Sorting algorithms

Data Structures and Algorithms

Luu Quang Huan, MsC
Faculty of Computer Science and Engineering
Ho Chi Minh University of Technology, VNU-HCM

Overview

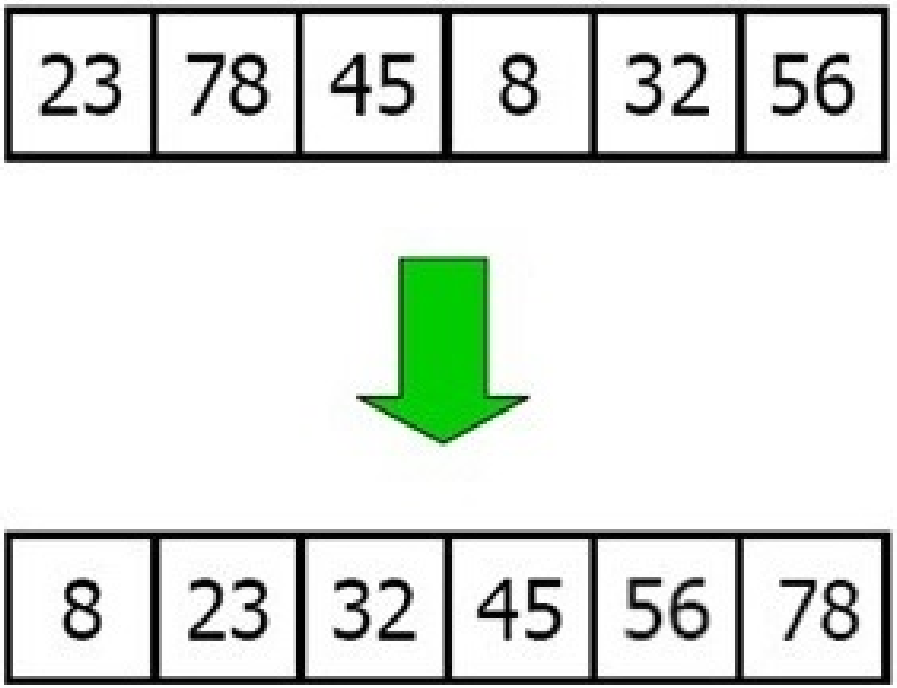
1. **Sorting concepts**
2. **Insertion Sort**
   - Straight Insertion Sort
   - Shell Sort
3. **Selection Sort**
   - Straight Selection Sort
4. **Exchange Sort**
   - Bubble Sort
5. **Devide-and-Conquer**
   - Quick Sort
   - Merge Sort

Luu Quang Huan, MsC

Sorting concepts

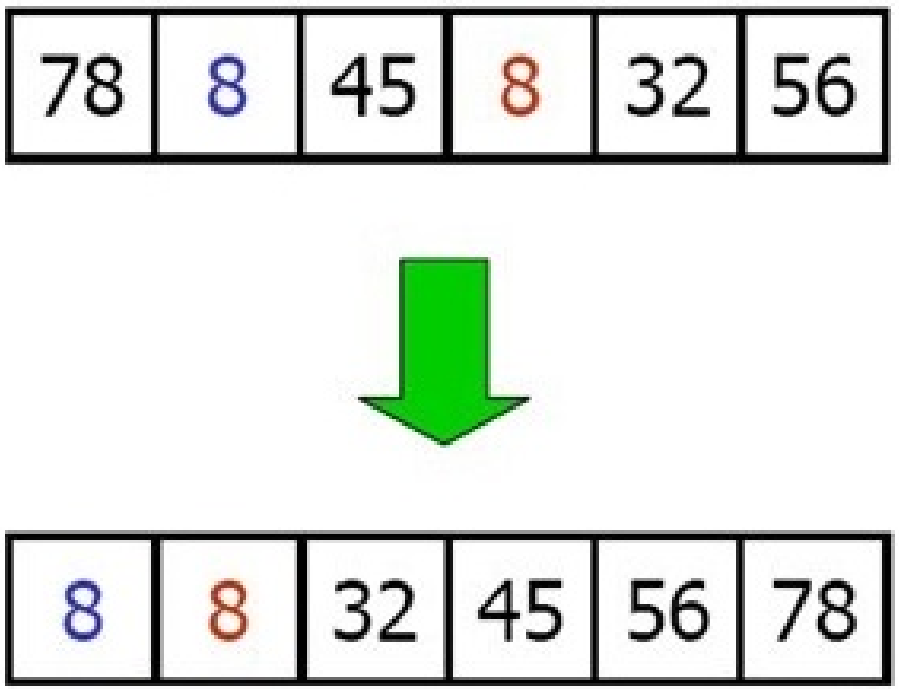
Sorting

One of the most important concepts and common applications in computing.



Sorting

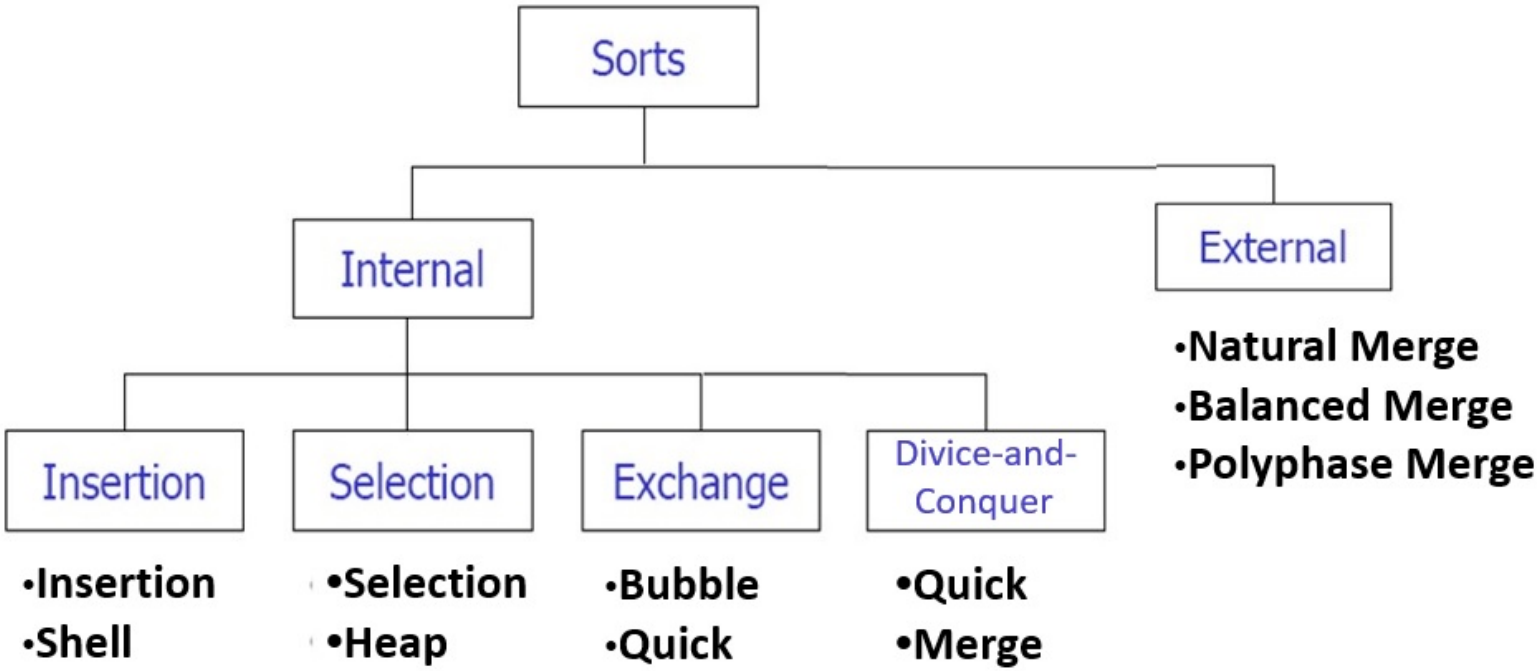
Sort stability: data with equal keys maintain their relative input order in the output.



Sorting

Sort efficiency: a measure of the relative efficiency of a sort = number of comparisons + number of moves.

Sorting



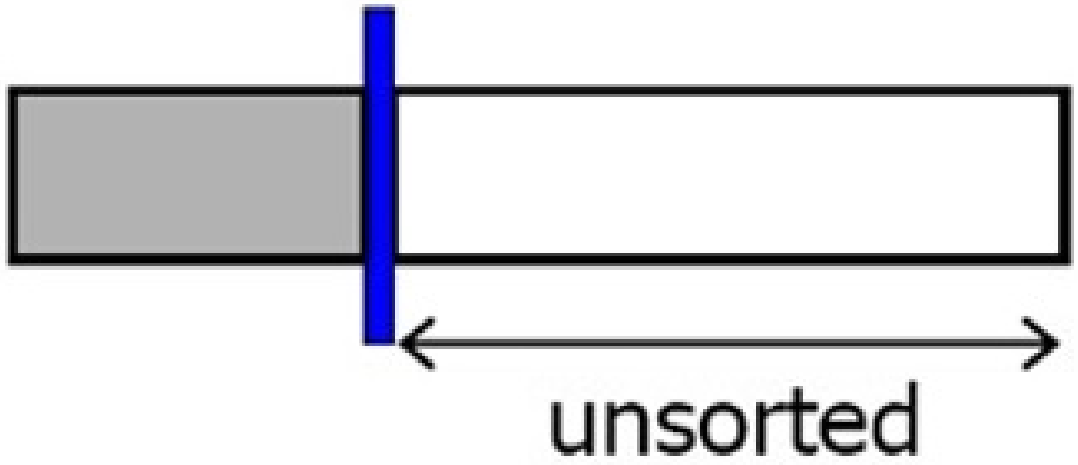
Luu Quang Huan, MsC

Insertion Sort

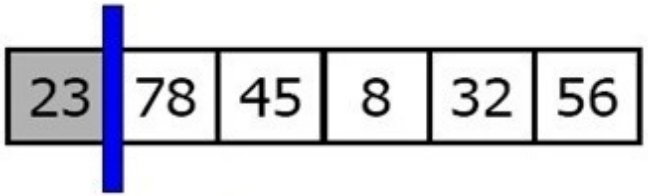
Luu Quang Huan, MsC

Straight Insertion Sort

- The list is divided into two parts: sorted and unsorted.
- In each pass, the first element of the unsorted sublist is inserted into the sorted sublist.

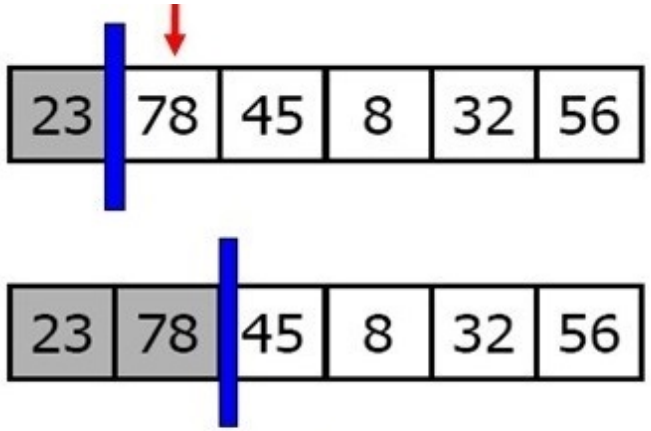


Straight Insertion Sort

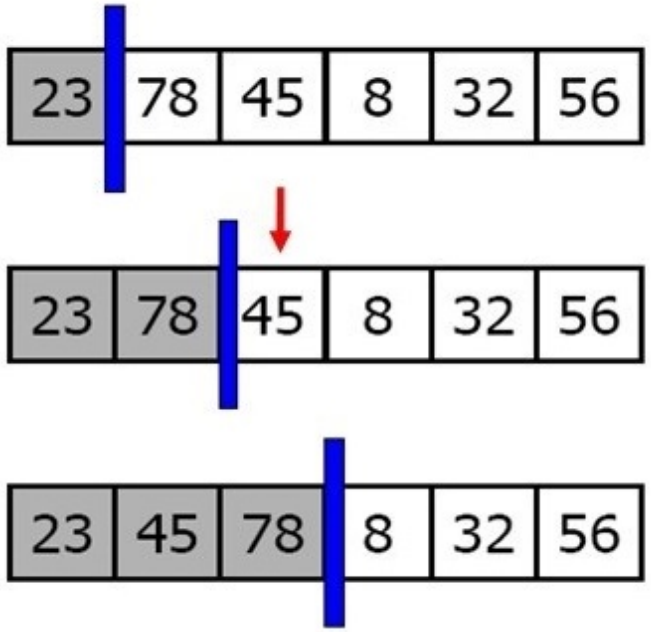


Luu Quang Huan, MsC

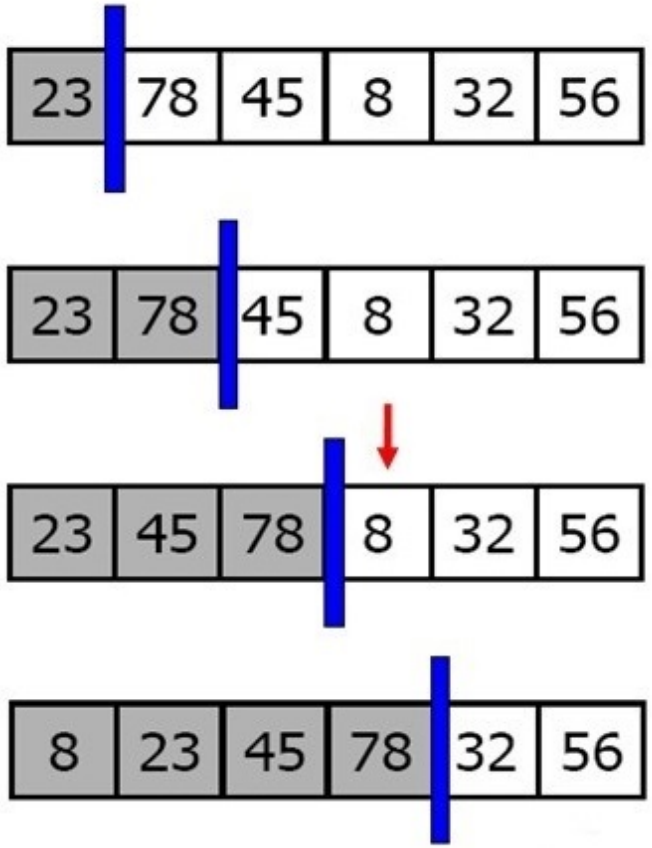
Straight Insertion Sort



Straight Insertion Sort

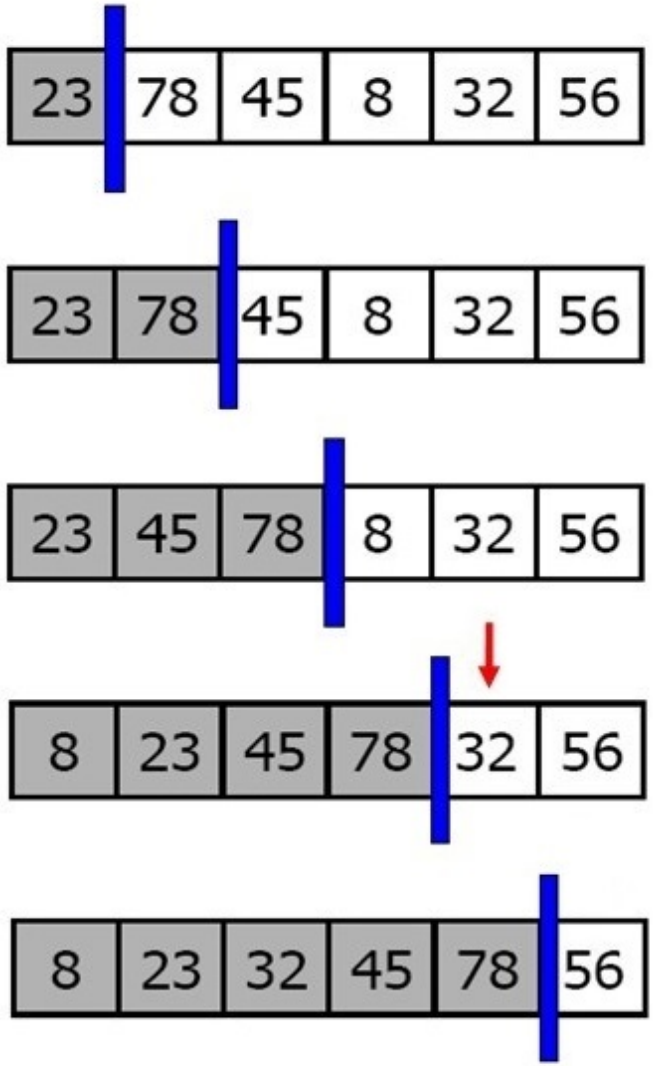


Straight Insertion Sort

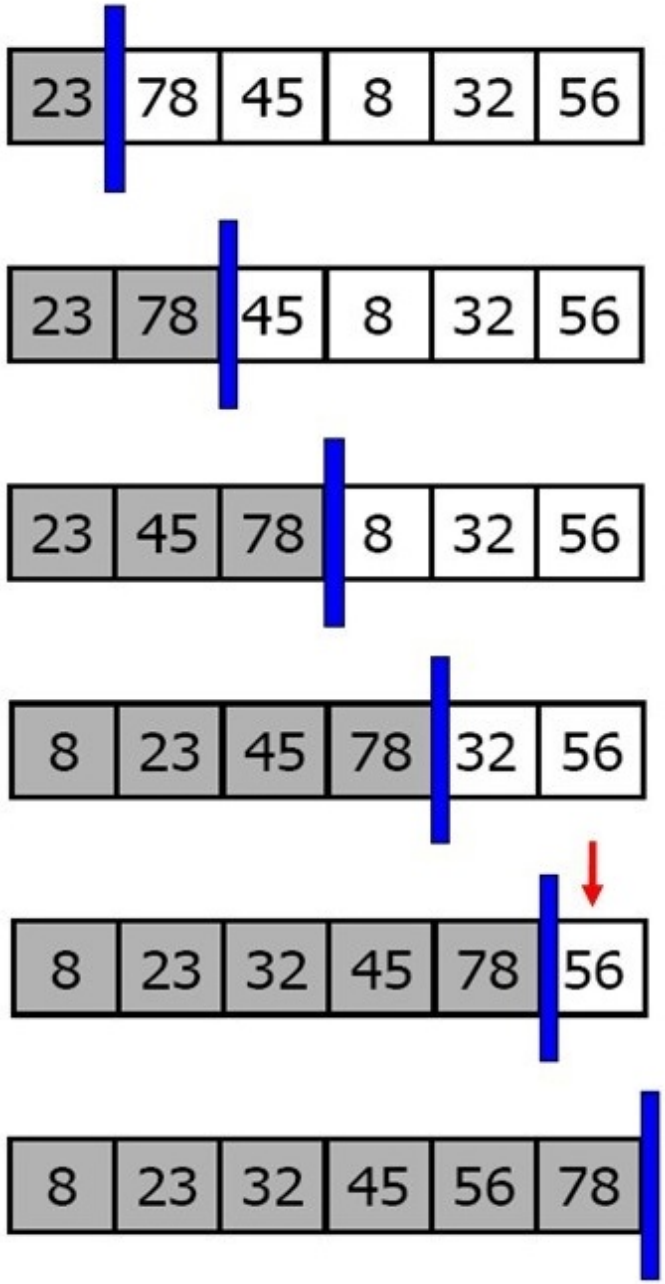


Luu Quang Huan, MsC

Straight Insertion Sort



Straight Insertion Sort



Luu Quang Huan, MsC

Straight Insertion Sort

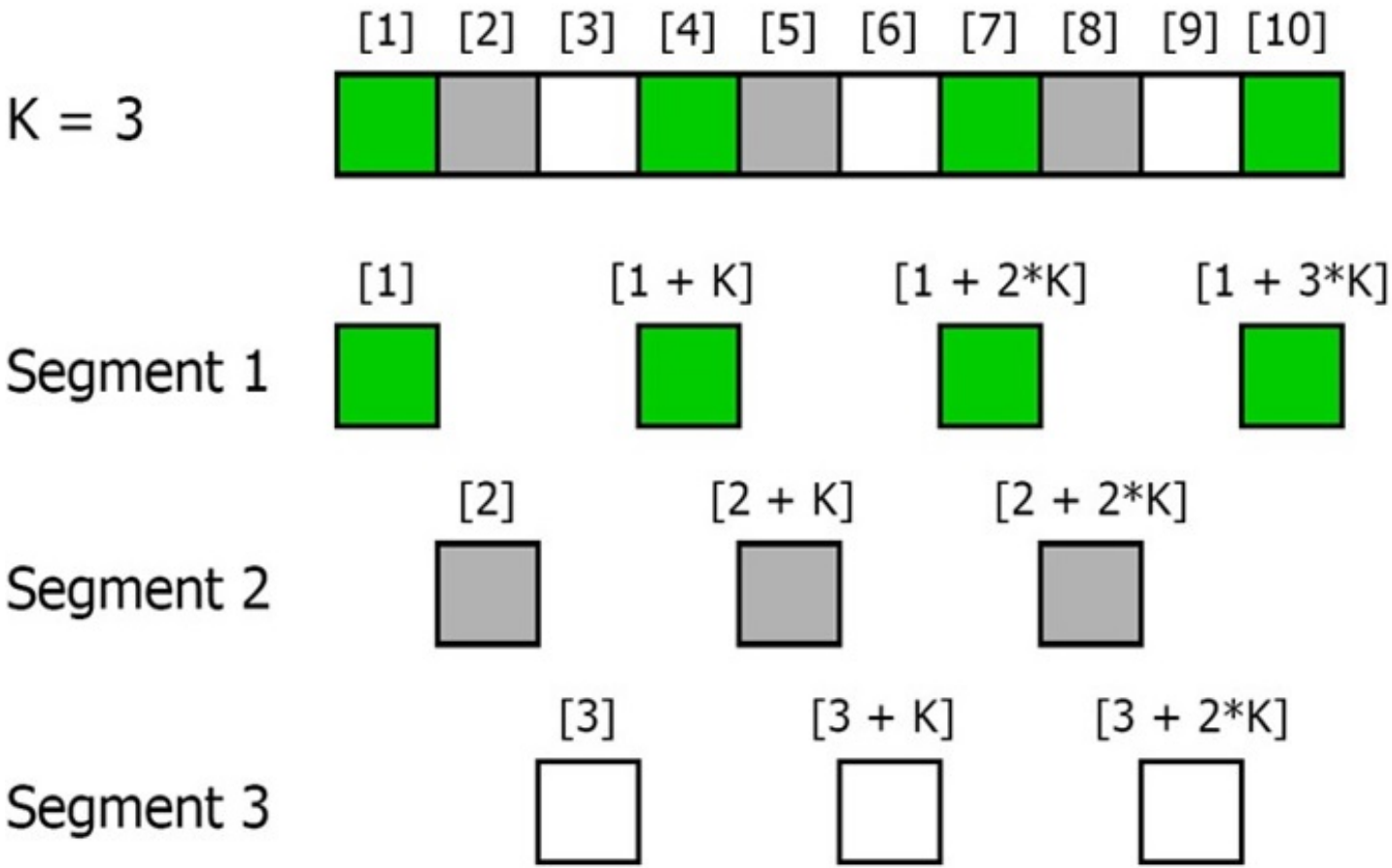
```
1  Algorithm InsertionSort()
2  Sorts the contiguous list using straight insertion
     sort.

3  if count > 1 then
4      current = 1
5      while current < count do
6          temp = data[current]
7          walker = current - 1
8          while walker >= 0 AND temp.key <
             data[walker].key do
9              data[walker+1] = data[walker]
10             walker = walker - 1
11         end
12         data[walker+1] = temp
13         current = current + 1
14     end
15 end
```

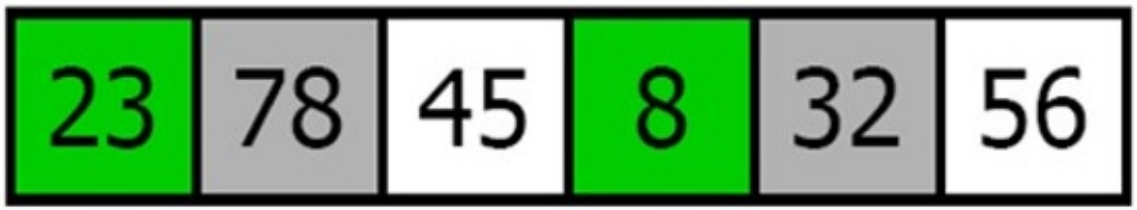

Shell Sort

- Named after its creator Donald L. Shell (1959).
- Given a list of N elements, the list is divided into K segments (K is called the increment).
- Each segment contains N/K or more elements.
- Segments are dispersed throughout the list.
- Also is called diminishing-increment sort.

Shell Sort

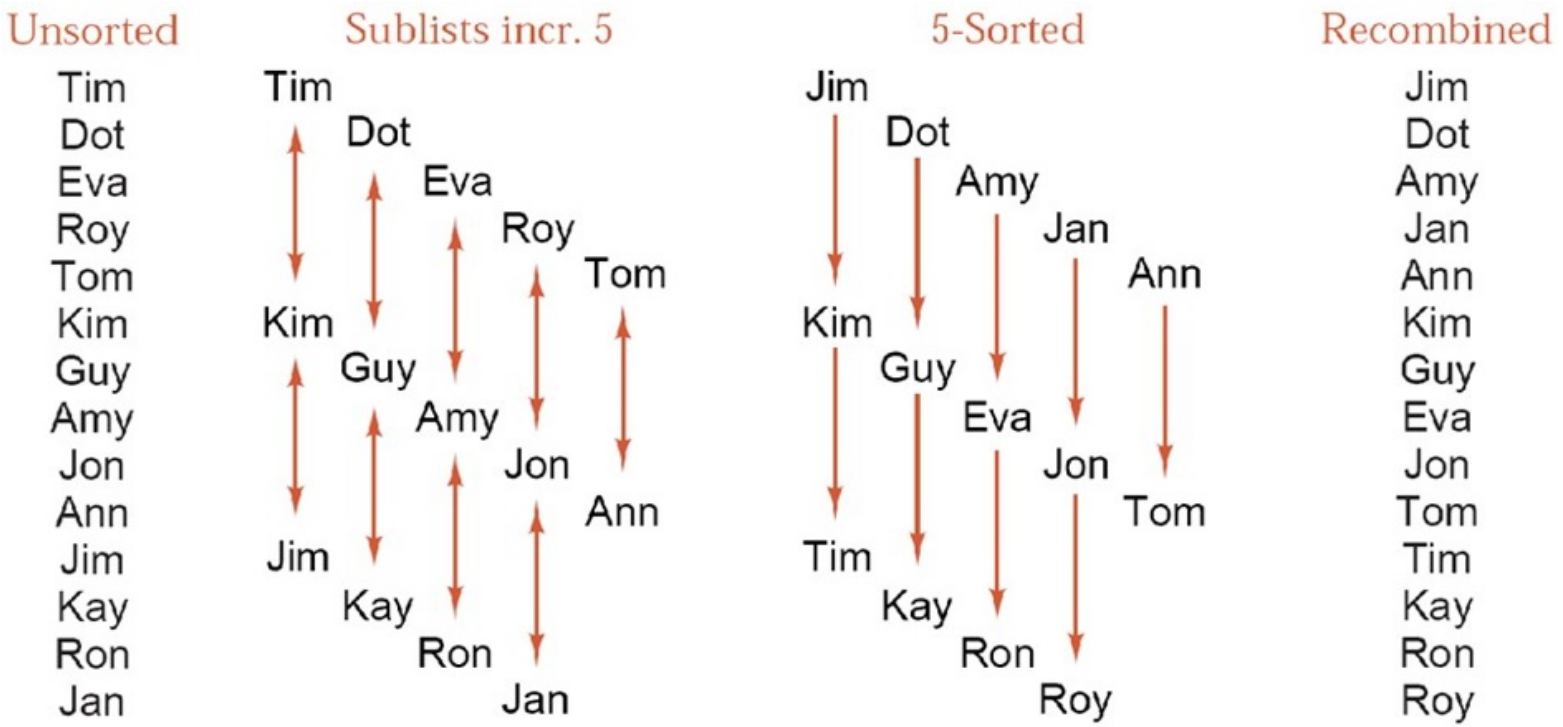


Shell Sort

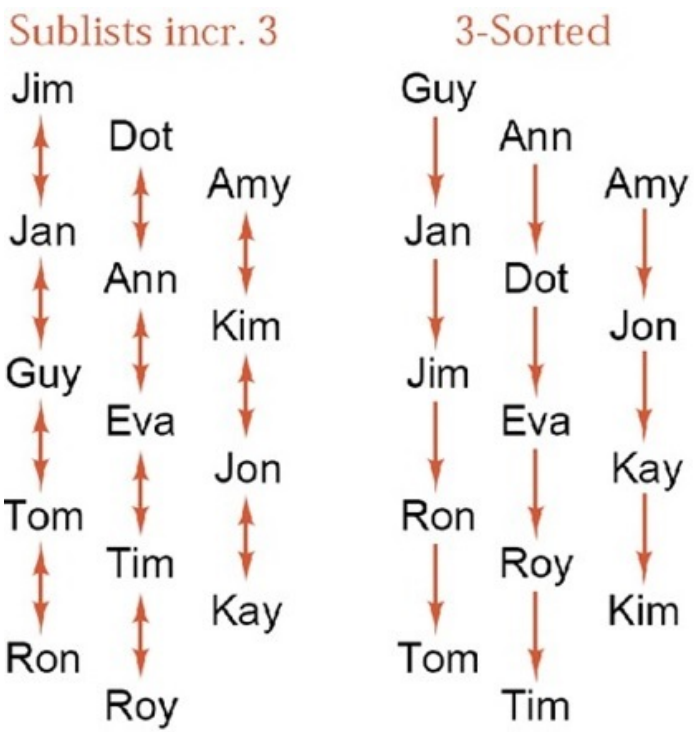


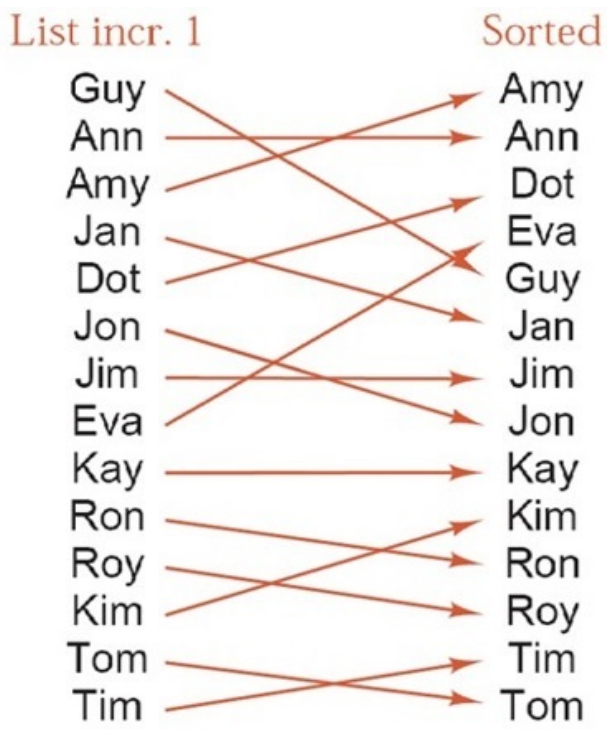
- For the value of K in each iteration, sort the K segments.
- After each iteration, K is reduced until it is 1 in the final iteration.

Example of Shell Sort



Example of Shell Sort





Choosing incremental values

- From more of the comparisons, it is better when we can receive more new information.
- Incremental values should not be multiples of each other, other wise, the same keys compared on one pass would be compared again at the next.
- The final incremental value must be 1.

Choosing incremental values

- Incremental values may be:
  $1, 4, 13, 40, 121, ...$
  $k_t = 1$
  $k_{i-1} = 3 * k_i + 1$
  $t = \lfloor \log_3 n \rfloor - 1$

- or:
  $1, 3, 7, 15, 31, ...$
  $k_t = 1$
  $k_{i-1} = 2 * k_i + 1$
  $t = \lfloor \log_2 n \rfloor - 1$

Shell Sort

```
1  Algorithm ShellSort()
2  Sorts the contiguous list using Shell sort.

3  k = first_incremental_value
4  while k >= 1 do
5  |   segment = 1
6  |   while segment <= k do
7  |   |   SortSegment(segment)
8  |   |   segment = segment + 1
9  |   end
10 |   k = next_incremental_value
11 end
12 End ShellSort
```


Shell Sort

```
1  Algorithm SortSegment(val segment <int>, val k
     <int>)
2  Sorts the segment beginning at segment using
     insertion sort, step between elements in the
     segment is k.

3  current = segment + k
4  while current < count do
5  |   temp = data[current]
6  |   walker = current - k
7  |   while walker >=0 AND temp.key <
   |     data[walker].key do
8  |   |   data[walker + k] = data[walker]
9  |   |   walker = walker - k
10 |   end
11 |   data[walker + k] = temp
12 |   current = current + k
13 end
```


Luu Quang Huan, MsC

Insertion Sort Efficiency

- Straight insertion sort:
  $f(n) = n(n+1)/2 = O(n^2)$
- Shell sort:
  $O(n^{1.25})$ (Empirical study)

Selection Sort

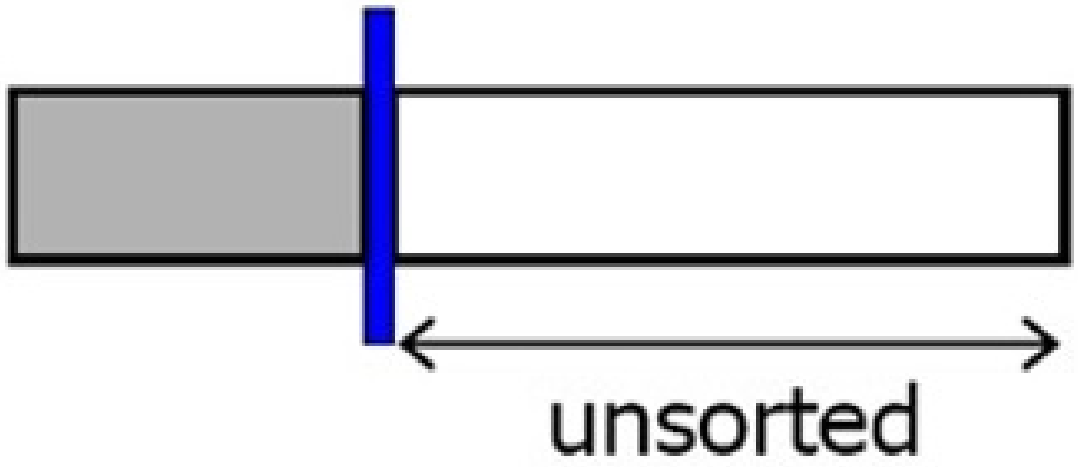
Selection Sort

In each pass, the smallest/largest item is selected and placed in a sorted list.

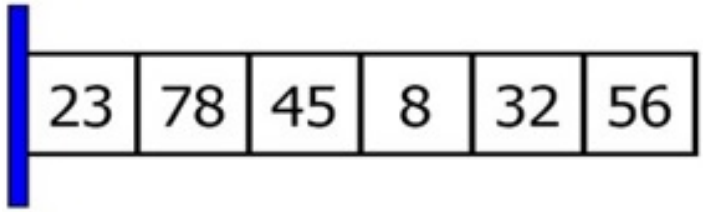
Luu Quang Huan, MsC

Straight Selection Sort

- The list is divided into two parts: sorted and unsorted.
- In each pass, in the unsorted sublist, the smallest element is selected and exchanged with the first element.

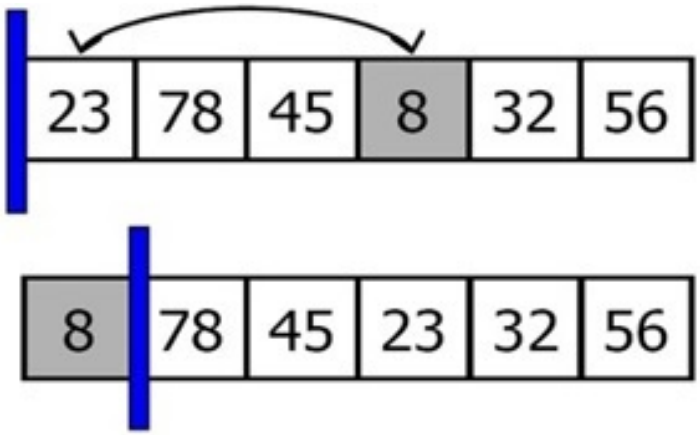


Straight Selection Sort

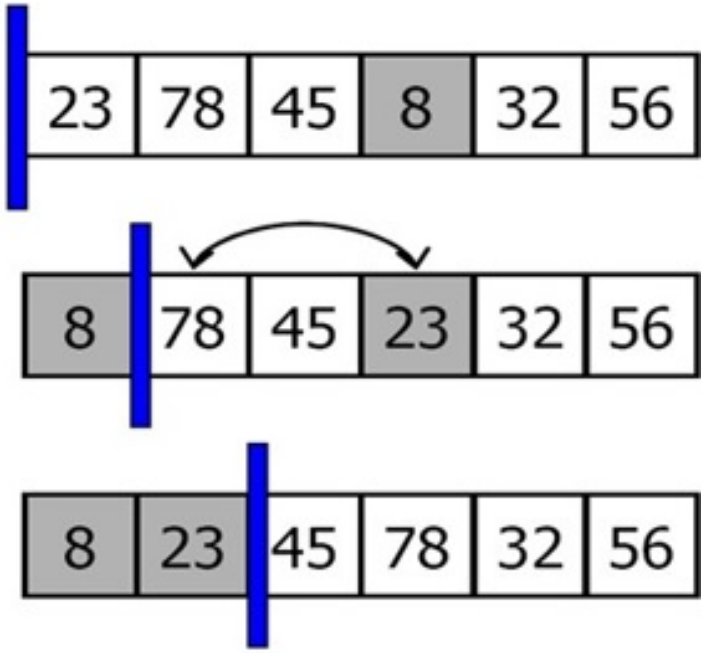


Luu Quang Huan, MsC

Straight Selection Sort

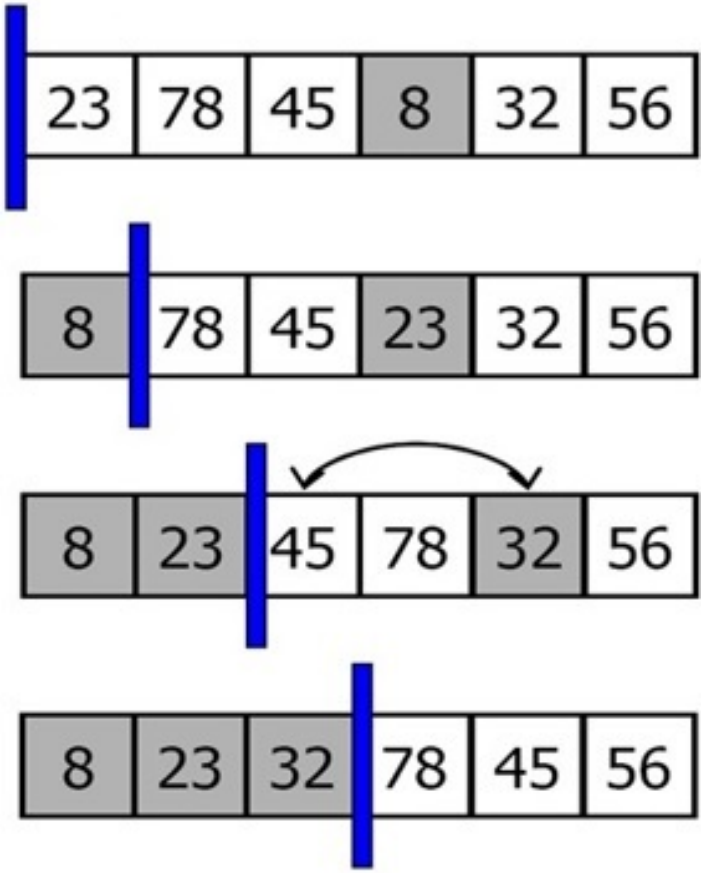


Straight Selection Sort



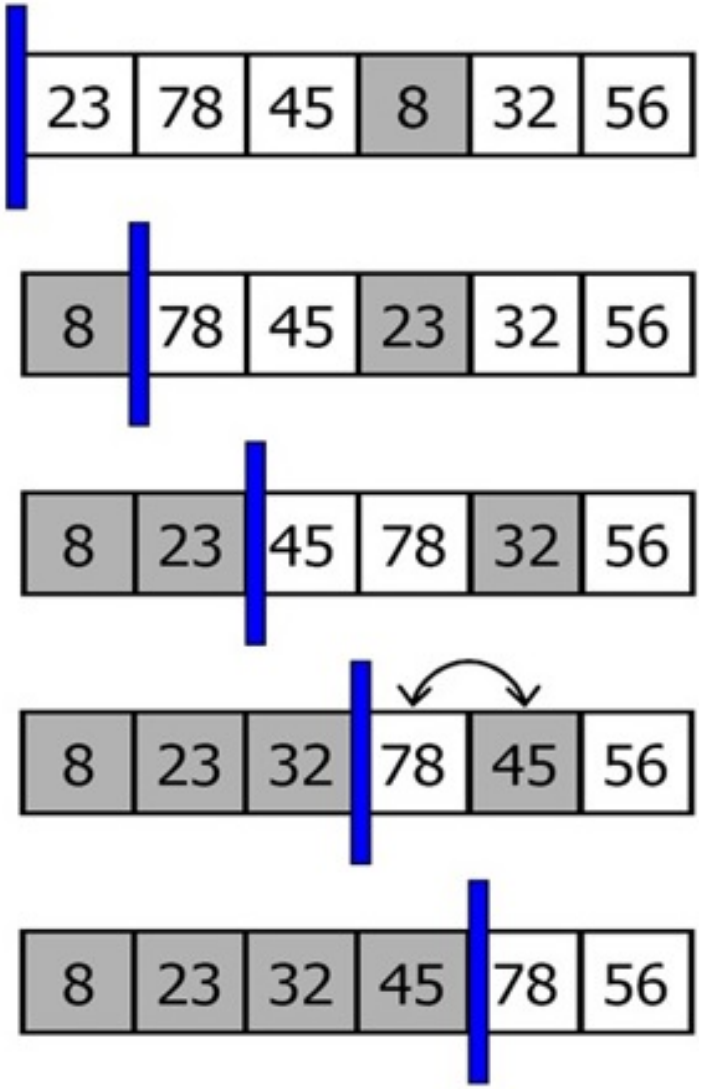
Luu Quang Huan, MsC

Straight Selection Sort



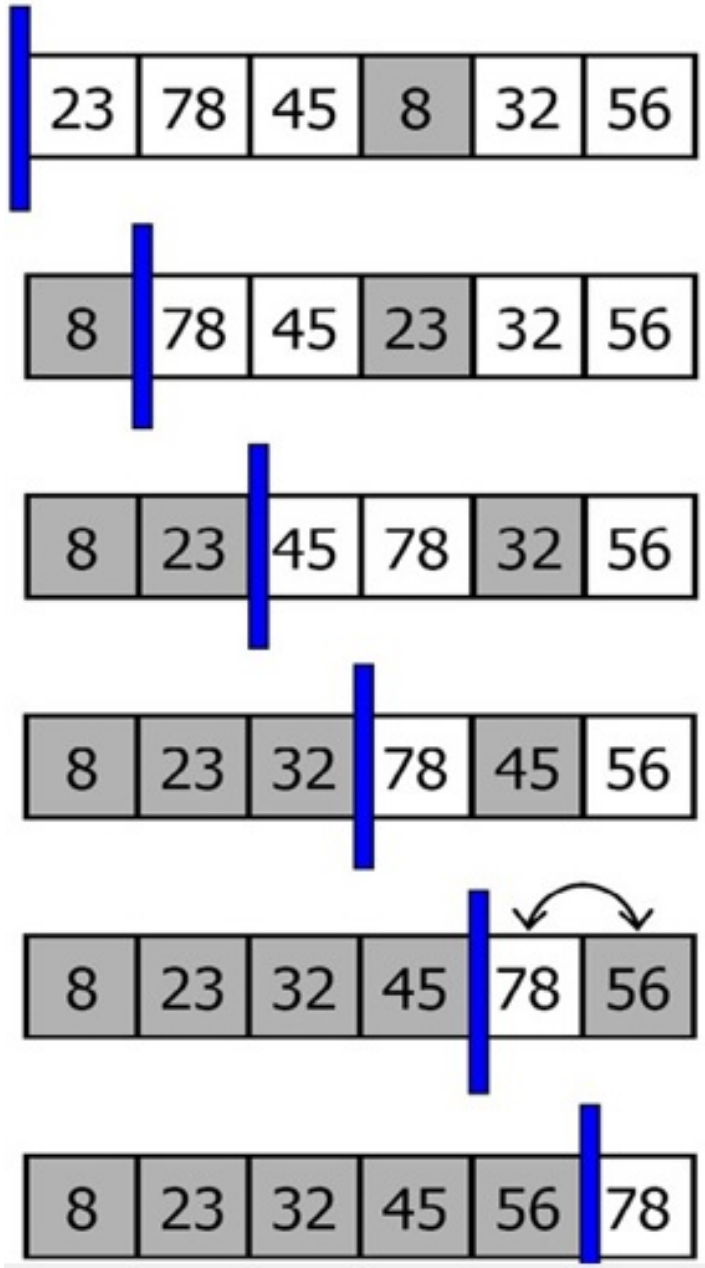
Luu Quang Huan, MsC

Straight Selection Sort



Luu Quang Huan, MsC

Straight Selection Sort



Luu Quang Huan, MsC

Straight Selection Sort

```
1  Algorithm SelectionSort()
2  Sorts the contiguous list using straight selection
     sort.

3  current = 0
4  while current < count - 1 do
5  |    smallest = current
6  |    walker = current + 1
7  |    while walker < count do
8  |    |    if data [walker].key < data [smallest].key
   |    |      then
9  |    |    |    smallest = walker
10 |    |    end
11 |    |    walker = walker + 1
12 |    end
13 |    swap(current, smallest)
14 |    current = current + 1
15 end
```


Luu Quang Huan, MsC

Selection Sort Efficiency

- Straight selection sort: $O(n^2)$

Exchange Sort

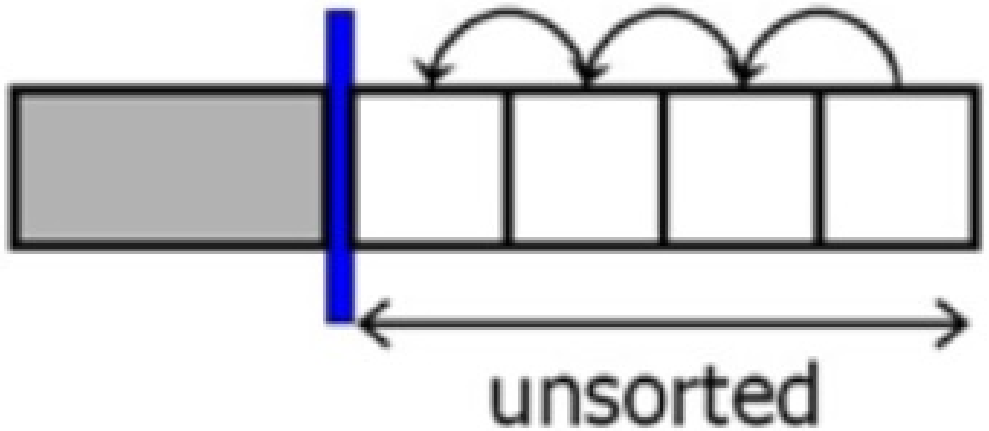
Exchange Sort

- In each pass, elements that are out of order are exchanged, until the entire list is sorted.
- Exchange is extensively used.

Luu Quang Huan, MsC

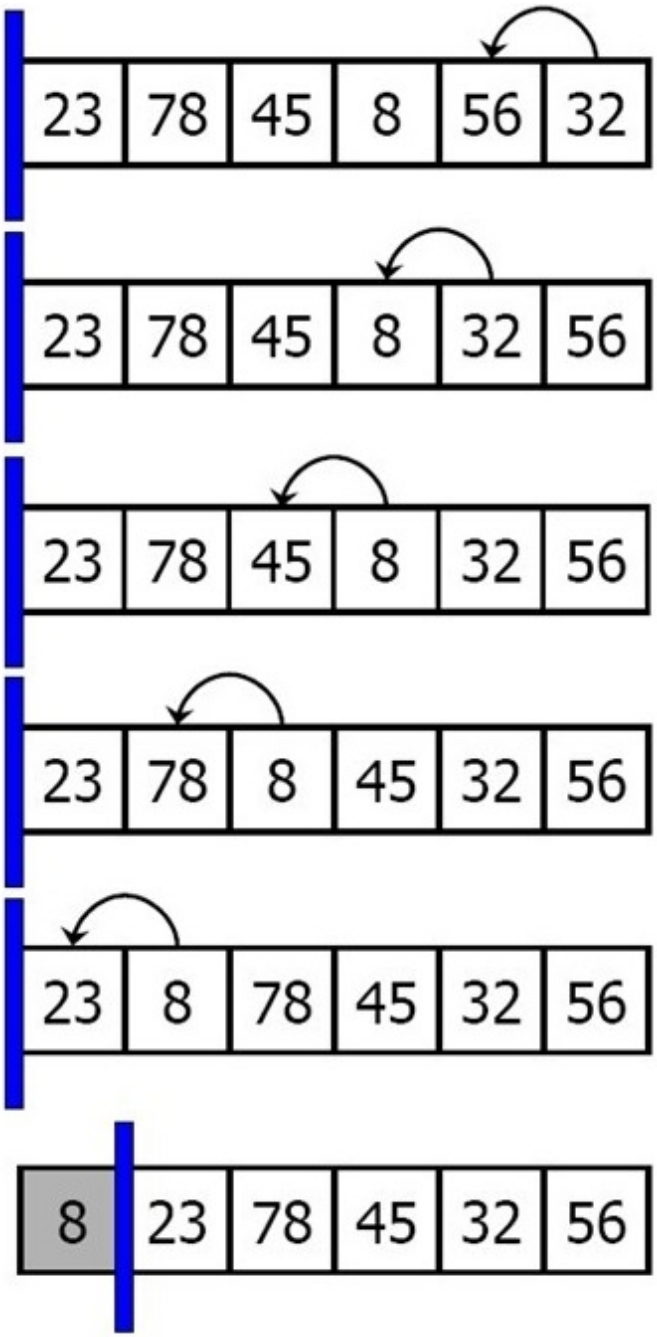
Bubble Sort

- The list is divided into two parts: sorted and unsorted.
- In each pass, the smallest element is bubbled from the unsorted sublist and moved to the sorted sublist.



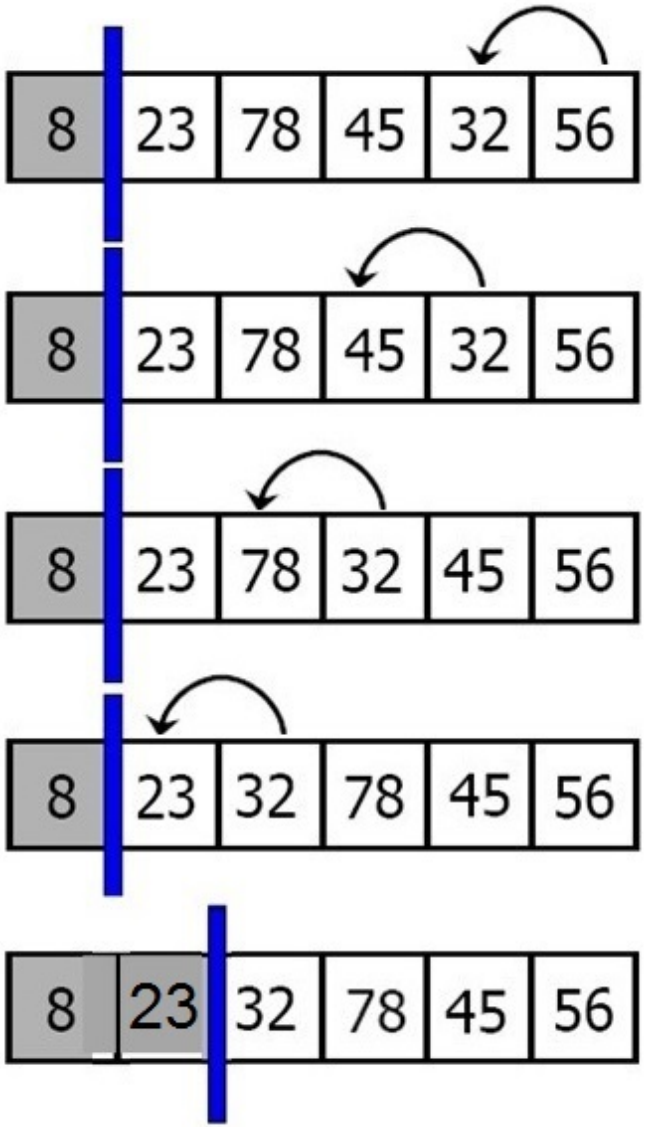
Luu Quang Huan, MsC

Bubble Sort



Luu Quang Huan, MsC

Bubble Sort



Bubble Sort

```
1  Algorithm BubbleSort()
2  Sorts the contiguous list using bubble sort.

3  current = 0
4  flag = False
5  while current < count AND flag = False do
6  |   walker = count - 1
7  |   flag = True
8  |   while walker > current do
9  |   |   if data [walker].key < data [walker-1].key
   |   |     then
10 |   |   |   flag = False
11 |   |   |   swap(walker, walker - 1)
12 |   |   end
13 |   |   walker = walker - 1
14 |   end
15 |   current = current + 1
16 end
```


Exchange Sort Efficiency

- Bubble sort:
  $f(n) = n(n+1)/2 = O(n^2)$

Devide-and-Conquer

Devide-and-Conquer Sort

```
1 Algorithm DevideAndConquer()
2 if the list has length > 1 then
3 |   partition the list into lowlist and
  |     highlist
4 |   lowlist.DevideAndConquer()
5 |   highlist.DevideAndConquer()
6 |   combine(lowlist, highlist)
7 end
8 End DevideAndConquer
```


Devide-and-Conquer Sort

	Partition	Combine
Merge Sort	easy	hard
Quick Sort	hard	easy

Quick Sort

```
1 Algorithm QuickSort()
2 Sorts the contiguous list using quick sort.

3 recursiveQuickSort(0, count - 1)
4 End QuickSort
```

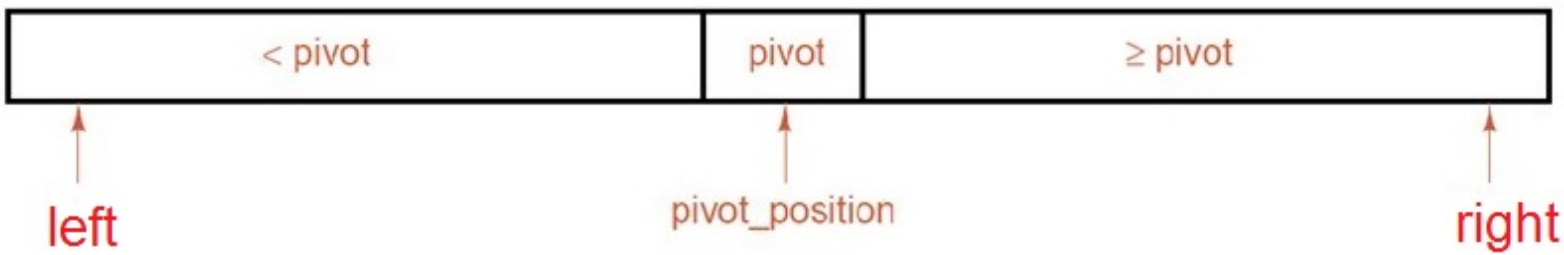

Quick Sort

```
1 Algorithm recursiveQuickSort(val left
    <int>, val right <int>)
2 Sorts the contiguous list using quick sort.
3 Pre: left and right are valid positions
    in the list
4 Post: list sorted

5 if left < right then
6 |   pivot_position = Partition(left, right)
7 |   recursiveQuickSort(left,
  |     pivot_position - 1)
8 |   recursiveQuickSort(pivot_position +
  |     1, right)
9 end
```


Quick Sort

Given a pivot value, the partition rearranges the entries in the list as the following figure:

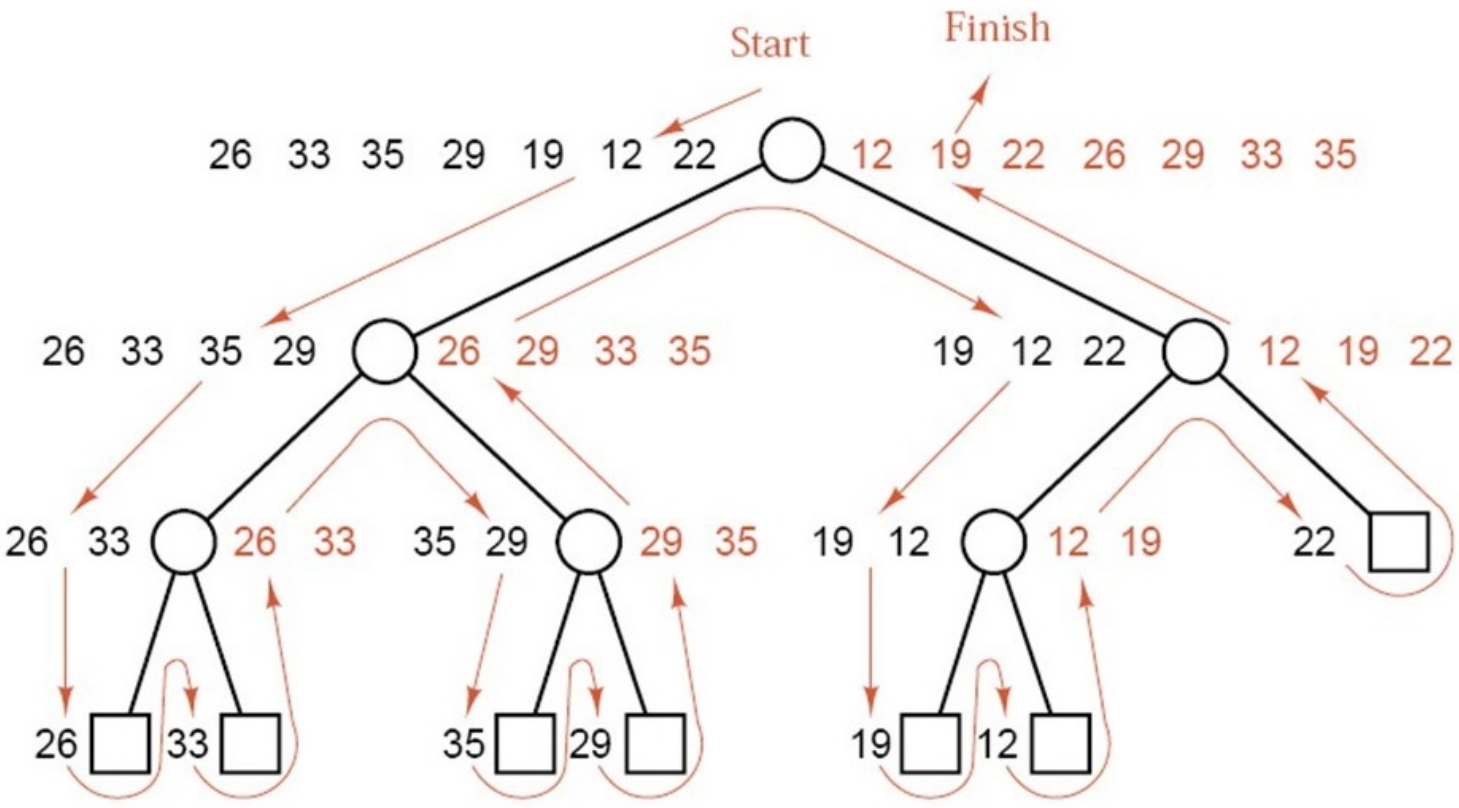


Luu Quang Huan, MsC

Quick Sort Efficiency

- Quick sort: $O(nlog_2n)$

Merge Sort



Merge Sort

```
1 Algorithm MergeSort()
2 Sorts the linked list using merge sort.

3 recursiveMergeSort(head)
4 End MergeSort
```


Merge Sort

```
1 Algorithm recursiveMergeSort(ref sublist <pointer>)
2 Sorts the linked list using recursive merge sort.

3 if sublist is not NULL AND sublist->link is not NULL then
4 |   Divide(sublist, second_list)
5 |   recursiveMergeSort(sublist)
6 |   recursiveMergeSort(second_list)
7 |   Merge(sublist, second_list)
8 end
9 End recursiveMergeSort
```

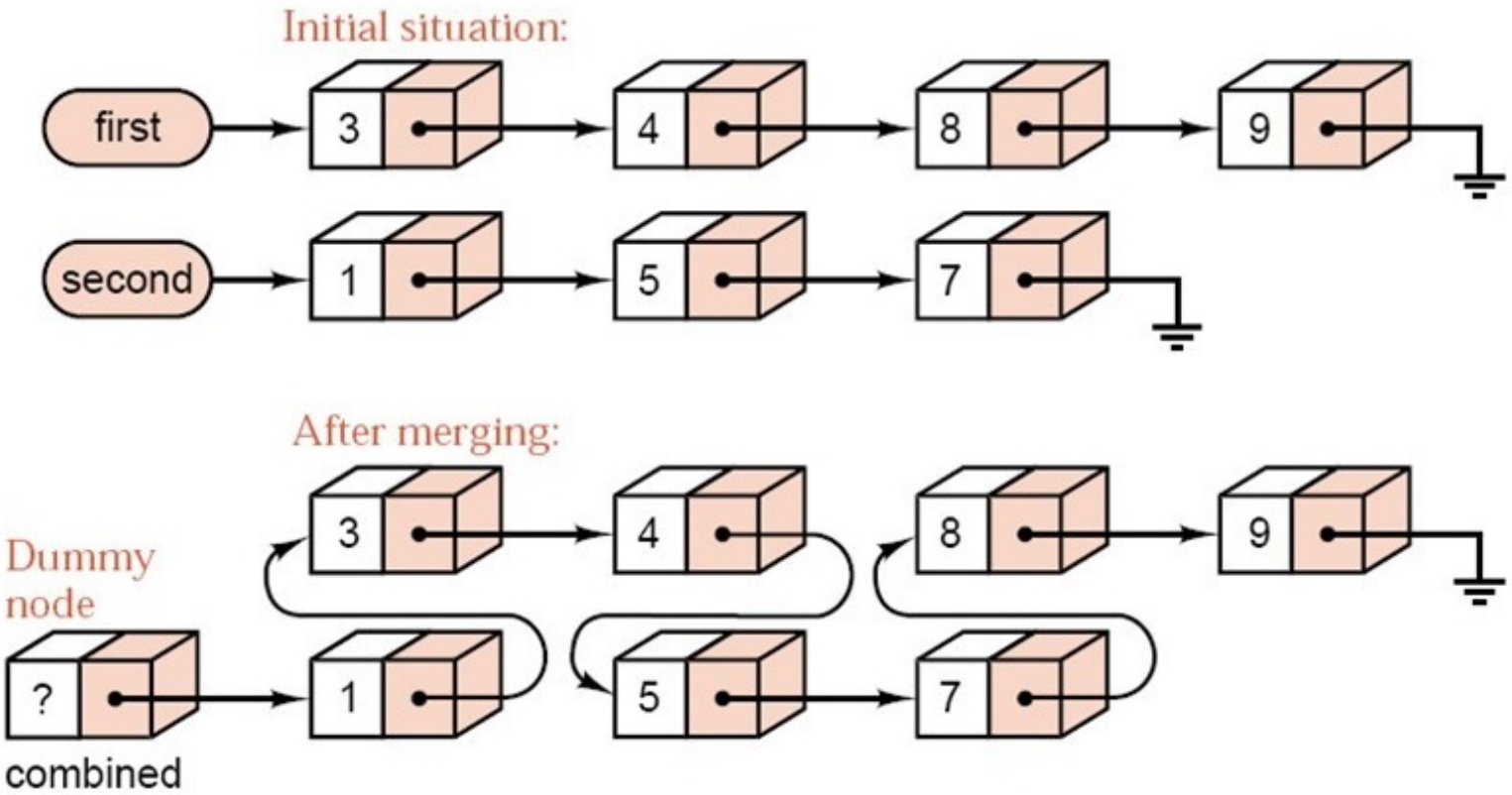

Luu Quang Huan, MsC

Merge Sort

```
1  Algorithm Divide(val sublist <pointer>, ref
     second_list <pointer>)
2  Divides the list into two halves.

3  midpoint = sublist
4  position = sublist->link
5  while position is not NULL do
6  |    position = position->link
7  |    if position is not NULL then
8  |    |    midpoint = midpoint->link
9  |    |    position = position->link
10 |    end
11 end
12 second_list = midpoint->link
13 midpoint->link = NULL
14 End Divide
```


Merge two sublists



Luu Quang Huan, MsC

Merge two sublists

```
1  Algorithm Merge(ref first <pointer>, ref second
     <pointer>)
2  Merges two sorted lists into a sorted list.

3  lastSorted = address of combined
4  while first is not NULL AND second is not NULL
     do
5  |    if first->data.key <= second->data.key then
6  |    |    lastSorted->link = first
7  |    |    lastSorted = first
8  |    |    first = first->link
9  |    else
10 |    |    lastSorted->link = second
11 |    |    lastSorted = second
12 |    |    second = second->link
13 |    end
14 end
```


Merge two sublists

```
1 // ...

2 if first is NULL then
3 |   lastSorted->link = second
4 |   second = NULL
5 else
6 |   lastSorted->link = first
7 end
8 first = combined.link
9 End Merge
```


THANK YOU.