passes from left (lo) 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

Bubble Sort

Example

Analysis

Properties

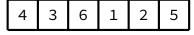
Insertion Sort

Summary

Sorting Lists

Appendix

Example



Example

Selection Sort

Bubble Sort

Example

Analysis

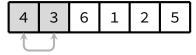
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Selection Sort

Bubble Sort

Example

Implmentation Analysis

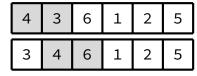
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Selection Sort
Bubble Sort

Example

Implmentation Analysis

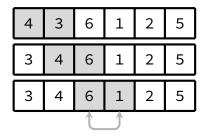
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort Example

Selection Sort

Bubble Sort Example

Implmentation Analysis

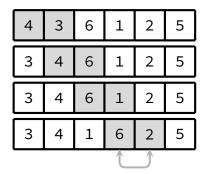
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort Example

Selection Sort

Bubble Sort

Example

Implmentat Analysis

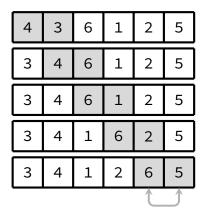
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort Example

Selection Sort

Bubble Sort

Example

Analysis

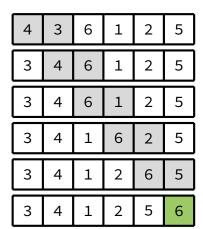
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Selection Sort

Bubble Sort

Example Implmentati

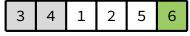
Analysis Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Selection Sort

Bubble Sort

Example

Analysis

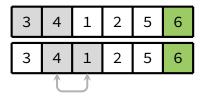
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Selection Sort

Bubble Sort

Example Implmentation

Analysis

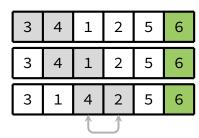
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Implmentation Analysis

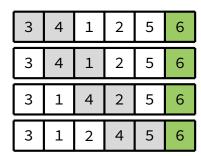
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Example

Implmentation
Analysis

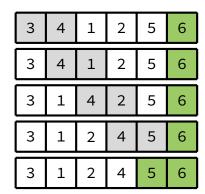
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort

Example

Implmentati Analysis

Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort

Example

Implmentati Analysis

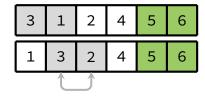
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort

Example Implmentation

Analysis

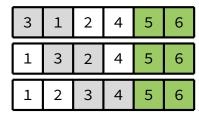
Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort

Example

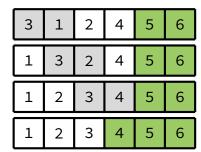
Implmentation Analysis Properties

Insertion Sort

Summary

Sorting Lists

Appendix



Bubble Sort Example

Implmentation
Analysis

Properties

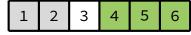
Insertion Sort

Summary

Sorting Lists

Appendix

Fourth pass



Bubble Sort Example

Implmentatio Analysis

Properties

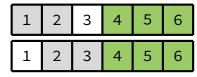
Insertion Sort

Summary

Sorting Lists

Appendix

Fourth pass



Bubble Sort

Example

mplmentati Analysis

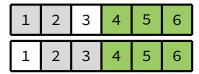
Insertion Sort

Summary

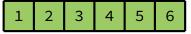
Sorting Lists

Appendix

Fourth pass



No swaps made; stop



Bubble Sort C Implementation

Selection Sort

Bubble Sort

Implmentation

Insertion Sort

Summary

Sorting Lists

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

Bubble Sort Analysis

Selection Sort
Bubble Sort

Analysis Properties

Insertion Sort

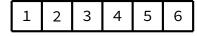
Summary

Sorting Lists

Appendix

Best case: Array is sorted

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



Analysis

Bubble Sort

Example Implmentation

Propertio

Insertion Sort

Summary

Sorting Lists

Appendix

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: $(n-1) + (n-2) + ... + 1 = \frac{1}{2}n(n-1)$
- 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 Implmentation Analysis

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

Bubble Sort Properties

Selection Sort

Bubble Sort

Implmentation

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

Implementa Analysis

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

Example

Selection Sort

Bubble Sort

Insertion Sort

Example

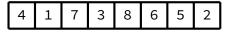
Analysis

Summary

Sorting Lists

Appendix

Example



Selection Sort

Bubble Sort

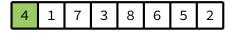
Insertion Sort

Example

Analysis
Properties

Summary

Sorting Lists



Example

Selection Sort

Bubble Sort

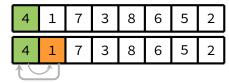
Insertion Sort

Example

Analysis Properties

Summary

Sorting Lists



Example

Selection Sort

Bubble Sort

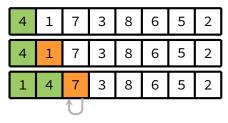
Insertion Sort

Example Implementation

Analysis Properties

Summary

Sorting Lists



Selection Sort

Bubble Sort

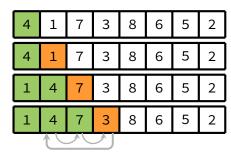
Insertion Sort

Example

Implementation Analysis

Summary

Sorting Lists



Example

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation Analysis Properties

Summary

Sorting Lists

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

Selection Sort

Bubble Sort

Insertion Sort

Example

Implementation Analysis Properties

Summary

Sorting Lists

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

Selection Sort

Bubble Sort

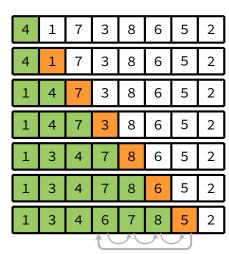
Insertion Sort

Example Implementation

Analysis Properties

Summary

Sorting Lists



Selection Sort

Bubble Sort

Insertion Sort

Example

Analysis

Summary

Sorting Lists

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

Example

Selection Sort

Bubble Sort

Insertion Sort

Example

Analysis Properties

Summary

Sorting Lists

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

C Implementation

Selection Sort **Bubble Sort**

Insertion Sort

Implementation Analysis

Summary

Sorting Lists

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

Analysis

Selection Sort

Bubble Sort

Insertion Sort

Analysis

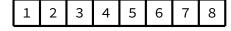
Summary

Appendix

Sorting Lists

Best case: Array is sorted

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



Analysis

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: $1 + 2 + ... + (n-1) = \frac{1}{2}n(n-1)$
- Worst-case time complexity: $O(n^2)$

Analysis

Selection Sort

Bubble Sort

Insertion Sort
Example
Implementation

Analysis Propertie

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

Insertion Sort Properties

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 $\mathcal{O}(n^2)$ on average, $\mathcal{O}(n)$ if input array is sorted

In-place

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

Summary of Elementary Sorts

Selection Sort Bubble Sort

Insertion Sort

Summary

Sorting Lists
Appendix

	Tin	ne complex	Properties		
	Best	Average	verage Worst		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

Aside: Sorting Linked Lists

Selection Sort
Bubble Sort
Insertion Sort
Summary

Sorting Lists

Annandiy

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

Bubble sort average case Insertion sort walkthrough

Selection Sort

Bubble Sort

Insertion Sort

Summary

Sorting Lists

Appendix

Bubble sort average

Insertion sort

Note: Not required knowledge in COMP2521!

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:

The array contains 5 inversions: (4, 2), (4, 1), (4, 3), (2, 1), (5, 3)

Continued

Selection Sort
Bubble Sort
Insertion Sort

Summary

Sorting Lists

Bubble sort average case
Insertion sort

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 ${\cal O}(n^2)$ 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 $O(n^2)$)

Continued

Selection Sort
Bubble Sort

Insertion Sort

Summary
Sorting Lists

Appendix

Bubble sort average case

Insertion sort

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)

Bubble Sort - Proof of $\mathcal{O}(n^2)$ Average Case

Continued

Selection Sort

Insertion Sort

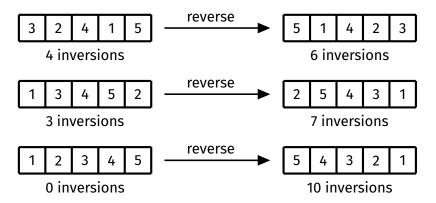
Summary
Sorting Lists

Appendix

Bubble sort average case

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:



Continued

Selection Sort
Bubble Sort

Insertion Sort

Summary
Sorting Lists

Appendix

Bubble sort average case

Insertion 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 aver

Insertion sort walkthrough Sort the following array:

```
4 2 1 5 3
```

Selection Sort Bubble Sort

Insertion Sort
Summary

Sorting Lists

Appendix

Bubble sort average case

Insertion 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

walkthrough

Appendix

Bubble sort average case

```
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 averag
case

Insertion sort
walkthrough

```
[0] [1] [2] [3] [4]

lo hi

4 2 1 5 3 2 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
case

Insertion sort
walkthrough

```
[0] [1] [2] [3] [4]

lo hi

4 2 1 5 3 2 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
case
Insertion sort
walkthrough

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

walkthrough

```
[0]
       \lceil 1 \rceil
              [2] [3]
                            [4]
                              hi
lo
                              3
                                                   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[i] = items[i - 1];
        items[i] = item;
```

Selection Sort Bubble Sort

Insertion Sort
Summary

Sorting Lists

Appendix

Bubble sort average case

Insertion sort walkthrough

```
[0] [1] [2] [3] [4]

lo hi

4 4 1 5 3 2 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

```
[0] [1] [2] [3] [4]

lo hi

4 4 1 5 3 2 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

```
[0] [1] [2] [3] [4]

lo hi

2  4  1  5  3  2 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

```
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

```
[0] [1] [2] [3] [4]

lo hi

2 4 1 5 3 1 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

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

```
[0] [1] [2] [3] [4]

lo hi

2 4 1 5 3 1 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

```
[0] [1] [2] [3] [4]

lo hi

2 4 4 5 3 1 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

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

```
[0] [1] [2] [3] [4]

lo hi

2 2 4 5 3 1 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

```
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

```
[0] [1] [2] [3] [4]

lo hi

2 2 4 5 3 1 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

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

1 2 4 5 3 1 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

```
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

```
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

Bubble sort average

walkthrough

```
[0]
     [1]
          [2] [3]
                    [4]
                     hi
lo
                      3
                                     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[i] = items[i - 1];
        items[i] = item;
```

Selection Sort Bubble Sort

Insertion Sort
Summary

Sorting Lists

```
[0] [1] [2] [3] [4]

lo hi

1 2 4 5 3 5 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

```
[0] [1] [2] [3] [4]

lo hi

1 2 4 5 3 5 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

```
[0] [1] [2] [3] [4]

lo hi

1 2 4 5 3 5 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 averag

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

```
3 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

```
3 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

```
3 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**

walkthrough

```
lo
Insertion Sort
Summary
Sorting Lists
Bubble sort average
```

[0]

 $\lceil 1 \rceil$

[2] [3]

[4]

hi

5

```
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[i] = items[i - 1];
        items[i] = item;
```

Selection Sort Bubble Sort

Insertion Sort
Summary

Sorting Lists

```
3 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

Bubble Sort
Insertion Sort
Summary

Sorting Lists

```
3 litem
```

```
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**

Bubble sort average walkthrough

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

```
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[i] = items[i - 1];
        items[i] = item;
```

item

Selection Sort
Bubble Sort
Insertion Sort

Summary

Sorting Lists

```
3 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

```
3 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

walkthrough

Appendix

Bubble sort average case

```
[0] [1] [2] [3] [4] hi hi label{eq:10} 2 3 4 5 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

```
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;
    }
}
```