SCJ2013 Data Structure & Algorithms

Selection Sort

Nor Bahiah Hj Ahmad & Dayang
Norhayati A. Jawawi

Selection Sort

- Strategy
  - Choose the largest/smallest item in the array and place the item in its correct place
  - Choose the next larges/next smallest item in the array and place the item in its correct place.
  - Repeat the process until all items are sorted.
- Does not depend on the initial arrangement of the data
- Only appropriate for small *n* - O(n^2) algorithm

Selection Sort Implementation

```
void selectionSort(DataType Data[], int n)
{
  for (int last = n-1; last >= 1; --last)
  {// select largest item in theArray

     int largestIndex = 0;

     // largest item is assumed start at index 0

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

     {   if (Data[p] > Data[largestIndex])

                 largestIndex = p;

     }  // end for


   // swap largest item Data[largestIndex] with
   // Data[last]
    swap(Data[largestIndex],Data[last]);
  } // end for
}  // end selectionSort
```

last : index of the last item in the subarray of items yet to be sorted.

largestIndex : index of the largest item found

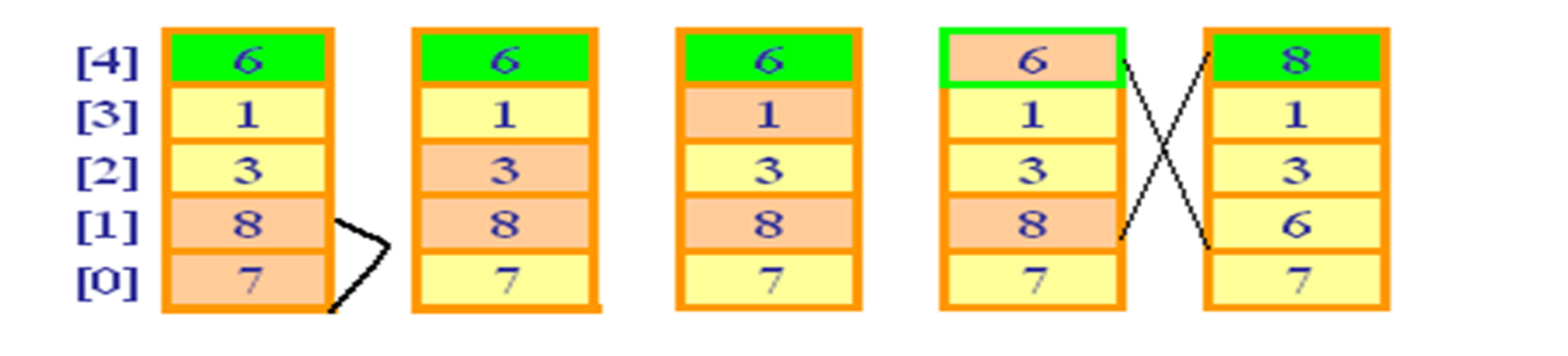
swap: change largest value with item at last index of the subarray.

`swap()`function

Swap function interchange value x and y. In this example both x and y need to be pass by reference

```
void swap(DataType& x, DataType& y)
{   DataType temp = x;
    x = y;
    y = temp;
}  // end swap
```

Selection Sort Implementation: [7 8 3 1 6]

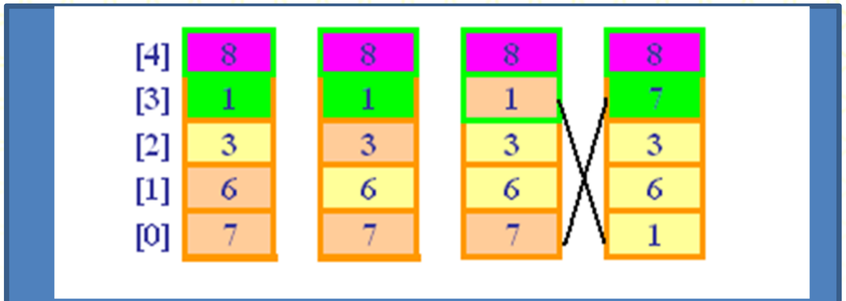


Pass 1, last = 4
largestIndex = 0, 1, 1, 1
p = 1, 2, 3, 4

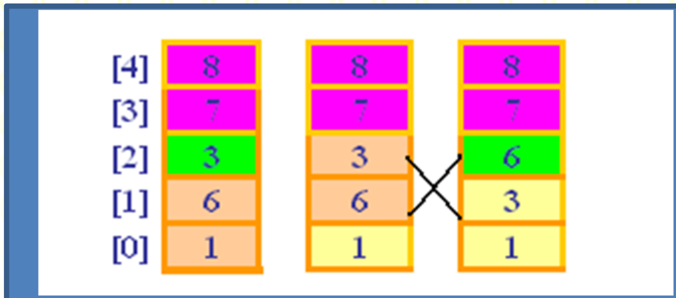
Starting from index 1 to index 4, the largest value in the array will be searched. In pass 1, the largest value is 8 and was found at index 1. Therefore value at index 1 (8) will be swap with value at index last(4).

There are 4 comparisons in this pass.

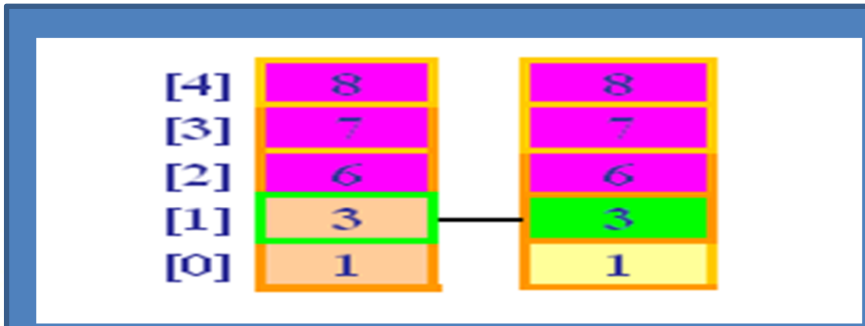
Selection Sort Implementation: [7 8 3 1 6]



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



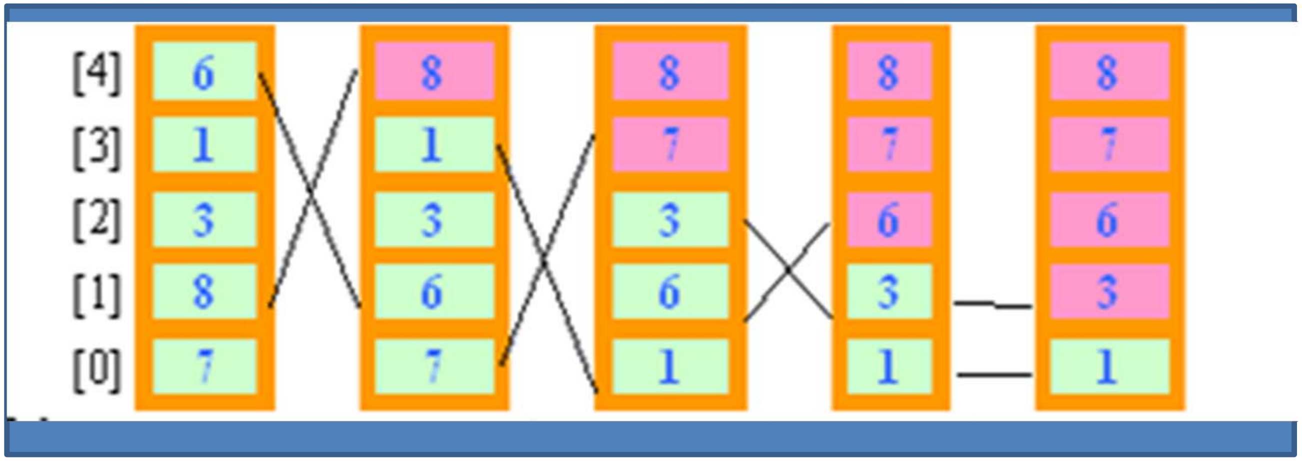
Pass 3
last = 2
largestIndex = 0, 1
p = 1, 2



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

Selection Sort Implementation: [7 8 3 1 6]

last = 4 3 2 1

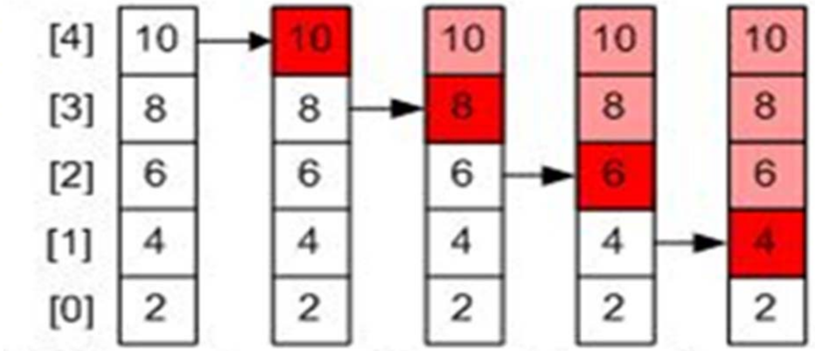


largestIndex = 1 0 1 1

Step by step changes in the list that show the swapping process during selection sort implementation on array [7 8 3 1 6]

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

last = 4 3 2 1



[4]	10	10	10	10	10
[3]	8	8	8	8	8
[2]	6	6	6	6	6
[1]	4	4	4	4	4
[0]	2	2	2	2	2

largestIndex = 4 3 2 1

Step by step changes 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)
```

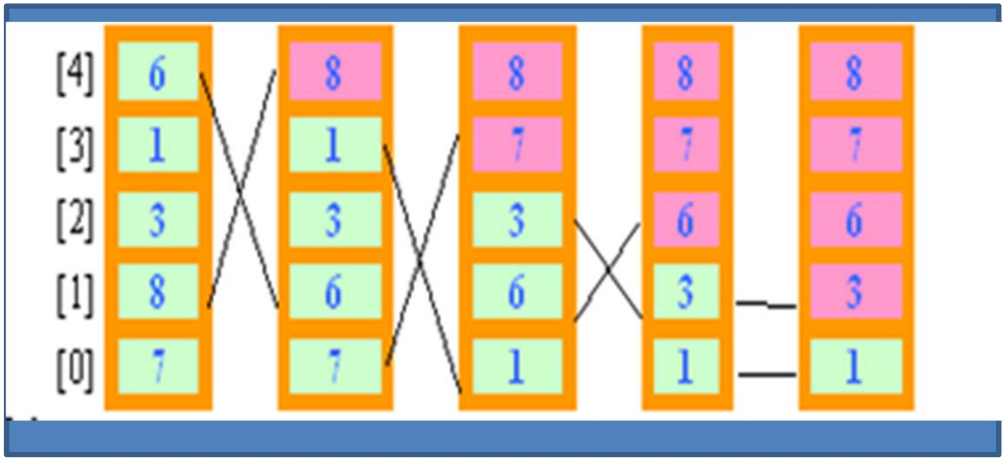
- Therefore the total comparison for Selection Sort in each iteration is *(n-1) + (n-2) + ... 2 + 1*.
- Generally, the number of comparisons between elements in Selection Sort can be stated as follows:

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

Selection Sort Analysis

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

last = 4 3 2 1



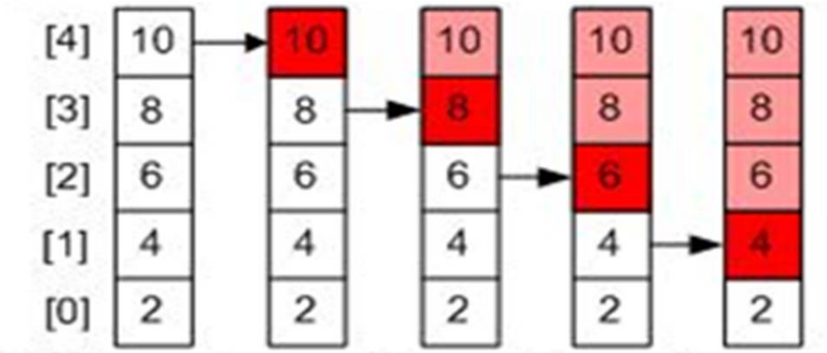
largestIndex = 1 0 1 1

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 Issue

- It can be seen that the swapping process occur even though the largest index is at last.
- This is not efficient and can be improved by putting a condition statement as follows:

```
If (largestIndex !=last);
     swap(Data[largestIndex],Data[last]);
```

Selection Sort – Algorithm Complexity

Selection	Comparison	Swap
Best Case	O(n^2)	O(n)
Average Case	O(n^2)	O(n)
Worst Case	O(n^2)	O(n)

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