NANYANG TECHNOLOGICAL UNIVERSITY

SC2001/CE2101/CZ2101: Algorithm Design and Analysis

Insertion Sort

Instructor: Assoc. Prof. ZHANG Hanwang

Courtesy of Dr. Ke Yiping, Kelly’s slides

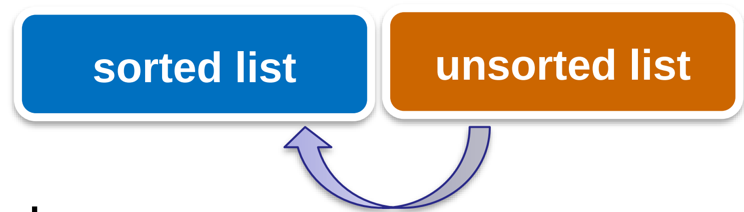
Learning Objectives

At the end of this lecture, students should be able to:

- Explain the **incremental approach** as a strategy of algorithm design
- Describe how **insertion sort** algorithm works, by manually running its pseudo code on a toy example
- Analyse the **time complexities** of Insertion sort in the best case, worst case and average case

Insertion Sort of a Hand of Cards

Insertion Sort



The incremental approach

- An intuitive, primitive sorting method
- A form of insertion into an ordered list
- Given an unordered set of objects, repeatedly remove an entry from the set and insert it into a new ordered list
- Ensure that the new list is ordered at all times
- Each insertion requires movements of certain entries in the ordered list

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
                if (slot[j].key < slot[j-1].key)
                        swap(slot[j], slot[j-1]);
                else break;
        }
}
```

Insertion Sort (Pseudo Code)

The incremental approach

```
class ALIST {
    KeyType    key;
    DataType   data;
};
```

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
      for (int j=i; j > 0; j--) {
          if (slot[j].key < slot[j-1].key)
              swap(slot[j], slot[j-1]);
          else break;
      }
}
```

45	29	06	64	12	16

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
      for (int j=i; j > 0; j--) {
          if (slot[j].key < slot[j-1].key)
              swap(slot[j], slot[j-1]);
          else break;
      }
}
```

0	1	2	3	4	5
45	29	06	64	12	16

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```

0	1	2	3	4	5
45	29	06	64	12	16

Insertion Sort (Pseudo Code)

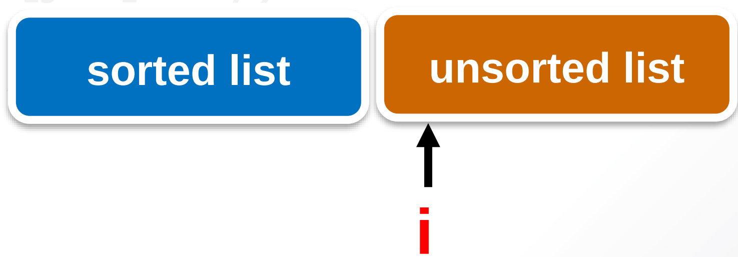
The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
          for (int j=i; j > 0; j--) {
                    if (slot[j].key < slot[j-1].key)
                              swap(slot[j], slot[j-1]);
                    else break;
          }
}
```

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)      Pick up a new item from slot[ ]
       for (int j=i; j > 0; j--) {
            if (slot[j].key < slot[j-1].key)
                 swap(slot[j]
            else break;
       }
}
```



Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```

Find the correct position to insert the item.

sorted list | unsorted list

i,j

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
                if (slot[j].key < slot[j-1].key)
                        swap(slot[j], slot[j-1]);
                else break;
        }
}
```

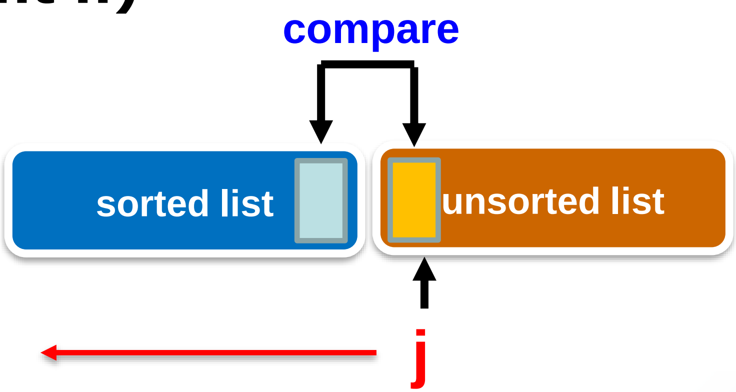
sorted list | unsorted list

j

Insertion Sort (Pseudo Code)

The incremental approach

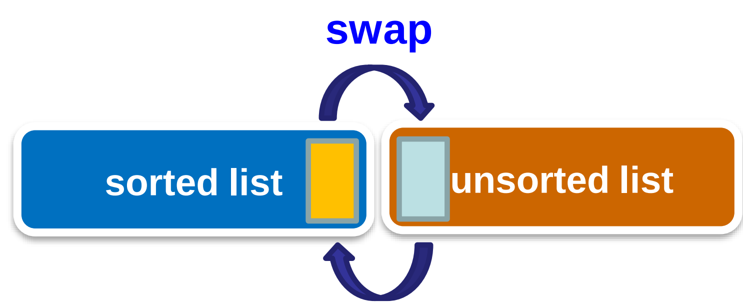
```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```



Insertion Sort (Pseudo Code)

The incremental approach

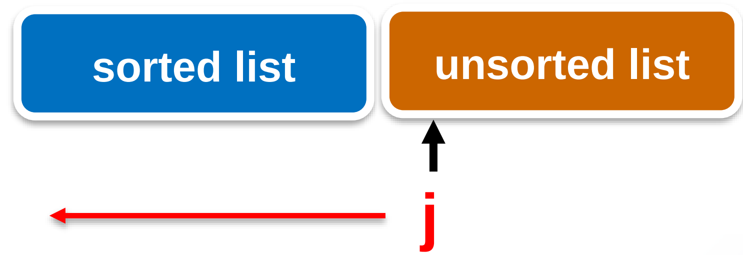
```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```



Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```



Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
 // assume n > 1;
 for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
                if (slot[j].key < slot[j-1].key)
                        swap(slot[j], slot[j-1]);
                else break;
        }
}
```

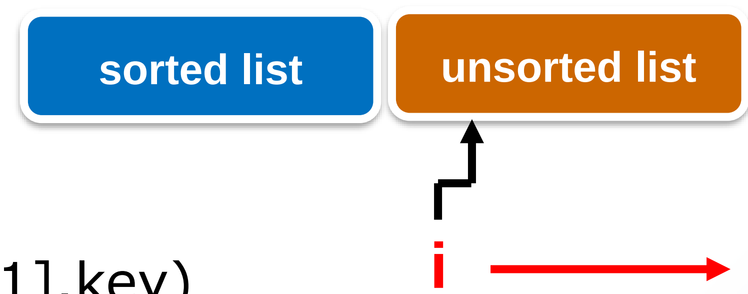
What does it mean?

The correct position was found!!

Insertion Sort (Pseudo Code)

The incremental approach

```
void InsertionSort (ALIST slot[ ], int n)
{ // input slot is an array of n records;
  // assume n > 1;
  for (int i=1; i < n; i++)
        for (int j=i; j > 0; j--) {
              if (slot[j].key < slot[j-1].key)
                    swap(slot[j], slot[j-1]);
              else break;
        }
}
```



NANYANG TECHNOLOGICAL UNIVERSITY

Insertion Sort Example

Insertion Sort Example

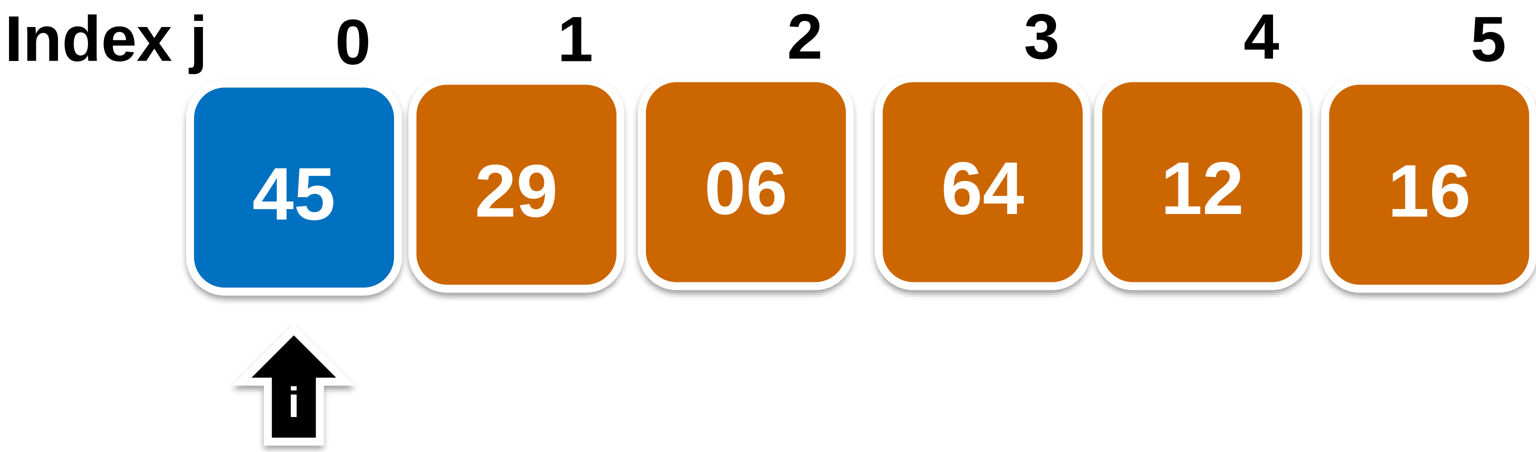
Sort in ascending order

Index j	0	1	2	3	4	5
	45	29	06	64	12	16

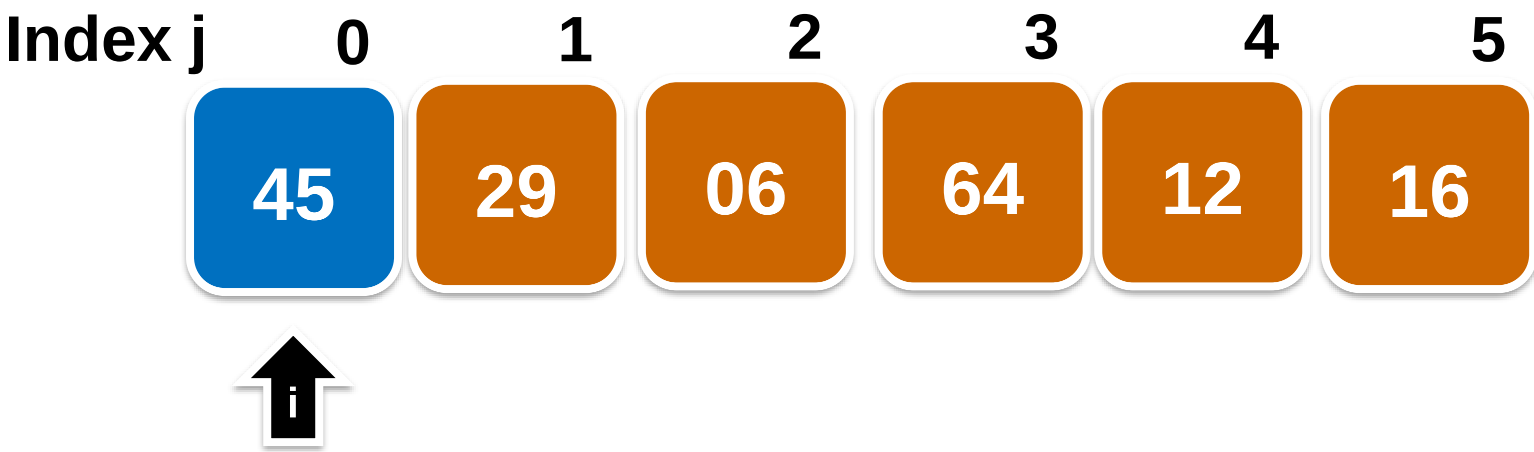
Insertion Sort Example

Index j	0	1	2	3	4	5
	45	29	06	64	12	16
	i					

Insertion Sort Example



Insertion Sort Example



Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**1**].key < slot[**0**].key)

29 < **45** ✓

Index j	0	1	2	3	4	5
	45	29	06	64	12	16
		i				

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[**1**], slot[**0**]);

Index j	0	1	2	3	4	5
	45	29	06	64	12	16
		i				

Insertion Sort Example

Index j	0	1	2	3	4	5
	29	45	06	64	12	16
		i				

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**2**].key < slot[**1**].key)

06 < **45** ✓

Index j	0	1	2	3	4	5
	29	45	06	64	12	16

i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[**2**], slot[**1**]);

Index j	0	1	2	3	4	5
	29	45	06	64	12	16

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**1**].key < slot[**0**].key)

06 < **29** ✓

Index j	0	1	2	3	4	5
	29	06	45	64	12	16

i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[**1**], slot[**0**]);

Index j	0	1	2	3	4	5
	29	06	45	64	12	16

i

Insertion Sort Example

Index j	0	1	2	3	4	5
	06	29	45	64	12	16

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**3**].key < slot[**2**].key)

64 < **45** ✘

Index j	0	1	2	3	4	5
	06	29	45	64	12	16
			i			

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**4**].key < slot[**3**].key)

12 < **64** ✓

Index j	0	1	2	3	4	5
	06	29	45	64	12	16
				i		

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[4], slot[3]);

Index j	0	1	2	3	4	5
	06	29	45	64	12	16

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**3**].key < slot[**2**].key)

12 < **45** ✓

Index j	0	1	2	3	4	5
	06	29	45	12	64	16

i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[3], slot[2]);

Index j	0	1	2	3	4	5
	06	29	45	12	64	16
					i	

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**2**].key < slot[**1**].key)

12 < **29** ✓

Index j	0	1	2	3	4	5
	06	29	12	45	64	16

i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[**2**], slot[**1**]);

Index j	0	1	2	3	4	5
	06	29	12	45	64	16
					i	

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**1**].key < slot[**0**].key)

12 < **06** ✘

Index j	0	1	2	3	4	5
	06	12	29	45	64	16

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**5**].key < slot[**4**].key)

16 < **64** ✓

Index j	0	1	2	3	4	5
	06	12	29	45	64	16

i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[**5**], slot[**4**]);

Index j	0	1	2	3	4	5
	06	12	29	45	64	16

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**4**].key < slot[**3**].key)

16 < **45** ✓

Index j	0	1	2	3	4	5
	06	12	29	45	16	64
						i

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[4], slot[3]);

Index j	0	1	2	3	4	5
	06	12	29	45	16	64

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**3**].key < slot[**2**].key)

16 < **29**

Index j	0	1	2	3	4	5
	06	12	29	16	45	64

Insertion Sort Example

swap(slot[j], slot[j-1]);

swap(slot[3], slot[2]);

Index j	0	1	2	3	4	5
	06	12	29	16	45	64

i

Insertion Sort Example

If (slot[j].key < slot[j-1].key)

(slot[**2**].key < slot[**1**].key)

16 < **12** ✘

Index j	0	1	2	3	4	5
	06	12	16	29	45	64

i

Insertion Sort Example

Sorted in ascending order

Index j	0	1	2	3	4	5
	06	12	16	29	45	64

Insertion Sort Algorithm
(Recap)

NANYANG TECHNOLOGICAL UNIVERSITY

Insertion Sort Algorithm

- Original unsorted set and final sorted list are both in array slot[ ]. (in-place sorting)

06	12	29	45	64	16

NANYANG TECHNOLOGICAL UNIVERSITY

Insertion Sort Algorithm

- Original unsorted set and final sorted list are both in array slot[ ]. (in-place sorting)
- Since sorting is performed directly on original array without any working storage, swapping and shifting are essential.

06	12	29	45	64	16

NANYANG TECHNOLOGICAL UNIVERSITY

Insertion Sort Algorithm

- Original unsorted set and final sorted list are both in array slot[ ].
- Since sorting is performed directly on original array without any working storage, swapping and shifting are essential.
- During sorting, slot[ ] contains sorted portion on the ‘left’ and unsorted portion on the ‘right’; sorted portion grows while unsorted portion shrinks.

Sorted				Unsorted	
06	12	29	45	64	16

Insertion Sort Algorithm

- In the outer ‘for’ loop, i begins with 1 because the ordered list begins with one element (slot[0]); hence slot[1] is the first element from the unordered list.

```
for (int i=1; i < n; i++)
    for (int j=i; j > 0; j--) {
        if (slot[j].key < slot[j-1].key)
            swap(slot[j], slot[j-1]);
        else break;
    }
```

Insertion Sort Algorithm

- At each iteration, number at slot[i] is inserted into the new ordered list.

```
for (int i=1; i < n; i++)
    for (int j=i; j > 0; j--) {
        if (slot[j].key < slot[j-1].key)
            swap(slot[j], slot[j-1]);
        else break;
```

NANYANG TECHNOLOGICAL UNIVERSITY

Insertion Sort Algorithm

- The inner ‘for’ loop finds the correct position in the ordered list by swapping slot[j] with slot[j-1] as long as the key of slot[j-1] is > the key of slot[j].

```
for (int i=1; i < n; i++)
    for (int j=i; j > 0; j--) {
        if (slot[j].key < slot[j-1].key)
            swap(slot[j], slot[j-1]);
        else break;
```

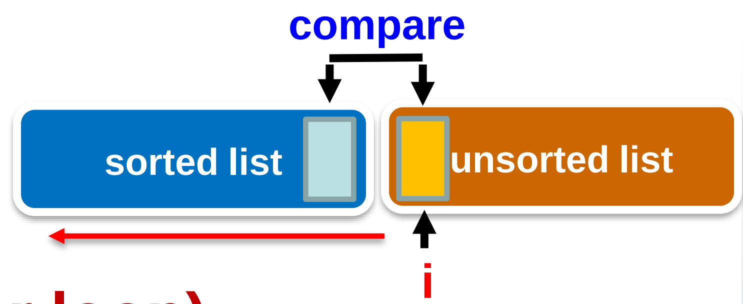
Insertion Sort Algorithm

- The inner ‘for’ loop finds the correct position in the ordered list by swapping slot[j] with slot[j-1] as long as the key of slot[j-1] is > the key of slot[j].

```
for (int i=1; i < n; i++)
    for (int j=i; j > 0; j--) {
        if (slot[j].key < slot[j-1].key)
            swap(slot[j], slot[j-1]);
        else break;
```

Complexity of Insertion Sort

Complexity of Insertion Sort



Number of key comparisons:

- There are $n - 1$ iterations **(the outer loop)**
- **Best case:** 1 key comparison/ iteration, **total: $n - 1$**
  - **Already sorted:** [06] [12] [16] [29] [45] [64]
- **Worst case:** i key comparisons for the ith iteration
  - **Reversely sorted:** [64] [45] [29] [16] [12] [06]

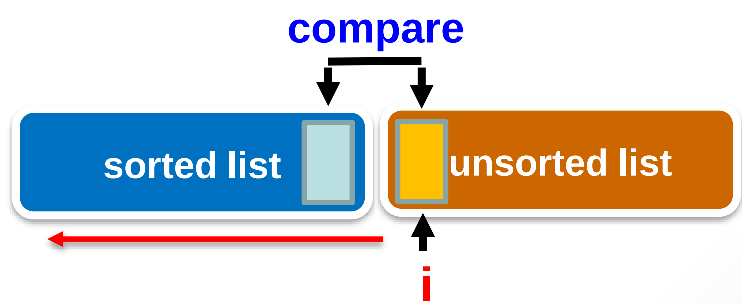
Total: $$1+2+3+\ldots+(n-1)=\sum_{i=1}^{n-1} i=\frac{(n-1)n}{2}$$

Insertion Sort Performance

- **Average case:** the *i*th iteration may have 1, 2, …, *i* key comparisons, each with 1/*i* chance.

The average no. of comparisons in the *i*th iteration:

$$\frac{1}{i}\sum_{j=1}^{i} j = \frac{1}{i}(1+2+\ldots+i)$$



Summation for the ***n*-1** iterations:

$$1+\frac{1}{2}(1+2)+\frac{1}{3}(1+2+3)+\ldots+\frac{1}{n-1}(1+\ldots+n-1) = \sum_{i=1}^{n-1}\left(\frac{1}{i}\sum_{j=1}^{i} j\right)$$

$$= \sum_{i=1}^{n-1}\left(\frac{1}{i}\frac{i(i+1)}{2}\right) = \frac{1}{2}\sum_{i=1}^{n-1}(i+1) = \frac{1}{2}\left(\frac{(n-1)(n+2)}{2}\right) = \Theta(n^2)$$

Insertion Sort Performance

☺ Strengths:

- ☞ Good when the unordered list is almost sorted.
- ☞ Need minimum time to verify if the list is sorted.
- ☞ Fast with linked storage implementation: no movement of data.

☹ Weaknesses:

- ☞ When an entry is inserted, it may still not be in the final position yet.
- ☞ Every new insertion necessitates movements for some inserted entries in ordered list.
- ☞ When each slot is large (e.g., a slot contains a large record of 10Mb), movement is expensive.
- ☞ Less suitable with contiguous storage implementation.

Summary

- Insertion sort uses the incremental approach.
- **Main idea:** Repeatedly pick up an element *x* to insert into a sorted sub-array on the left side, by comparing *x* with its left neighbour. If they are out of order, swap them; otherwise, insert *x* there.

Index j	**0**	**1**	**2**	**3**	**4**	**5**
	45	29	06	64	12	16

Summary

- Insertion sort uses the incremental approach.
- **Main idea:** Repeatedly pick up an element *x* to insert into a sorted sub-array on the left side, by comparing *x* with its left neighbour. If they are out of order, swap them; otherwise, insert *x* there.

Index j	0	1	2	3	4	5
	45	29	06	64	12	16

Summary

- Insertion sort uses the incremental approach.
- **Main idea:** Repeatedly pick up an element *x* to insert into a sorted sub-array on the left side, by comparing *x* with its left neighbour. If they are out of order, swap them; otherwise, insert *x* there.

Index j	**0**	**1**	**2**	**3**	**4**	**5**
	45	29	06	64	12	16

Summary

- Insertion sort uses the incremental approach.
- **Main idea:** Repeatedly pick up an element *x* to insert into a sorted sub-array on the left side, by comparing *x* with its left neighbour. If they are out of order, swap them; otherwise, insert *x* there.

Index j	**0**	**1**	**2**	**3**	**4**	**5**
	45	29	06	64	12	16

Summary

- Insertion sort uses the incremental approach.
- **Main idea:** Repeatedly pick up an element *x* to insert into a sorted sub-array on the left side, by comparing *x* with its left neighbour. If they are out of order, swap them; otherwise, insert *x* there.
- **Time complexity analysis:**
  - Best case: $\Theta(n)$, when input array is already sorted.
  - Worst case: $\Theta(n^2)$, when input array is reversely sorted.
  - Average case: $\Theta(n^2)$.