Divide and Conquer
Selection Sort

Dr. K.RAGHAVA RAO

Professor in CSE

KL University

krraocse@gmail.com

http://mcadaa.blog.com

Selection Sort

- Defintion
  - first find the smallest in the array and exchange it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continue in this way until the entire array is sorted.
- Selection sort is:
  - The simplest sorting techniques.
  - a good algorithm to sort a small number of elements
  - an incremental algorithm – induction method
- Selection sort is Inefficient for large lists.

Incremental algorithms *→ process the input elements one-by-one and maintain the solution for the elements processed so far.*

Selection Sort

- Let $A[1\ldots n]$ be an array of n elements. A simple and straightforward algorithm to sort the entries in A works as follows. First, we find the minimum element and store it in $A[1]$. Next, we find the minimum of the remaining n-1 elements and store it in $A[2]$. We continue this way until the second largest element is stored in $A[n-1]$.

Selection Sort algorithm

Input: $A[1\ldots n]$;

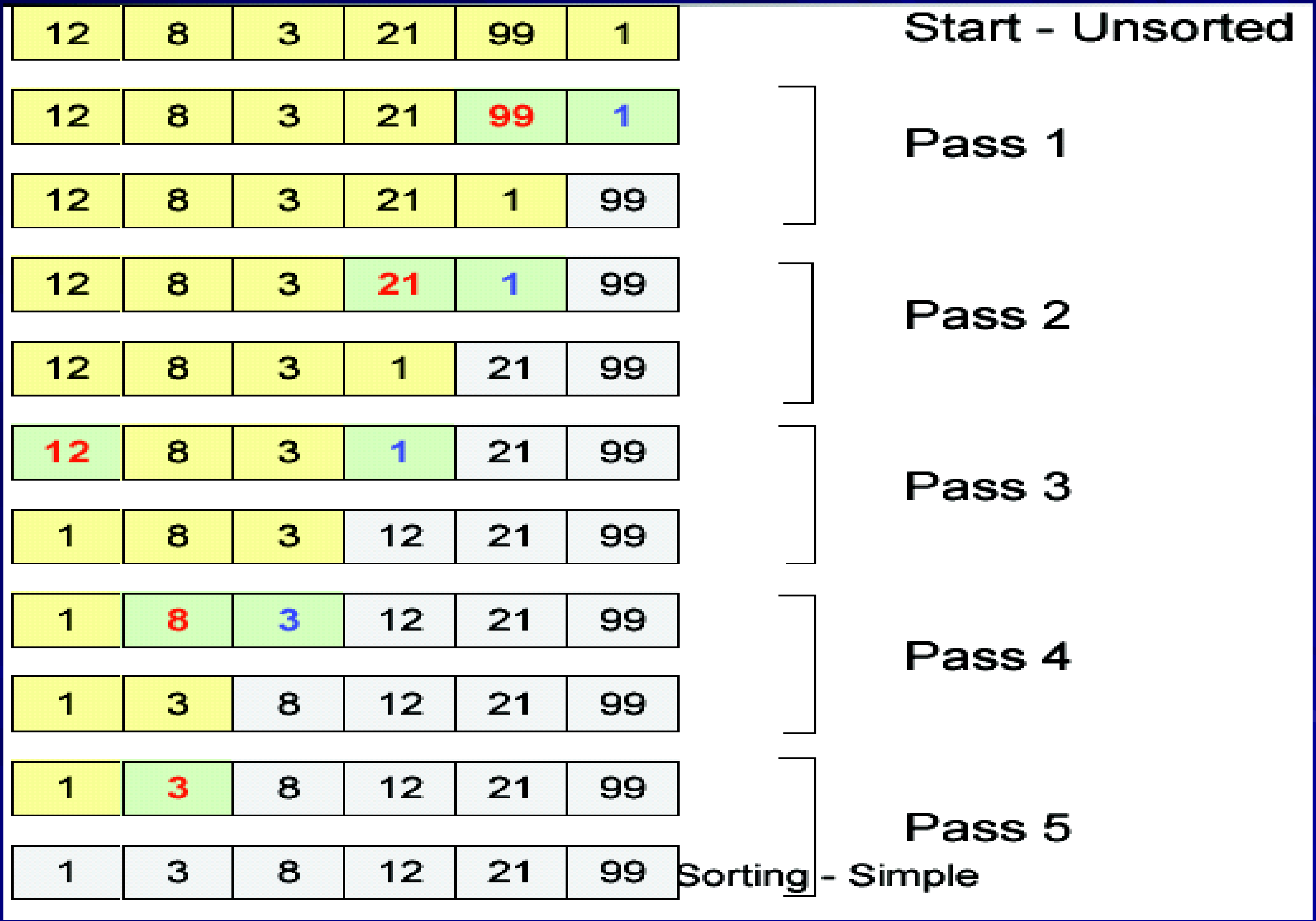
Output: $A[1\ldots n]$ sorted in nondecreasing order;

```
1. for $i \leftarrow 1$ to $n$-1
2.     $k \leftarrow i$;
3.     for $j \leftarrow i+1$ to $n$
4.         if $A[j]<A[k]$ then $k \leftarrow j$;
5.     end for;
6.     if $k \neq i$ then interchange $A[i]$ and $A[k]$;
7. end for;
```

Selection Sort

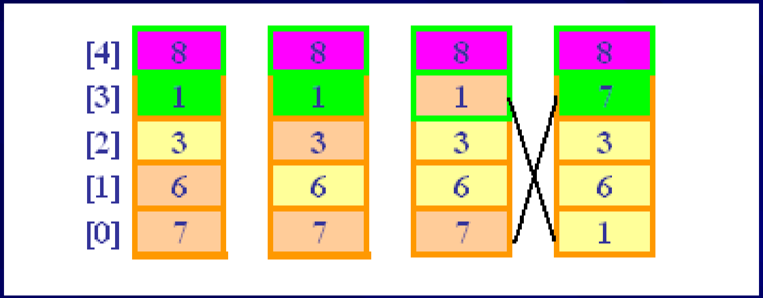
- Take multiple passes over the array
- Keep already sorted array at high-end
- Find the biggest element in unsorted part
- Swap it into the highest position in unsorted part
- **Invariant:** each pass guarantees that one more element is in the correct position (same as bubbleSort)
- a lot fewer swaps than bubbleSort!

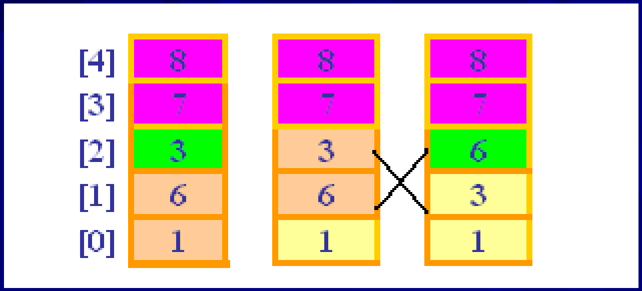
Selection Sort



12	8	3	21	99	1
12	8	3	21	99	1
12	8	3	21	1	99
12	8	3	21	1	99
12	8	3	1	21	99
12	8	3	1	21	99
1	8	3	12	21	99
1	8	3	12	21	99
1	3	8	12	21	99
1	3	8	12	21	99
1	3	8	12	21	99

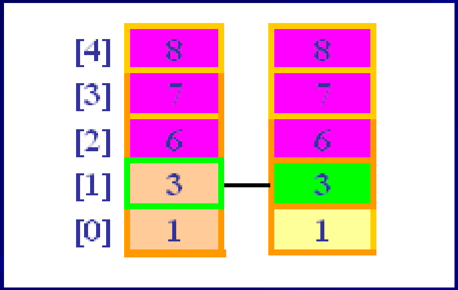
Example Execution of selection sort Tracing



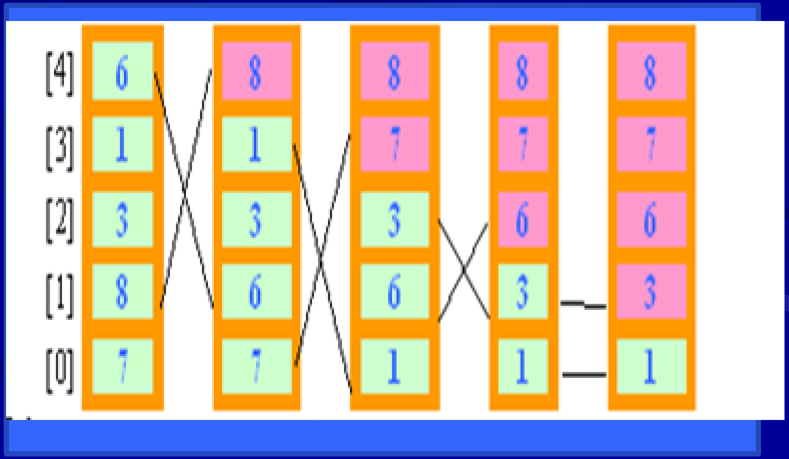


Pass 2
last = 3
largestIndex = 0, 0, 0
p = 1, 2, 3

Pass 3
last = 2
largestIndex = 0, 1
p = 1, 2



last = 4 3 2 1



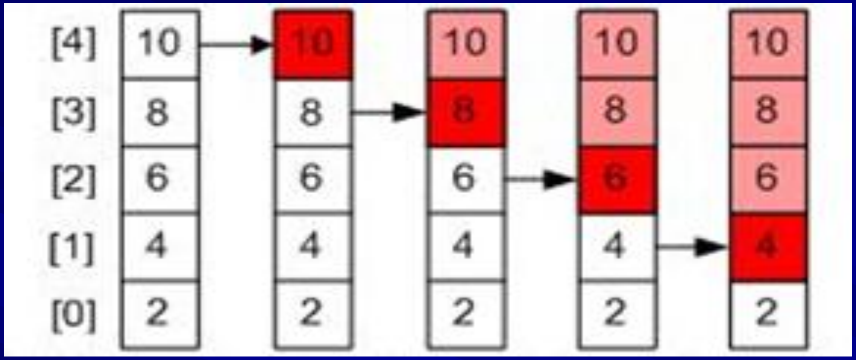
Pass 4
last = 1
largestIndex = 0, 1
p = 1

largestIndex = 1 0 1 1

Step by step changes/swaps in the list that show the swapping process during selection sort

Selection Sort Implementation for Best Case [2 4 6 8 10]

last = 4 3 2 1



largestIndex = 4 3 2 1

Step by step changes/swaps in the list that show the swapping process during selection sort implementation on array [2 4 6 8 10]

Selection Sort Analysis

- For an array with size n, the **external loop** will iterate from *n-1 to 1.*

```
for (int last = n-1; last>=1; --last)
```

- For each iteration, to find the largest number in subarray, the number of comparison inside the **internal loop** must is equal to the value of last.

```
for (int p=1;p <=last; ++p)
```

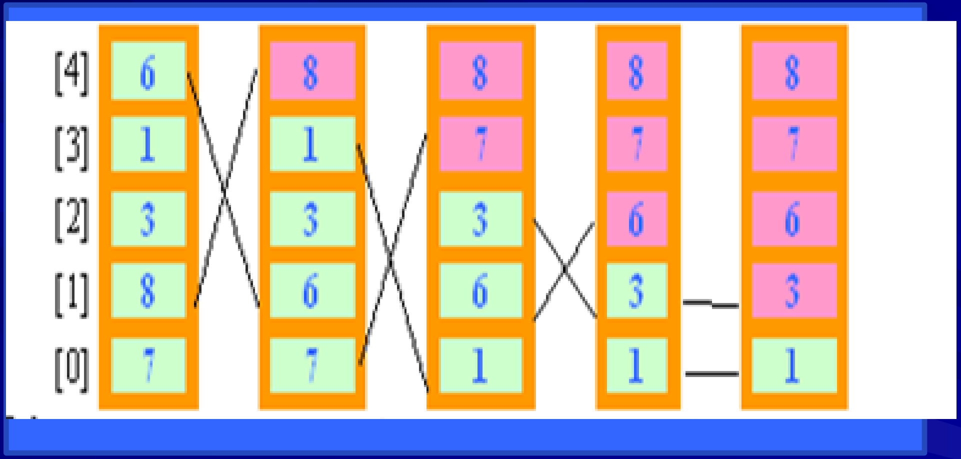
- Therefore the total comparison for Selection Sort in each iteration is *(n-1)* + *(n-2)* + *….. 2* + *1*.
- Generally, the number of comparisons between elements in Selection Sort can be stated as follows:

$$(n-1)+(n-2)+.......+2+1=\frac{n(n-1)}{2}=O(n^2)$$

Selection Sort Analysis

Similar To Bubble Sort, In any cases of Selection Sort (worst case, best case or average case) the number of comparisons between elements is the same.

last = 4 3 2 1



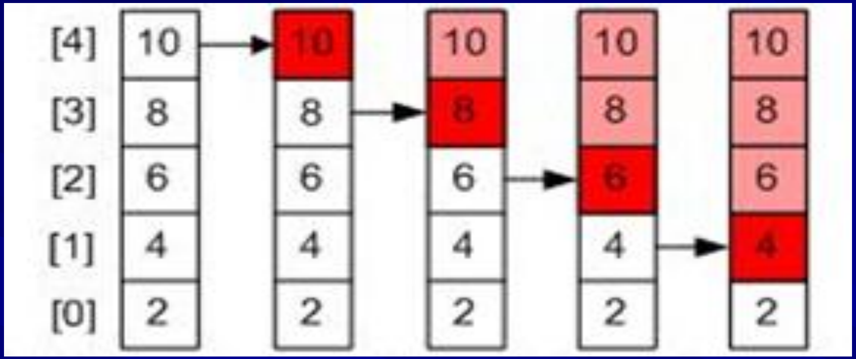
largestIndex = 1 0 1 1

Number of Comparisons: 4 + 3 + 2 + 1 = 10

For array n= 5 => (n-1) +(n-2) + ….+ 2 + 1 = n(n-1)/2 = O(n^2)

Selection Sort Analysis

last = 4 3 2 1



largestIndex = 4 3 2 1

Number of Comparisons for best case : 4 + 3 + 2 + 1 = 10

For array n= 5 =>(n-1) +(n-2)+ …. + 2 + 1 = n(n-1)/2 = O(n^2)

Selection Sort – Algorithm Complexity

- Time Complexity for Selection Sort is the same for all cases - worst case, best case or average case $O(n^2)$.
- The number of comparisons between elements is the same.
- The efficiency of Selection Sort does not depend on the initial arrangement of the data

Alg.: SELECTION-SORT(*A*)

```
1 n ← length[A]
 2  for j ← 1 to n - 1
3          do smallest ← j
4     for i ← j + 1 to n
5          do if A[i] < A[smallest]
6    then smallest ← i
 7 exchange A[j] ↔ A[smallest]
```

cost	times
c_1	1
c_2	n
c_3	n-1
c_4	$\sum_{j=1}^{n-1}(n-j+1)$
c_5	$\sum_{j=1}^{n-1}(n-j)$
c_6	$\sum_{j=1}^{n-1}(n-j)$
c_7	n-1

$$T(n) = c_1 + c_2 n + c_3(n-1) + c_4\sum_{j=1}^{n-1}(n-j+1) + c_5\sum_{j=1}^{n-1} n-j \ + c_6\sum_{j=2}^{n-1} n-j \ + c_7(n-1) = \Theta(n^2)$$