DATA STUCTURE

ALGORITHMS

ALGORITHMS

SEARCH ALGORITHMS

SEARCH ALGORITHMS

Linear Search

RULE

```
LinearSearch(A , size, target)
    For i=0 to size -1
    If(A[i] == target) return i
    Else return -1
```

EXAMPLE

45

45	34	64	55	67	12	57

1	34	45	55	67	12	57

1	34	9	55	67	12	45

RUNNING TIME

Best: 1 comparable time

Worst: n comparable time

AVG: (n+1) /2 comparable time

IMPLEMENT LINEAR SEARCH

```
Public int linearSearching(int[] array, int target){
for(….){
If(array[i]== target){
//TODO
}
}
```

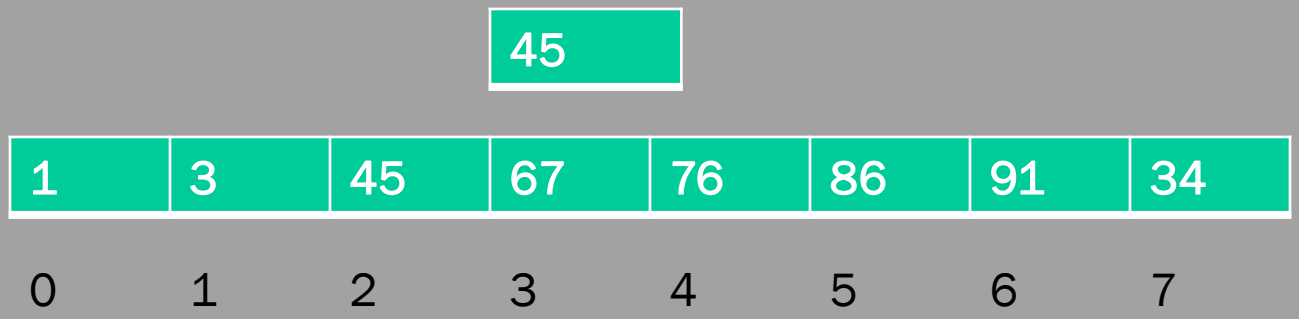
SEARCH ALGORITHM

Binary Search

RULE

```
BinarySearch(A[0..N-1], value) {
    low = 0
    high = N - 1
    while (low <= high) {
    mid = (high + low) / 2)
    if (A[mid] = value) return value;
    else if (A[mid] > value) high = mid - 1
    else
    low = mid + 1
    }
    return -1111;// no element in array equals value
}
```

EXAMPLE



Step1 : size of array => N= 8; high = N-1 =7, low =0, mid = (low +high)/2 = 7/2 = 3
Array[3] =67 > 45 => high = mid -1 = 3-1 = 2

Step2 : high =2, low =0, mid = (low +high)/2 = 2/2 =1
Array[1] =3 <45 => low = mid +1 = 1+1 = 2

Step3: high =2,low =2 => mid =(low +high) /2 = 4/2 =2
Array[2] =45 =45 ->| stop

RUNNING TIME

Best: ceil($log_2(n)$) +1

Worst: floor($log_2(n)$) +1

AVG: approx.$log_2(n)$ +1

IMPLEMENT BINARY SEARCH

```
Public int binarySearching(int[] sortedArray, int target){
  //TODO
}
```

ALGORITHM

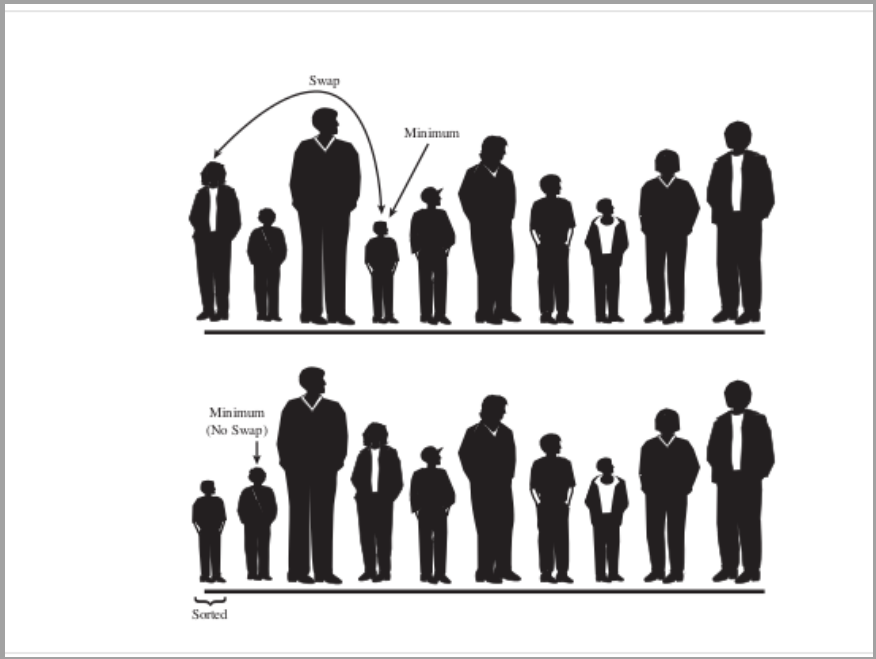
SORT ALGORITHM

SORT ALGORITHM

SELECT SORT ALGORITHM

SELECTION SORT ALGORITHM

- *Finding the smallest (or largest depending on the sorting order) element in the unsorted sub list exchanging it with the leftmost unsorted element (putting in sorted order) and moving the sub list boundaries one element to the right.*
-



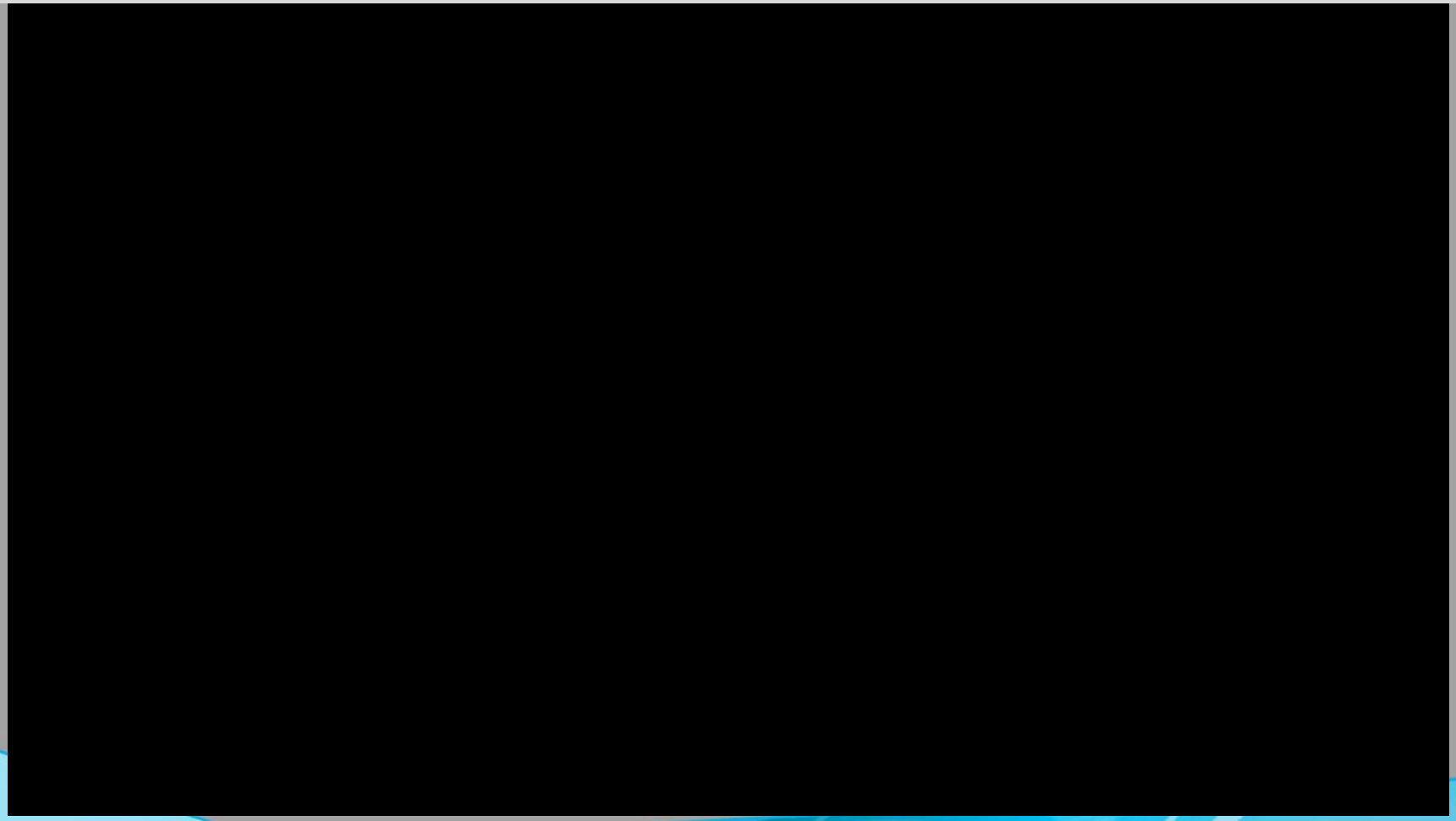
RUNNING TIME

Best: $O(n^2)$

Worst: $O(n^2)$

AVG: $O(n^2)$

Video



Example

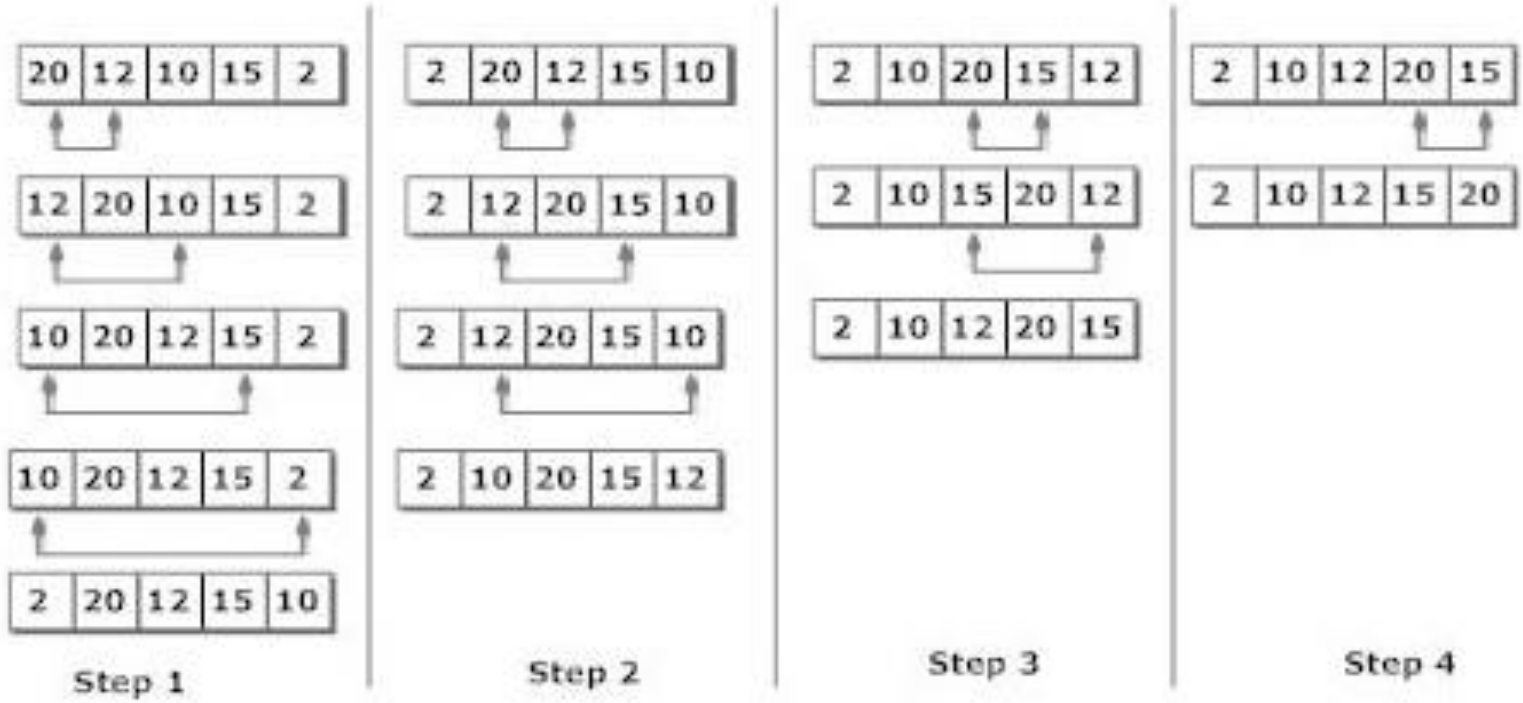


Figure: Selection Sort

RULE

Step 1: i = 1

Step 2: Finding X[min] or X[max] in X[i] ... X[n]

Step 3: Swap X[i] to X[min], if min or max equal i , quit this step.

Step 4:

** If i <= n-1 so that i = i +1, run step 2 again.*

** Else, stop, finish sort array.*

IMPLEMENT SELECTION SORT ALGORITHM

```
public class SelectionSort {
        private static void swap(int[] a, int i, int j) {
        // switch value at index i to value at index  j
}
        public static int[] selectionSort_Min(int[] array) {
        for (int i = 0; i < array.length - 1; i++) {
         for (int j = i + 1; j < array1.length; j++) {
        // Find the index of the minimum value
        // swap
        }}
        return array; }
```

IMPLEMENT SELECTION SORT ALGORITHM

```
public static int[] selectionSort_Max(int[] array) {
        for (int i = 0; i < array.length - 1; i++) {
        // Find the index of the max value
        // swap
        return array; }
```

SORT ALGORITHM

Bubble sort

Bubble sort (sinking sort)

- *Sorting is to place elements in increasing or decreasing order.*
- *Comparing the adjacent pair , if they are in not right order , then they swapped each other position. When there are no elements swapped in one full iteration of element list , then it indicates that bubble sort is completed.*

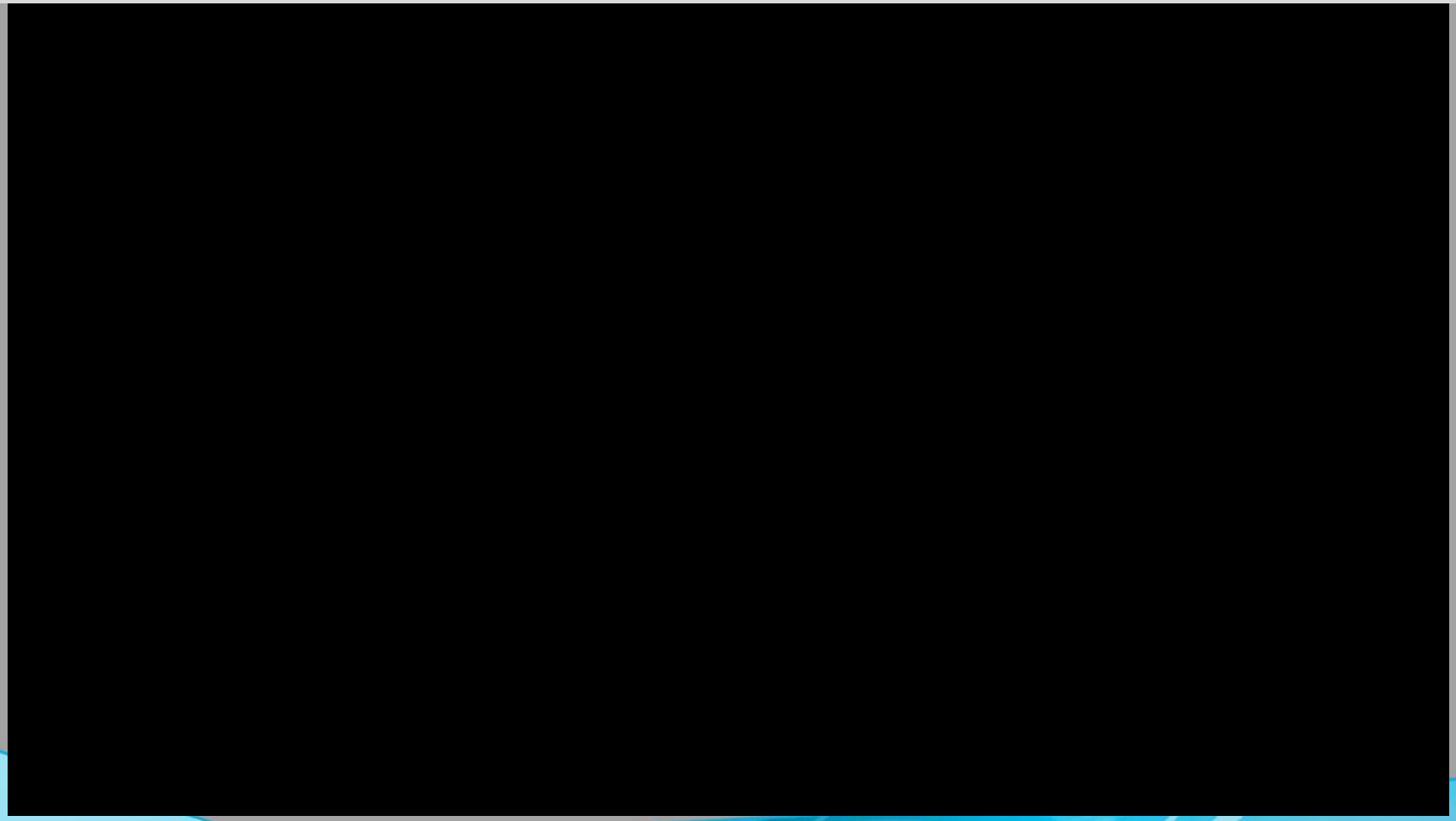
RUNNING TIME

Best: O(n)

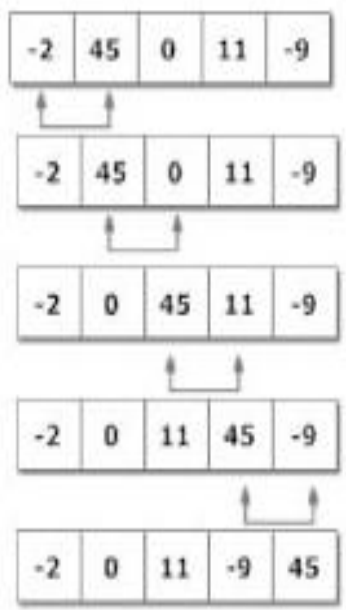
Worst: $O(n^2)$

AVG: $O(n^2)$

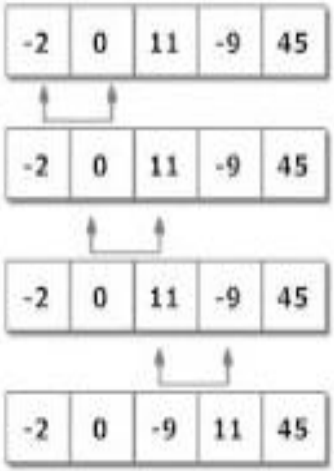
Video



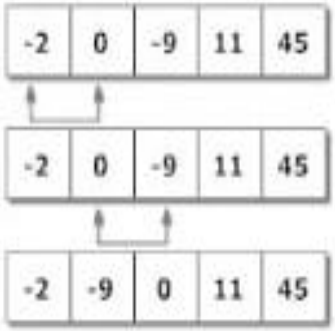
Example



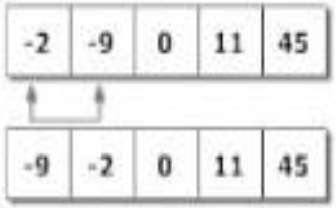
Step 1



Step 2



Step 3



Step 4

Figure: Working of Bubble sort algorithm

RULE

Step 1: i=1

Step 2: compare max or min and swap (if necessary) from X[i] to X[n] or X[n] to X[i]

Step 3: i=i+1

Step 4:

** If i < n, run step 2 again.*

** Else, stop, finish sorted array.*

IMPLEMENT BUBBLE SORT ALGORITHM

```
public class BubbleSort {

 public static int[] bubbleSort_Min(int[] array) {
            for (int i = (array.length - 1); i >= 0; i--) {
             for (int j = 1; j <= i; j++) {
            //TODO
}
 public static int[] bubbleSort_Max(int[] array) {
            for (int i = (array.length - 1); i >= 0; i--) {
             for (int j = 1; j <= i; j++) {
            //TODO
}

}
```

SORT ALGORITHM

Insertion sort

Definition

1. *Divide to sub list with 2 elements (begin or end)*
2. *Compare each element with other in sub list, sorted it by ASC or DESC*
3. *Adding 1 element to sub list and do step 2 again, until has sorted list*

RULE

for i ← 1 **to** length(A)-1

j ← i **while** j > 0 and A[j-1] > A[j]

swap A[j] and A[j-1] j ← j – 1

end while end for

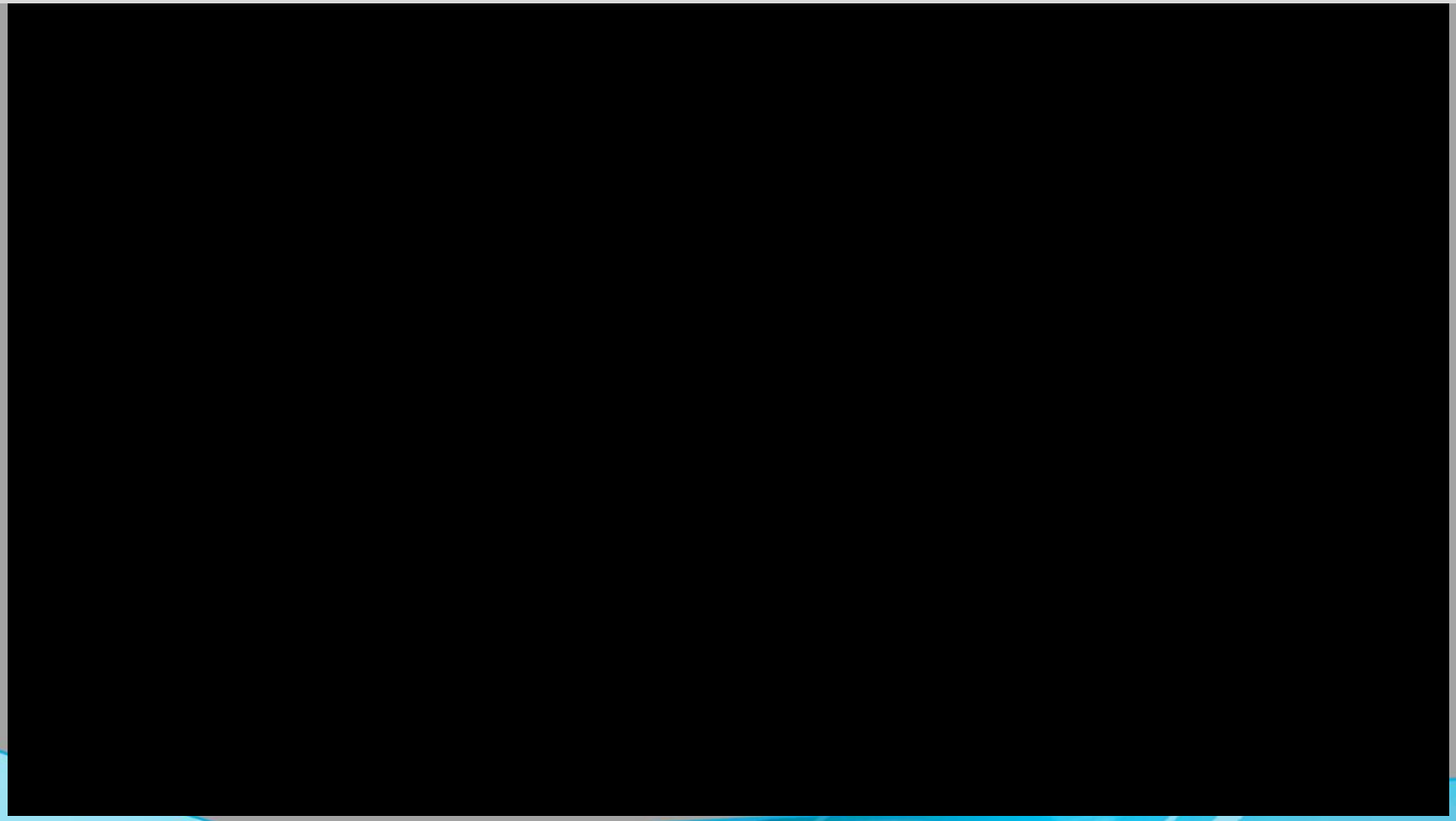
RUNNING TIME

Best: O(n)

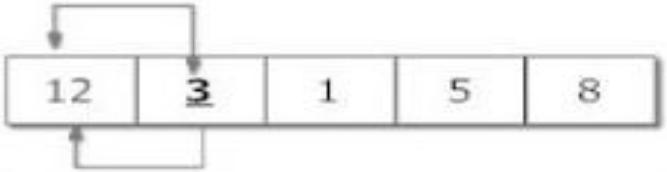
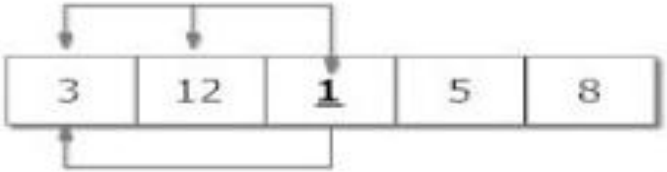
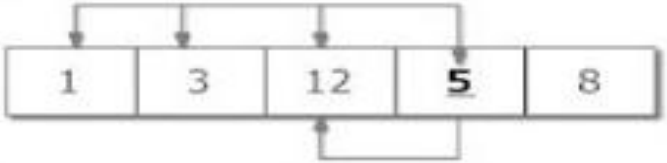
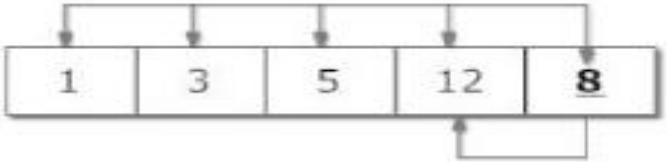
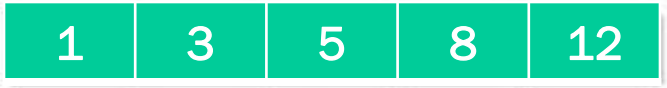
Worst: $O(n^2)$

AVG: $O(n^2)$

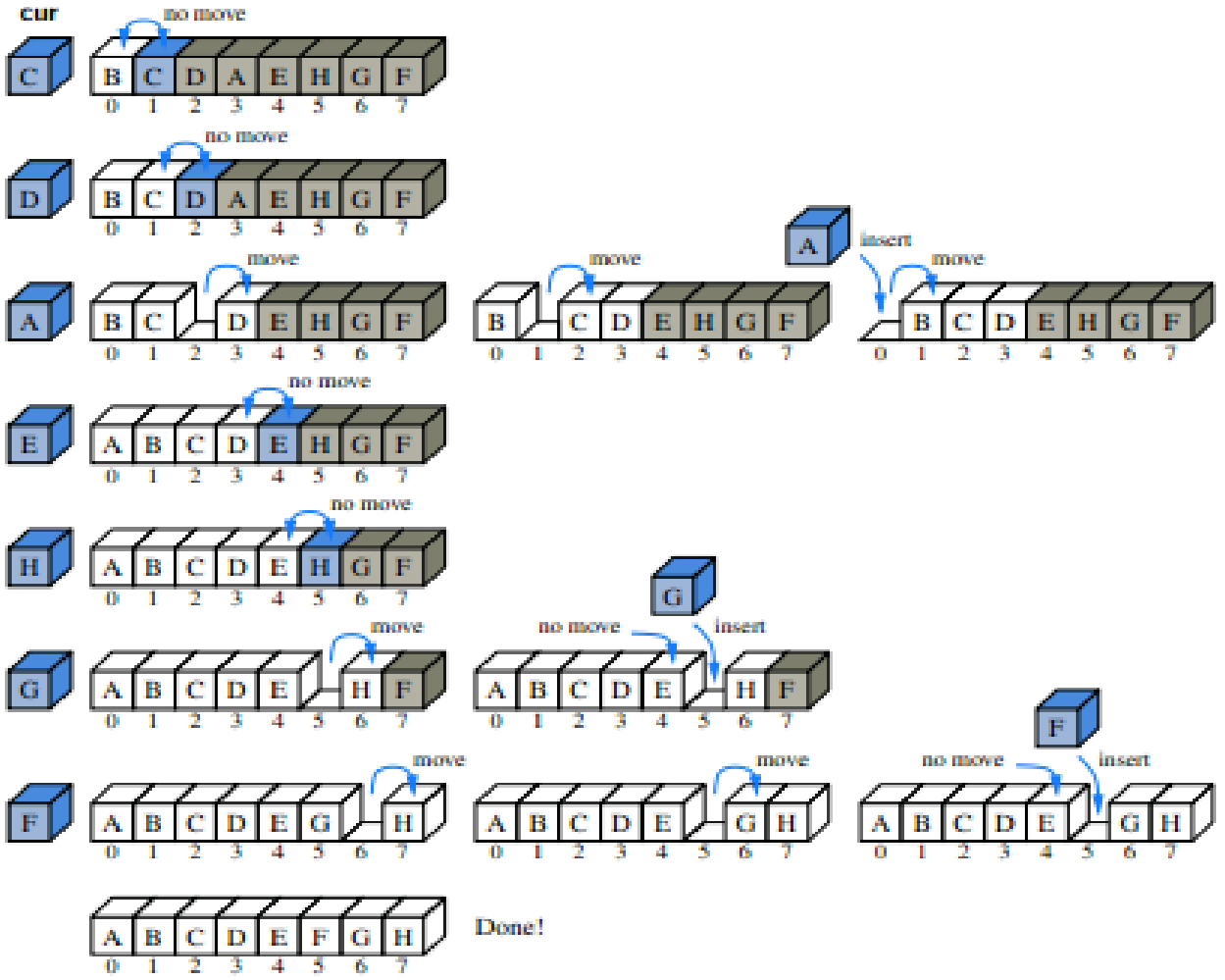
Video



EXAMPLE

Step 1		Checking second element of array with element before it and inserting it in proper position. In this case, 3 is inserted in position of 12.
Step 2		Checking third element of array with elements before it and inserting it in proper position. In this case, 1 is inserted in position of 3.
Step 3		Checking fourth element of array with elements before it and inserting it in proper position. In this case, 5 is inserted in position of 12.
Step 4		Checking fifth element of array with elements before it and inserting it in proper position. In this case, 8 is inserted in position of 12.
		Sorted Array in Ascending Order

EXAMPLE



IMPLEMENT INSERT-SORT ALGORITHM

```
/** Insertion-sort of an array of characters into nondecreasing order */
public static void insertionSort(char[ ] data) {
  int n = data.length;
  for (int k = 1; k < n; k++) {                  // begin with second character
    char cur = data[k];                          // time to insert cur=data[k]
    int j = k;                                   // find correct index j for cur
    while (j > 0 && data[j−1] > cur) {           // thus, data[j-1] must go after cur
      data[j] = data[j−1];                       // slide data[j-1] rightward
      j−−;                                       // and consider previous j for cur
    }
    data[j] = cur;                               // this is the proper place for cur
  }
}
```