DESIGN AND ANALYSIS OF ALGORITHMS

CS 4120/5120

THE CORRECTNESS OF AN ALGORITHM

CORRECTNESS OF AN ALGORITHM

- Recall
  - Algorithm is said to be ***correct*** if, for every input instance, it halts with the correct output.
- Sequential procedure
  - Step-by-step verification
- Loop structure
  - Some properties that **hold true throughout** the entire loop procedure

CORRECTNESS OF AN ALGORITHM
CASE STUDY

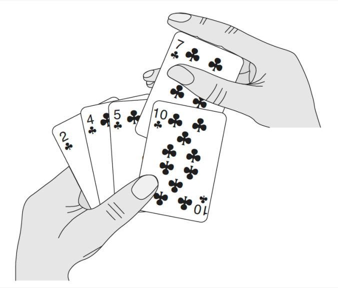
- The given problem
  - Input: $A[1 \dots n]$ with distinct numbers.
  - Output: A permutation of $A[1 \dots n]$ such that $A[i] < A[i+1]$, for $i \in [1, n-1]$
- Solve by the INSERTION-SORT algorithm

CORRECTNESS OF AN ALGORITHM
CASE STUDY: INSERTION-SORT

- The INSERTION-SORT algorithm

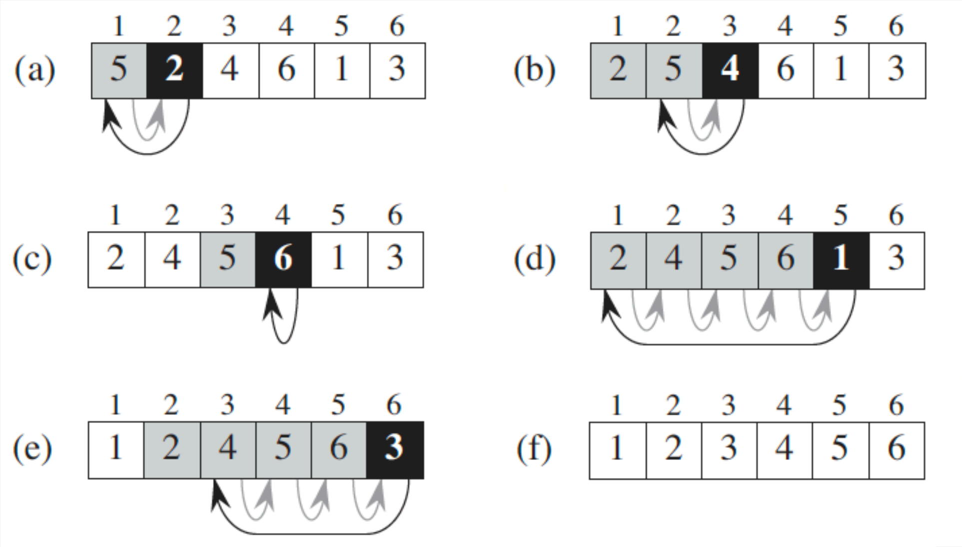
```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j − 1]
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

- Follow through the code to sort input instance $< 5, 2, 4, 6, 1, 3 >$.



CORRECTNESS OF AN ALGORITHM
CASE STUDY: INSERTION-SORT

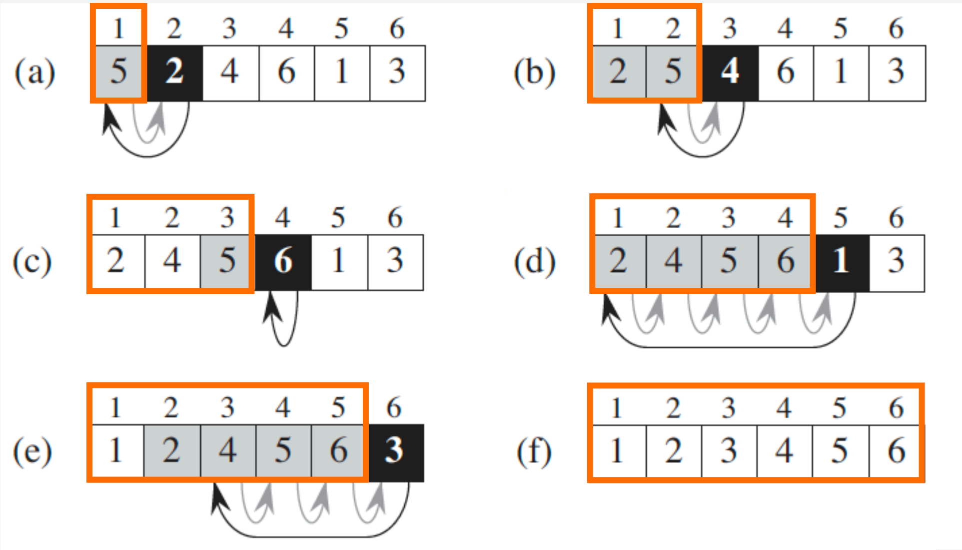
- Follow through the code to sort input instance $< 5, 2, 4, 6, 1, 3 >$.




```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the
           sorted sequence A[1..j − 1]
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
CASE STUDY: IS IT CORRECT?

- The blackened element is $A[j]$.
- Which part of the input (or what subarray of the input) stays sorted from (a) through (f)?




```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the
           sorted sequence A[1..j − 1]
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
LOOP INVARIANT

- The properties of subarray `A[1..j-1]` is called the ***loop invariant***
  - At the start of each iteration of the **for** loop of line 1-8, the subarray $A[1..j-1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order.

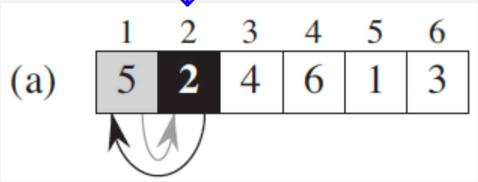
- We say the algorithm is correct if
  - the ***loop invariant*** holds true prior to the **initial** iteration
  - each iteration **maintains** the correctness of the ***loop invariant***
  - the ***loop invariant*** holds true at **termination**

```
INSERTION-SORT(A)
1   for j = 2 to A.length
2       key = A[j]
3       // Insert A[j] into the
            sorted sequence A[1..j-1]
4       i = j - 1
5       while i > 0 and A[i] > key
6           A[i + 1] = A[i]
7           i = i - 1
8       A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
LOOP INVARIANT @ INITIALIZATION

- The ***loop invariant*** must be true **prior to the first iteration** of the loop.

j ↓



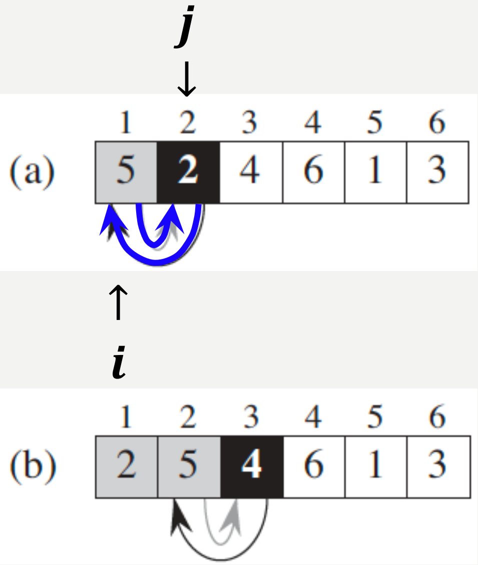
↑ ***i***

$\boldsymbol{key = 2}$

```
INSERTION-SORT(A)
    for j = 2 to A.length
        key = A[j]
        // Insert A[j] into the
           sorted sequence A[1..j − 1]
        i = j - 1
        while i > 0 and A[i] > key
            A[i + 1] = A[i]
            i = i - 1
        A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
LOOP INVARIANT @ MAINTENANCE

- If the ***loop invariant*** is true before an iteration of the loop, it **remains true before the next iteration**.



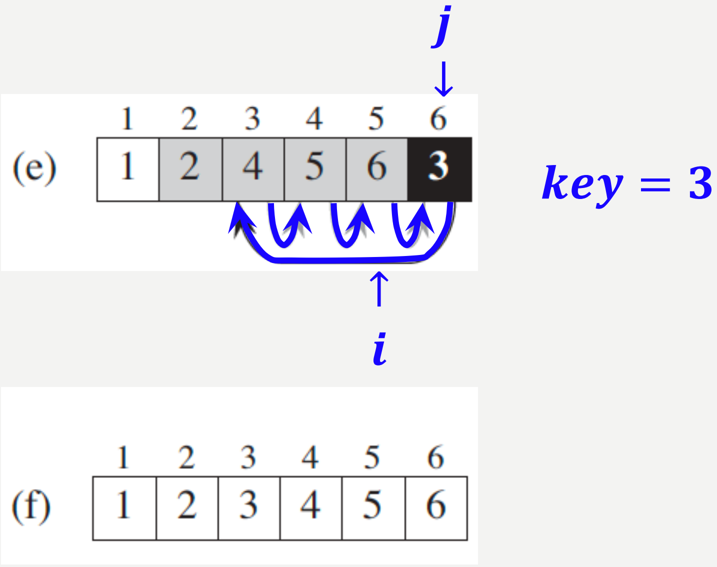
$\boldsymbol{key = 2}$

$\boldsymbol{A[i] = 5 > key}$

```
INSERTION-SORT(A)
   for j = 2 to A.length
       key = A[j]
       // Insert A[j] into the
          sorted sequence A[1..j-1]
       i = j - 1
       while i > 0 and A[i] > key
           A[i + 1] = A[i]
           i = i - 1
       A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
LOOP INVARIANT @ TERMINATION

- When the loop **terminates**, the ***loop invariant*** shows the algorithm is correct.




```
INSERTION-SORT(A)
   for j = 2 to A.length
       key = A[j]
       // Insert A[j] into the
          sorted sequence A[1..j − 1]
       i = j - 1
       while i > 0 and A[i] > key
           A[i + 1] = A[i]
           i = i - 1
       A[i + 1] = key
```

CORRECTNESS OF AN ALGORITHM
PRACTICE

- Read the following C++/Java code. Assume that a function **void** `swap(int *xp, int *yp)` is visible to the `bubbleSort` function.

```
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n-1; i++)
      for (j = 0; j < n-i-1; j++)
          if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}
```

$BubbleSort(A)$

  - Write the corresponding pseudocode.
  - Then, define the loop invariant of the outermost loop.

CORRECTNESS OF AN ALGORITHM
PRACTICE

- The pseudocode corresponds to the C++ source code

```cpp
void bubbleSort(int arr[], int n)
{
  int i, j;
  for (i = 0; i < n-1; i++)
      for (j = 0; j < n-i-1; j++)
          if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}
```

```
BubbleSort(A)
  n = A.length
  for i = 1 to n − 1
      for j = 1 to n − i
          if A[j] > A[j + 1]
              swap  A[j] and A[j + 1]
```

CORRECTNESS OF AN ALGORITHM
PRACTICE

- Finding the ***loop invariant*** of BubbleSort (A).

```
BubbleSort(A)
    n = A.length
    for i = 1 to n − 1
        for j = 1 to n − i
            if A[j] > A[j + 1]
                swap  A[j] and A[j + 1]
```

  - List one or two iterations on an input instance.
  - Find a property (subarray that stays sorted) for each iteration.

Array elements $A[n - i + 1 \,..\, n]$ are in place

NEXT UP
ANALYZE THE EFFICIENCY

REFERENCES

- https://www.geeksforgeeks.org/bubble-sort/