DATA STUCTURE

ALGORITHMS

Timing an algorithm

```
long startTime = System.currentTimeMillis();          // record the starting time
/* (run the algorithm) */
long endTime = System.currentTimeMillis();            // record the ending time
long elapsed = endTime − startTime;                   // compute the elapsed time
```

Code Fragment 4.1: Typical approach for timing an algorithm in Java.

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

SEARCH ALGORITHM

Binary Search

HOW IT WORK ?

- *Compare x with the middle element.*
- *If x matches with middle element, we return the mid index.*
- *Else If x is greater than the mid element, then x can only lie in right half subarray after the mid element. So we recur for right half.*
- *Else (x is smaller) recur for the left half.*

EXAMPLE

45

1	3	45	67	76	86	91	134
0	1	2	3	4	5	6	7

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

USING RECURSIVE TO IMPLEMENT BINARY SEARCH

- *Just using for sorted array*
- *Low = begin of array, high = end of array, mid = (high + low)/2*
- *STOP CONDITION*
  - *Target equals with value of element at middle*
- *CONTINIOUS CONDITION*
  - *Target larger than mid → Recursive with low= mid+1, high not change*
  - *Target smaller than mid → Recursive with low not change, high = mid -1*

USING RECURSIVE TO IMPLEMENT BINARY SEARCH

```
/**
 * Returns true if the target value is found in the indicated portion of the data array.
 * This search only considers the array portion from data[low] to data[high] inclusive.
 */
public static boolean binarySearch(int[ ] data, int target, int low, int high) {
  if (low > high)
    return false;                                          // interval empty; no match
  else {
    int mid = (low + high) / 2;
    if (target == data[mid])
      return true;                                         // found a match
    else if (target < data[mid])
      return binarySearch(data, target, low, mid − 1);     // recur left of the middle
    else
      return binarySearch(data, target, mid + 1, high);    // recur right of the middle
  }
}
```

NO USING RECURSIVE TO IMPLEMENT BINARY SEARCH

```
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
}
```

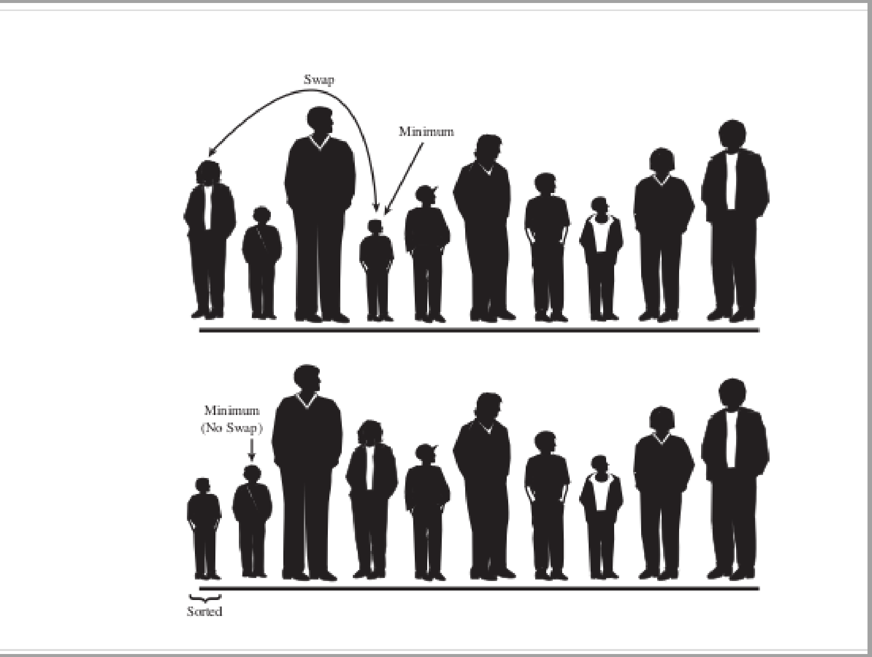
ALGORITHM

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.*
- 

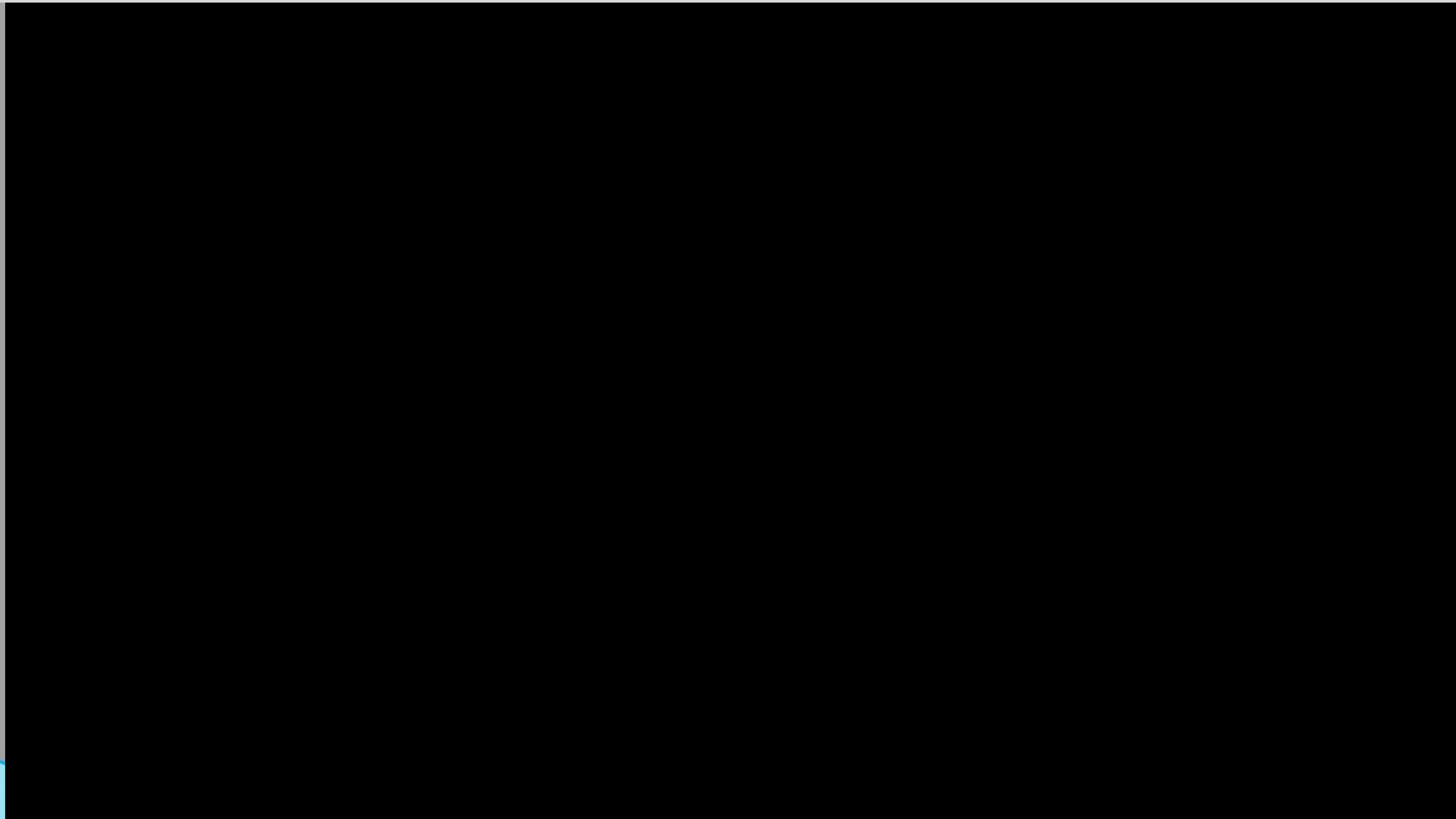
RUNNING TIME

Best: $O(n^2)$

Worst: $O(n^2)$

AVG: $O(n^2)$

Video



Example

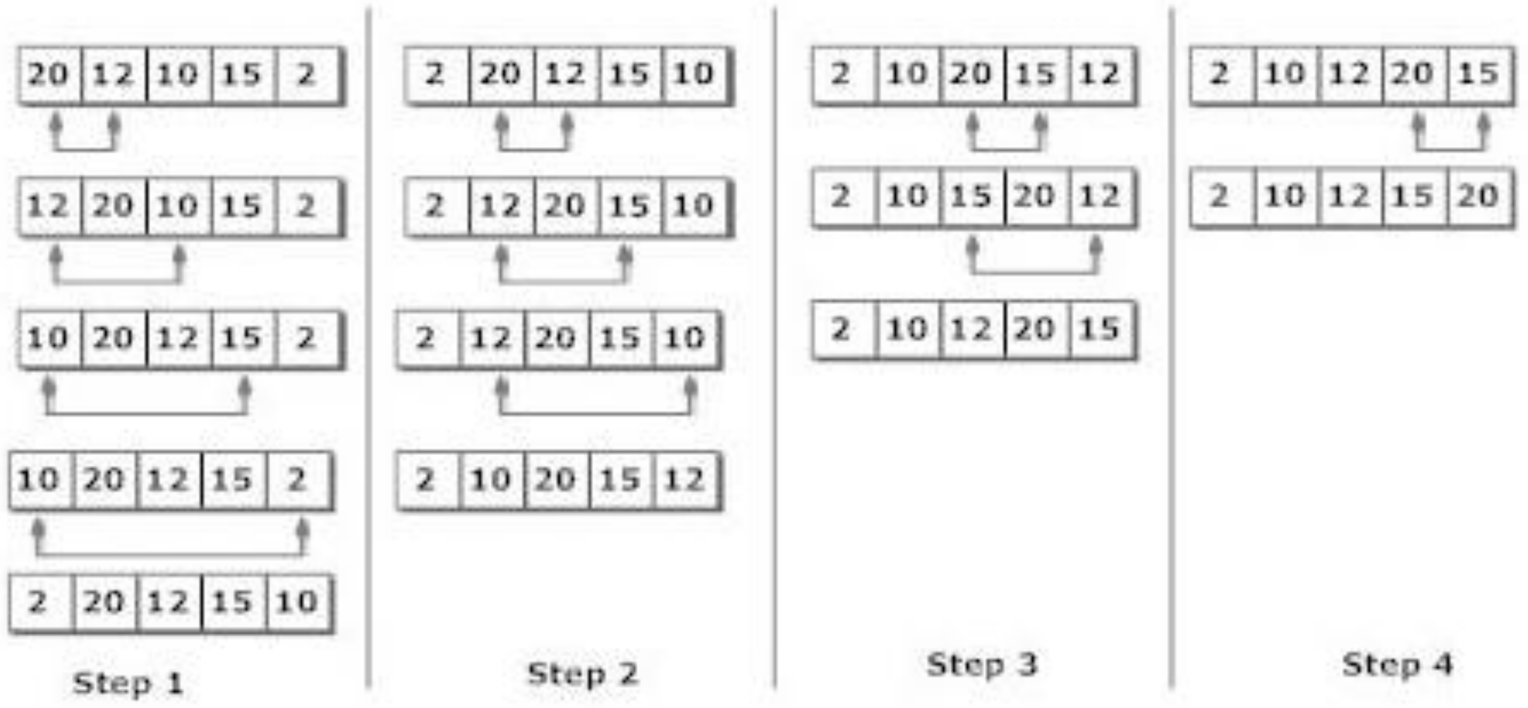


Figure: Selection Sort

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.*

RECURSIVE 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,int stepNum) {
        if(stepNum > array.length -1){
                return array;
        } else{
        for (int j = stepNum; j < a.length; j++) {
        // Find the index of the minimum value
        // swap
        }
       }
                return selectionSort_Min(array,stepNum + 1) ; }
```

NON RECURSIVE 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 < array.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++) {
         for (int j = i + 1; j < array.length; j++) {
        // Find the index of the max value
        // swap
        return array; }
```

Bài tập ứng dụng

Quản lý học sinh