Data Structure and Algorithms

Deepali Londhe
Information Technology,
PICT, Pune

1

Agenda

2

- Searching and sorting
- Concept of internal and external sorting
- Sort stability
- Sorting methods: Bubble, insertion, Quick, Merge, shell and comparison of all sorting methods.
- Case Studies Set Operation, String Operation
- Fibonacci Series.

DSA Unit 1.4 Insersion Sort

2

Insertion Sort

3

- Finding the element’s proper place
- Making room for the inserted element
- (by shifting over other elements)
- Inserting the element

- Insertion sort works the same way as arranging your hand when playing cards.
- Out of the pile of unsorted cards that were dealt to you, you pick up a card and place it in your hand in the correct position relative to the cards you’re already holding.

DSA Unit 1.4 Insersion Sort

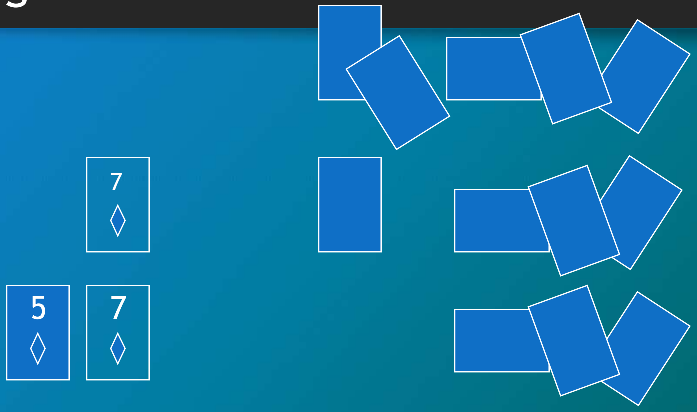
3

Arranging Your Hand

4

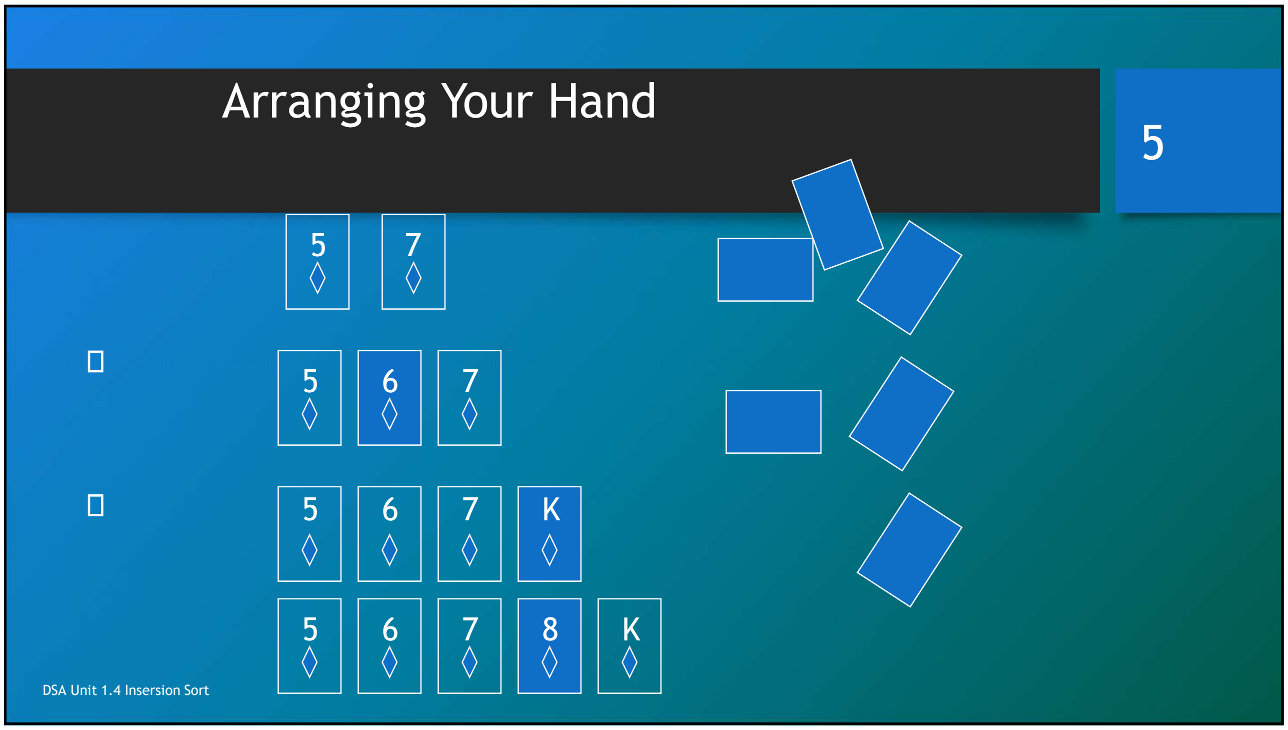
□

□



DSA Unit 1.4 Insersion Sort

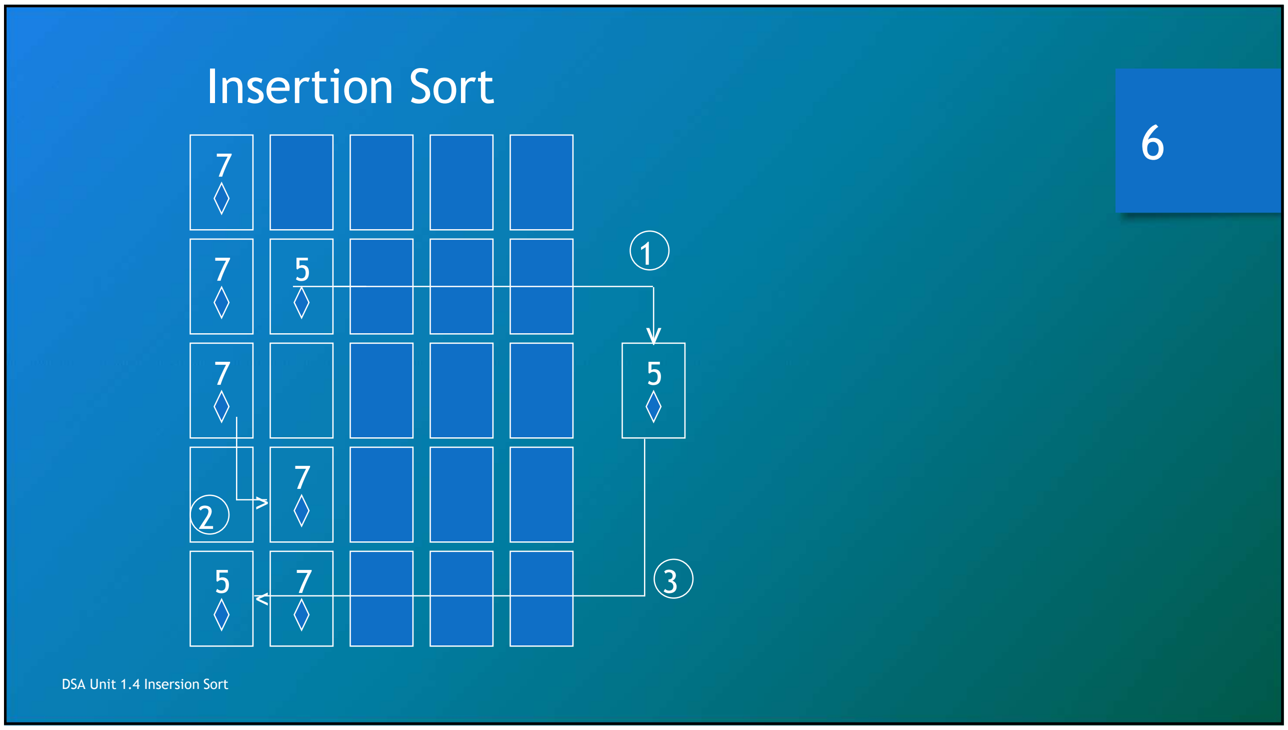
4



Arranging Your Hand

DSA Unit 1.4 Insersion Sort

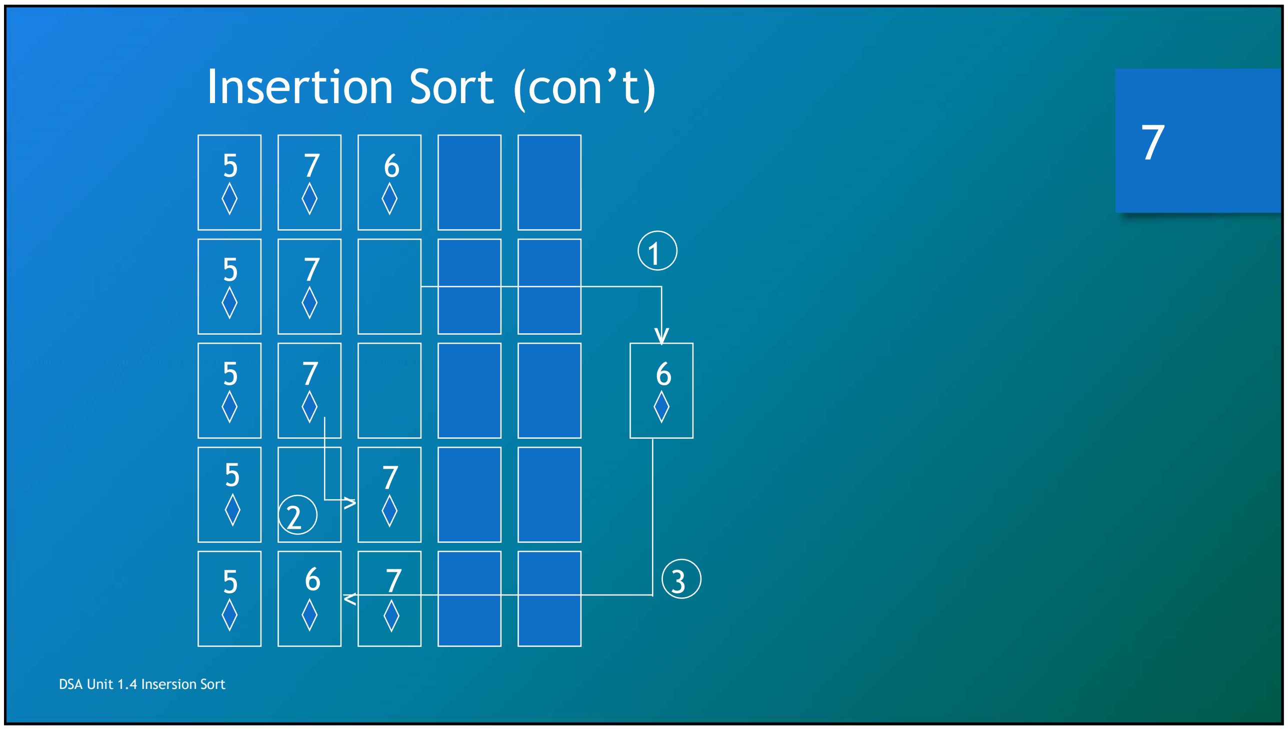
5



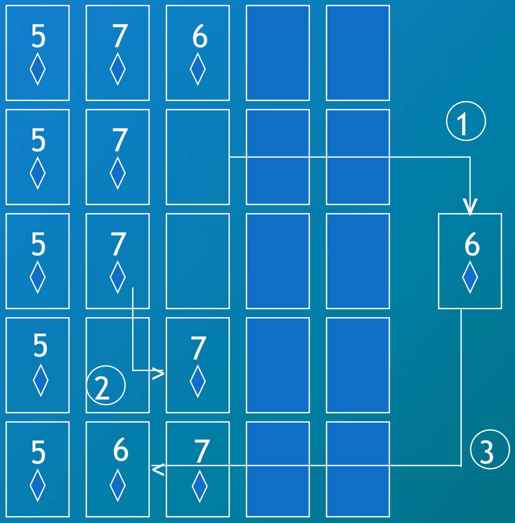
Insertion Sort

DSA Unit 1.4 Insersion Sort

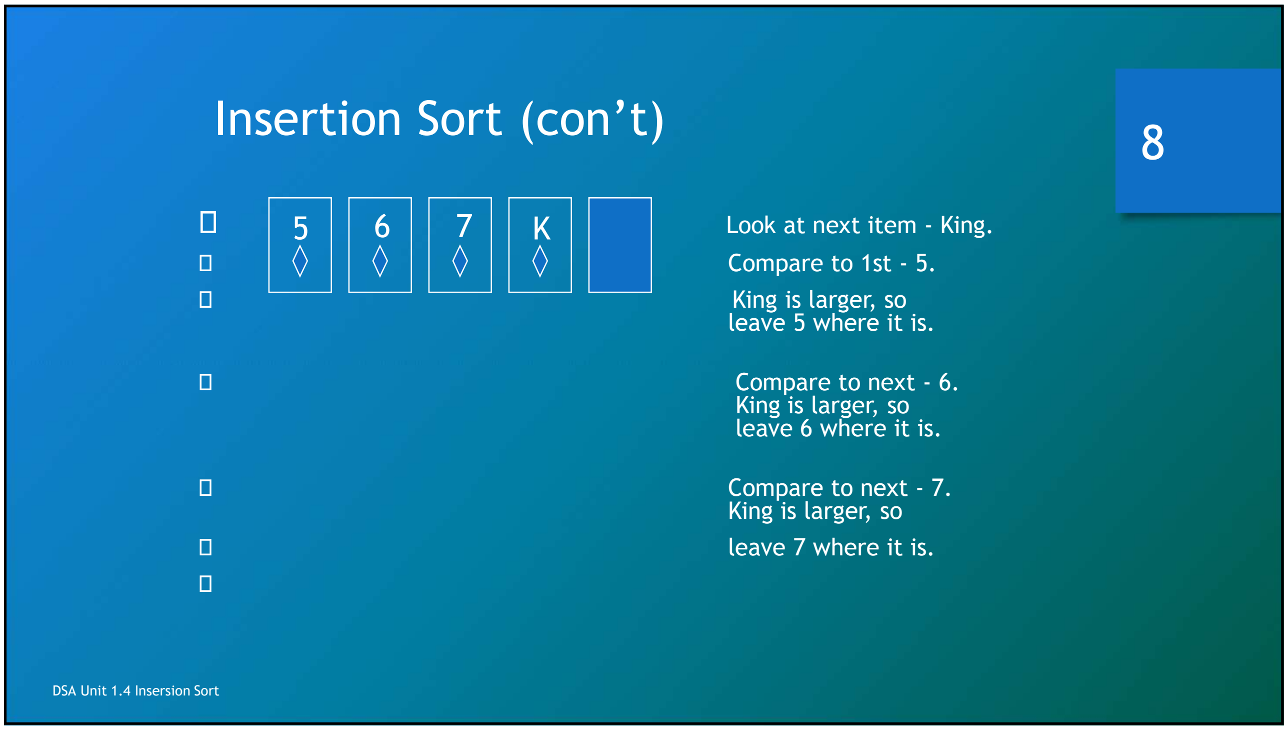
6



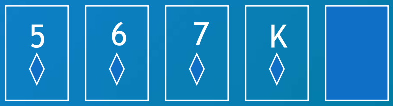
Insertion Sort (con’t)



7

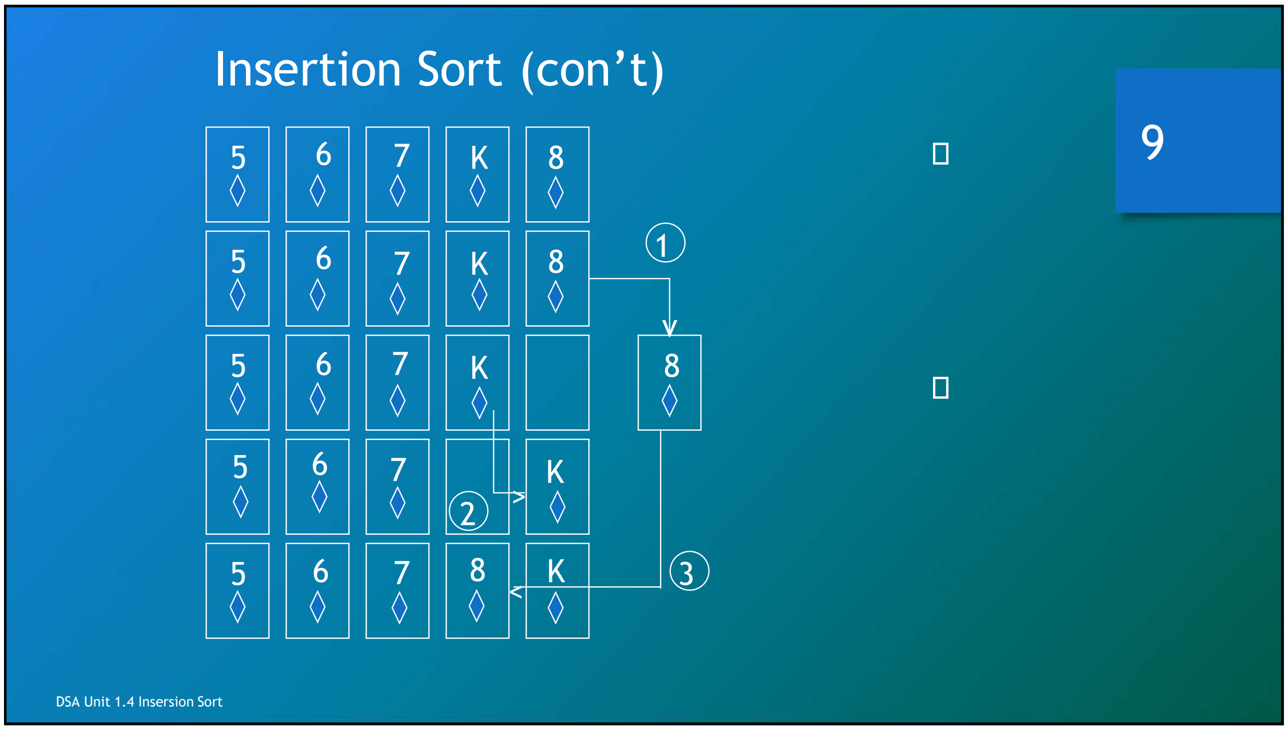


Insertion Sort (con’t)



- Look at next item - King.
- Compare to 1st - 5.
- King is larger, so leave 5 where it is.
- Compare to next - 6. King is larger, so leave 6 where it is.
- Compare to next - 7. King is larger, so
- leave 7 where it is.
-

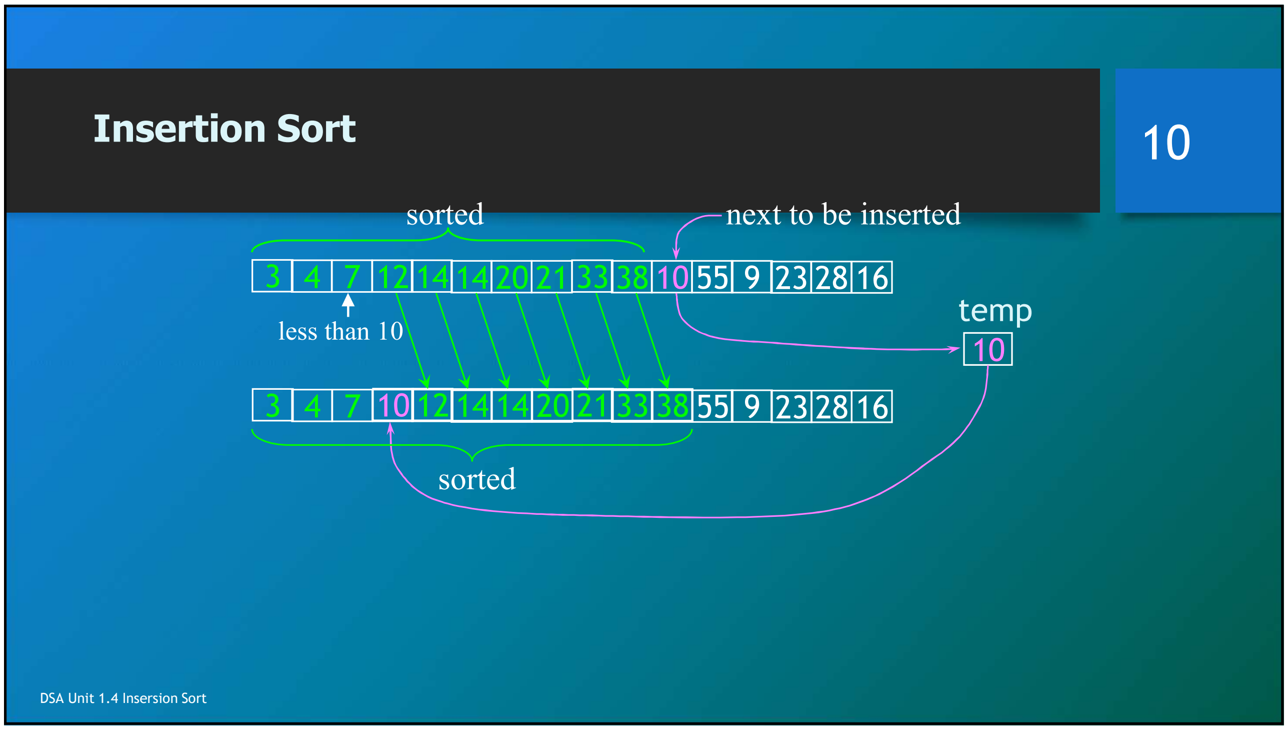
8



Insertion Sort (con't)

DSA Unit 1.4 Insersion Sort

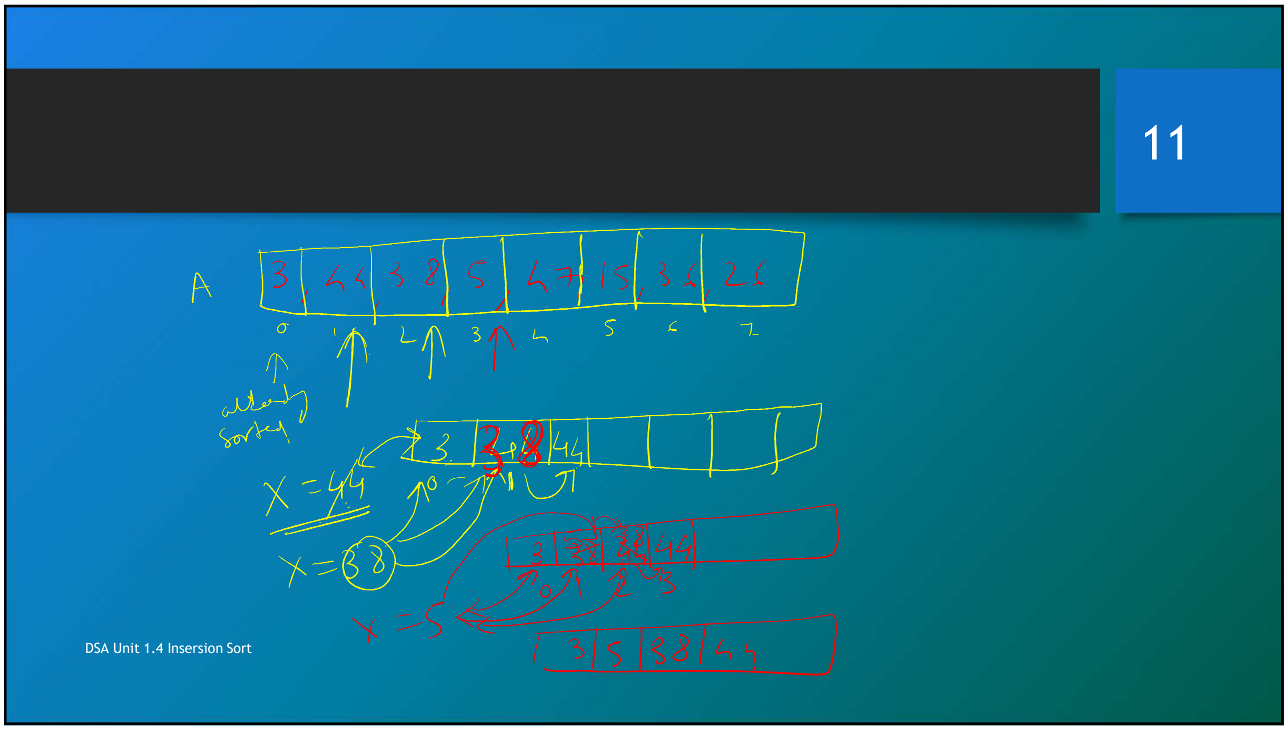
9



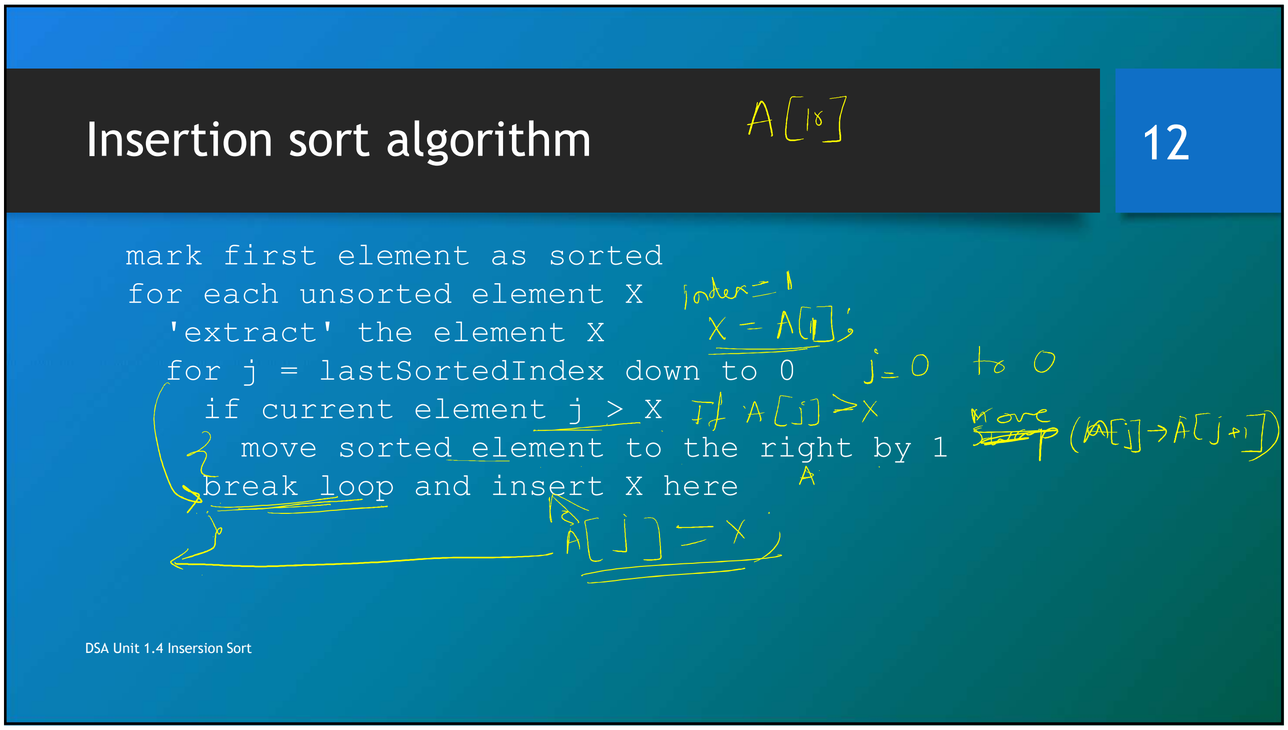
Insertion Sort

DSA Unit 1.4 Insersion Sort

10



11



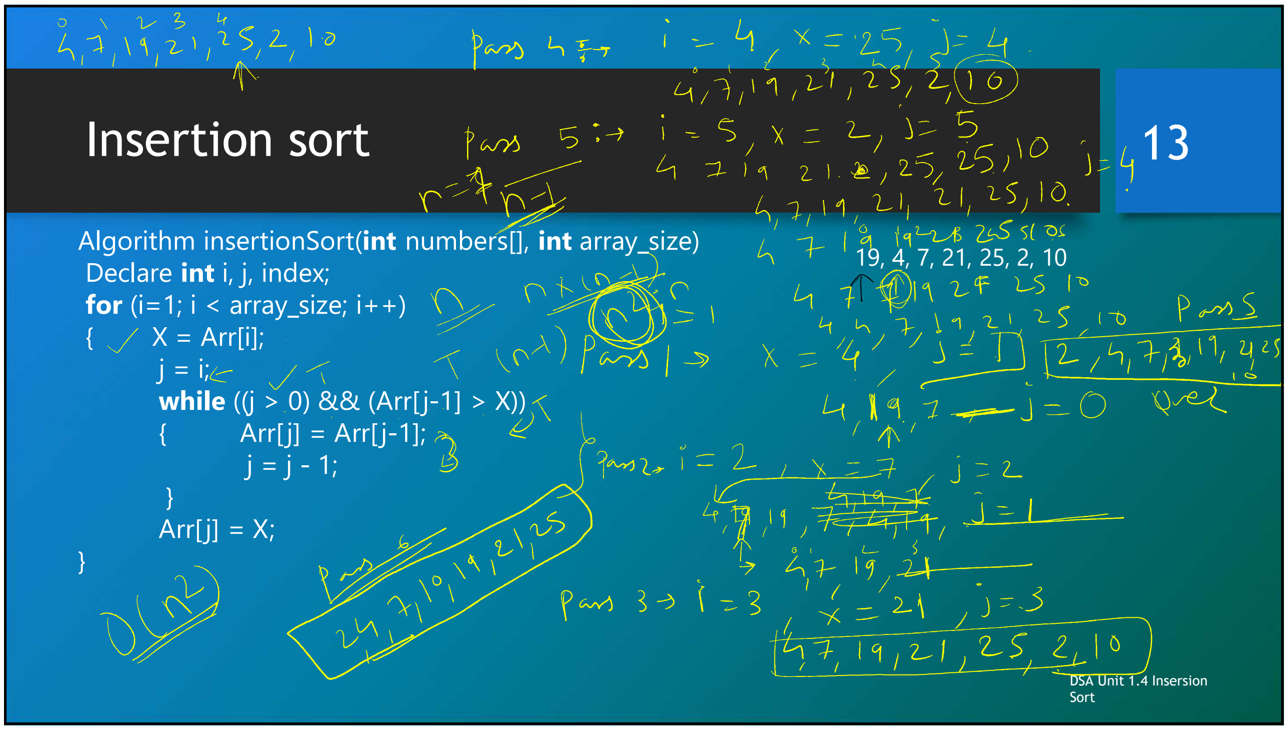
Insertion sort algorithm

A[18]

```
mark first element as sorted
for each unsorted element X                index=1
  'extract' the element X                  X = A[1];
  for j = lastSortedIndex down to 0        j=0 to 0
    if current element j > X               If A[j] > X
      move sorted element to the right by 1    ~~swap~~ move (A[j] → A[j+1])
    break loop and insert X here
                                           A[j] = X;
```

DSA Unit 1.4 Insersion Sort

12



0 1 2 3 4
4,7,19,21,25,2,10

Pass 4 :→ i = 4, x = 25, j = 4
4,7,19,21,25,2,10

Insertion sort

Pass 5 :→ i = 5, x = 2, j = 5
4 7 19 21 ~~2~~, 25, 25, 10 j = 4
4,7,19,21,21,25,10
n = ~~7~~ n-1

```
Algorithm insertionSort(int numbers[], int array_size)
 Declare int i, j, index;
 for (i=1; i < array_size; i++)
 {    X = Arr[i];
         j = i;
         while ((j > 0) && (Arr[j-1] > X))
         {        Arr[j] = Arr[j-1];
                  j = j - 1;
          }
         Arr[j] = X;
}
```

19, 4, 7, 21, 25, 2, 10

n   n×(n-1) $\frac{n(n-1)}{2}$ T (n-1)

Pass 1 → x = 4, j = 1 Pass 5
4 7 19 21 25 10
4 4 7 19 21 25 10
2,4,7,10,19,21,25
4, 19, 7 — j = 0 over

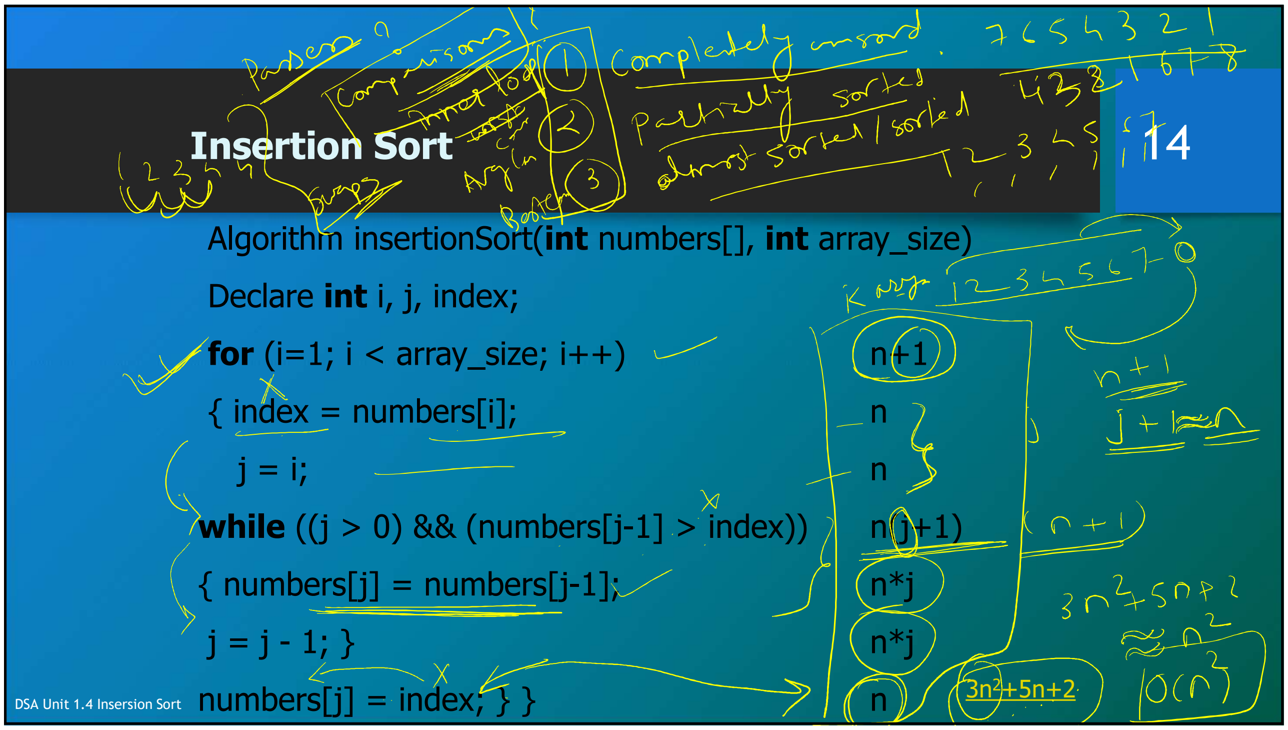
Pass 2 → i = 2, x = 7, j = 2
4, 19, 19, ~~7, 4, 19~~, j = 1
4, 7, 19, 21

Pass 3 → i = 3, x = 21, j = 3
4, 7, 19, 21, 25, 2, 10

Pass 6
2, 4, 7, 10, 19, 21, 25

O(n²)

13



Passes   Comparisons   Inner loop   Swaps   (1) Best case   (2) Avg case   (3) Worst
Completely unsorted   7 6 5 4 3 2 1
partially sorted   4 3 8 1 6 7 8
almost sorted / sorted   1 2 3 4 5
1 2 3 4 4

Insertion Sort

Algorithm insertionSort(**int** numbers[], **int** array_size)	
Declare **int** i, j, index;	
for (i=1; i < array_size; i++)	n+1
{ index = numbers[i];	n
j = i;	n
while ((j > 0) && (numbers[j-1] > index))	n(j+1)
{ numbers[j] = numbers[j-1];	n*j
j = j - 1; }	n*j
numbers[j] = index; } }	n

3n²+5n+2

K array 1 2 3 4 5 6 7 0
n+1
j+1 = n
(n+1)
3n²+5n+2 ≈ n²
O(n²)

14

Analysis of insertion sort

15

- We run once through the outer loop, inserting each of n elements; this is a factor of n
- On average, there are n/2 elements already sorted
  - The inner loop looks at (and moves) half of these
  - This gives a second factor of n/4
- Hence, the time required for an insertion sort of an array of n elements is proportional to $n^2/4$
- Discarding constants, we find that insertion sort is $O(n^2)$

DSA Unit 1.4 Insersion Sort

15

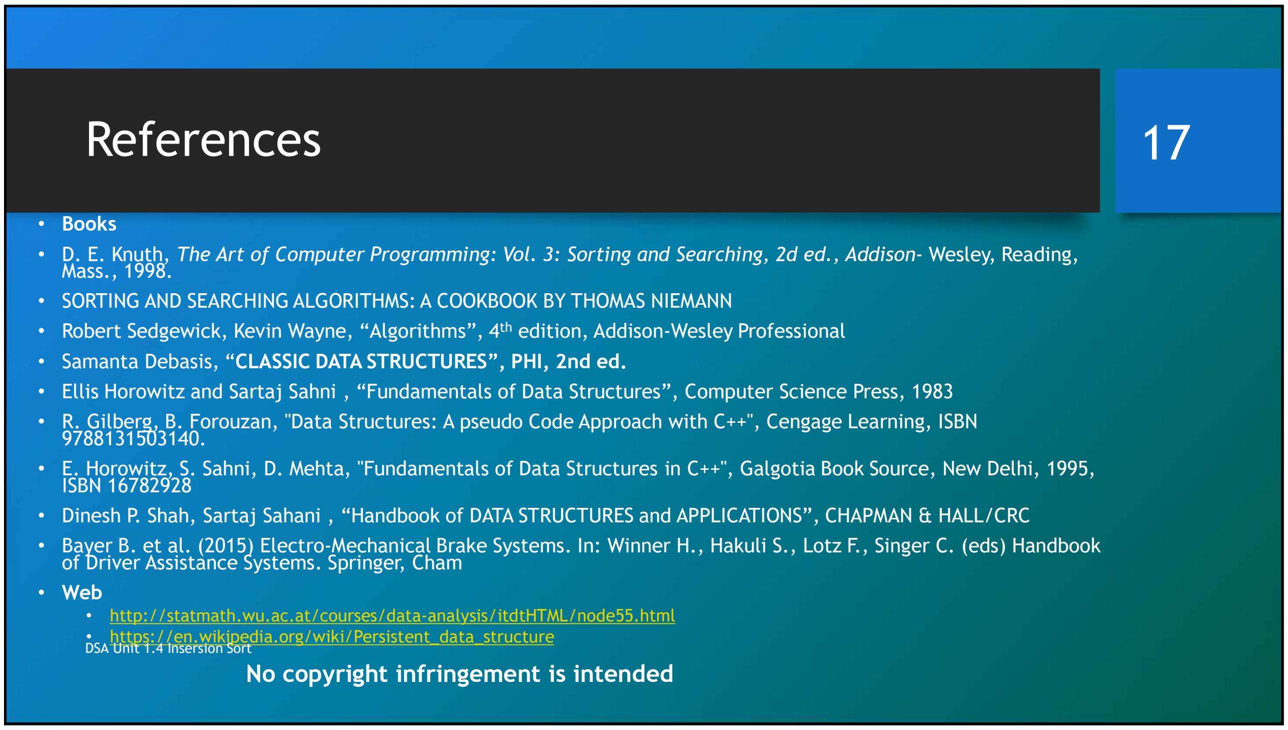
Insertion Sort Summary

16

• Worst Case Complexity	$O(n^2)$
• Comparisons -	
• Swap	$O(n^2)$
• Best Case Complexity	$O(n)$
• Comparisons	
• Swap	$O(1)$
• Average Case Complexity	
• Comparisons	$O(n^2)$
• Swap	$O(n^2)$
• Is it Stable?	Yes

DSA Unit 1.4 Insersion Sort

16



References

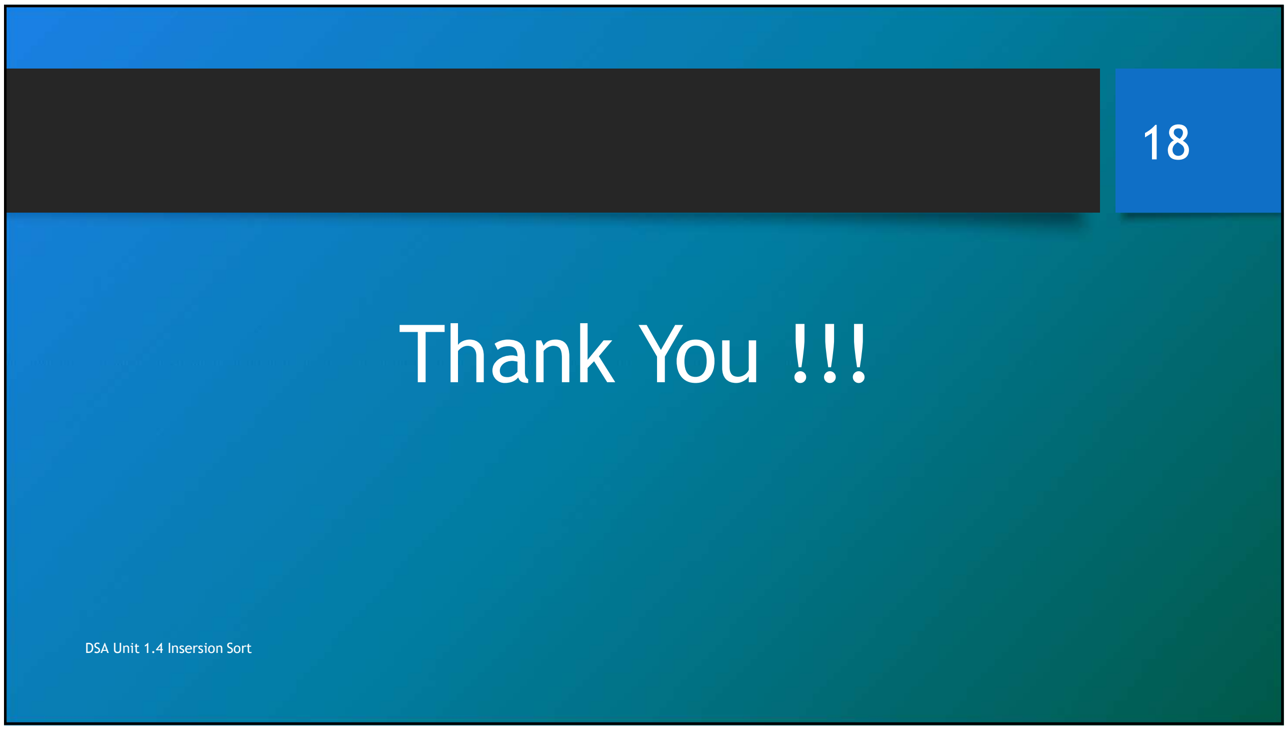
17

- **Books**
- D. E. Knuth, *The Art of Computer Programming: Vol. 3: Sorting and Searching, 2d ed., Addison-* Wesley, Reading, Mass., 1998.
- SORTING AND SEARCHING ALGORITHMS: A COOKBOOK BY THOMAS NIEMANN
- Robert Sedgewick, Kevin Wayne, “Algorithms”, 4th edition, Addison-Wesley Professional
- Samanta Debasis, **“CLASSIC DATA STRUCTURES”, PHI, 2nd ed.**
- Ellis Horowitz and Sartaj Sahni , “Fundamentals of Data Structures”, Computer Science Press, 1983
- R. Gilberg, B. Forouzan, "Data Structures: A pseudo Code Approach with C++", Cengage Learning, ISBN 9788131503140.
- E. Horowitz, S. Sahni, D. Mehta, "Fundamentals of Data Structures in C++", Galgotia Book Source, New Delhi, 1995, ISBN 16782928
- Dinesh P. Shah, Sartaj Sahani , “Handbook of DATA STRUCTURES and APPLICATIONS”, CHAPMAN & HALL/CRC
- Bayer B. et al. (2015) Electro-Mechanical Brake Systems. In: Winner H., Hakuli S., Lotz F., Singer C. (eds) Handbook of Driver Assistance Systems. Springer, Cham
- **Web**
  - http://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node55.html
  - https://en.wikipedia.org/wiki/Persistent_data_structure

DSA Unit 1.4 Insersion Sort

No copyright infringement is intended

17



18

Thank You !!!

DSA Unit 1.4 Insersion Sort

18