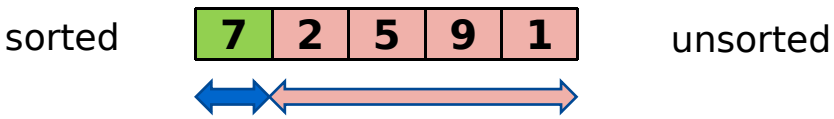
Design and Analysis of Algorithm

Loop Invariants

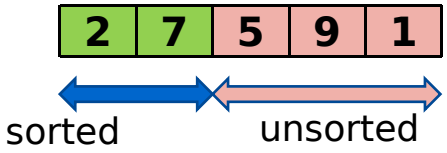
Insertion sort

- Applied in practice for sorting small lists of data.
- List is imaginary divided into two parts - sorted one and unsorted one.



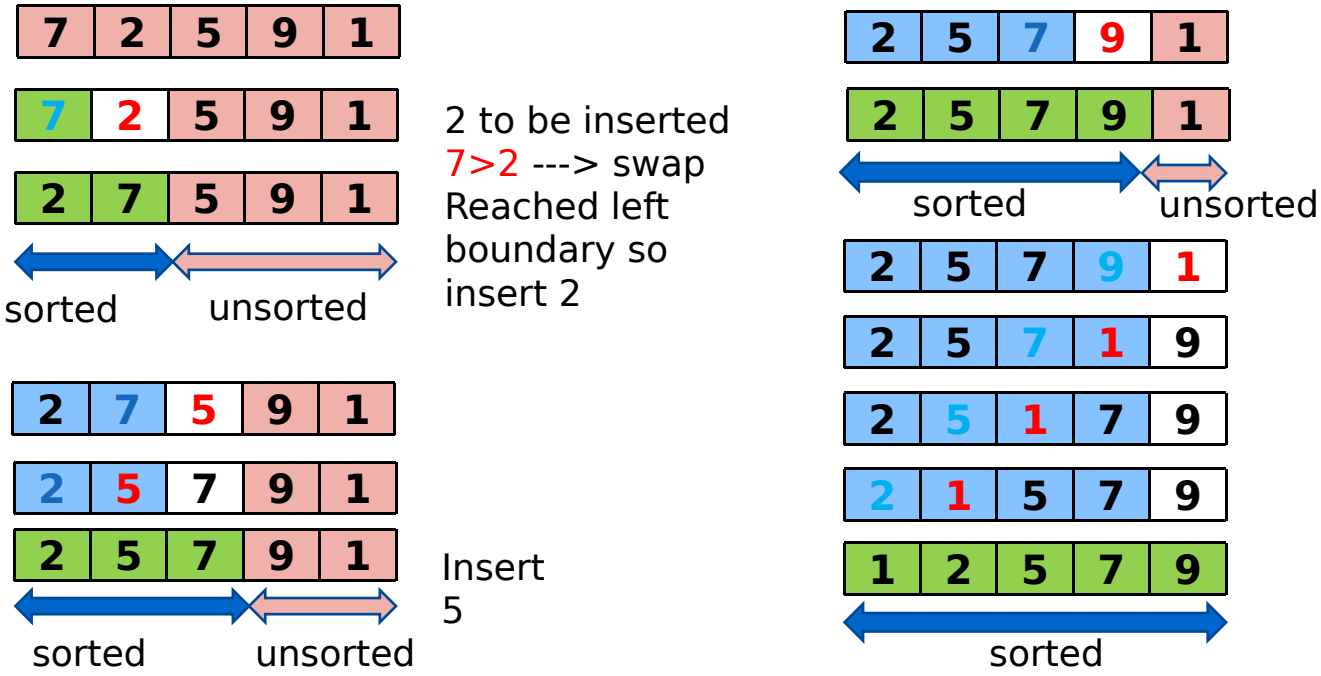
Insertion sort

- At the beginning, sorted part contains first element of the array and unsorted one contains the rest.
- At every step, algorithm takes first element in the unsorted part and inserts it into its correct position in the sorted one.



- When unsorted part becomes empty, algorithm stops.

Example – insertion sort



Example – insertion sort

6 5 3 1 8 7 2 4

Insertion sort, pseudo-code

$$
\begin{aligned}
&for\ i = 1 \rightarrow n: \\
&\quad j = i \\
&\quad while\ j > 0\ and\ list_{j-1} > list_j: \\
&\quad\quad swap(list_j, list_{j-1}) \\
&\quad\quad j--
\end{aligned}
$$

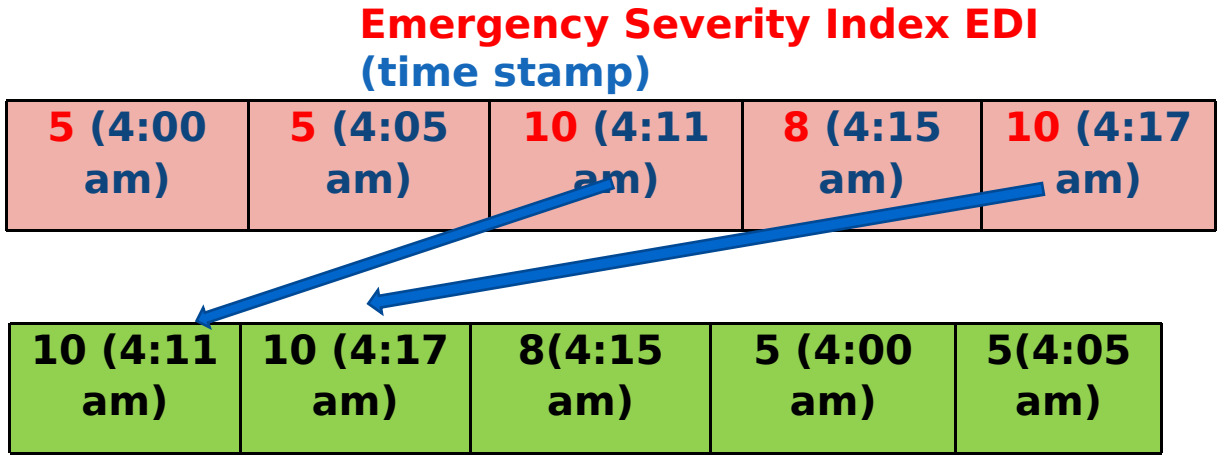
About Insertion Sort

- Inefficient for large data
  - O(n2) sorting algorithm (huge time proportional to size of the data)
  - Here the number of comparison made

    $1 + 2 + 3 + \ldots + (n - 1) = n(n - 1)/2$ ----> $O(n^2)$
- Stable
- Correct (Loop Invariant)
- In-place (No significant extra memory)
- Time is proportional to the number of the inversions
  - Better when list is almost sorted o(k+n) : k is inversions number.

Stability

- Example: The process of determining the next patient to be seen at Emergency Department (when resources are insufficient for all to be treated immediately)
- If two items have the same key as each other, they should have the same relative position in the output as they did in the input.



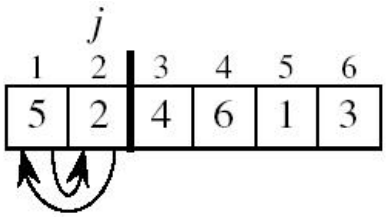
Loop Invariant

- Proving loop invariants works like mathematical induction
- **Initialization (base case):**
  - It is true prior to the first iteration of the loop
- **Maintenance (inductive step):**
  - If it is true before an iteration of the loop, it remains true before the next iteration
- **Termination:**
  - When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct
  - Stop the induction when the loop terminates

Initialization

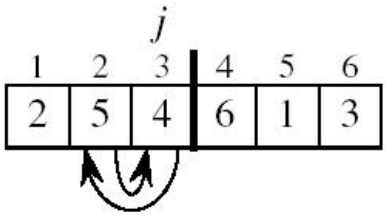
– Just before the first iteration, j = 2:

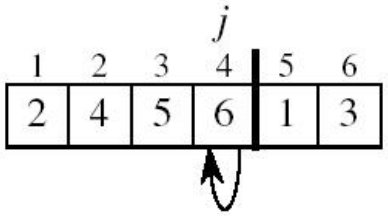
the subarray A[1 . . j-1] = A[1], (the element originally in A[1]) – is sorted



Maintenance

- The while inner loop moves A[j -1], A[j -2], A[j -3], and so on, by one position to the right until the proper position for key (which has the value that started out in A[j]) is found
- At that point, the value of key is placed into this position.





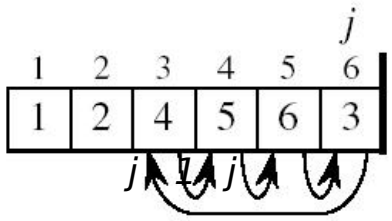
Termination

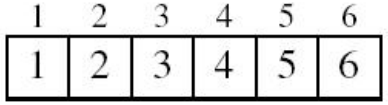
- The outer for loop ends when j = n + 1 j-1 = n

– Replace n with j-1 in the loop invariant:

  - the subarray A[1 . . n] consists of the elements originally in A[1 . . n], but in sorted order

- The entire array is sorted!
- Invariant: at the start of the for loop the elements in A[1 . . j-1] are in sorted order





Loop Invariant: Selection Sort

- Proving loop invariants works like mathematical induction
- **Initialization (base case):**
  - Before iteration 2 the elements from A[0:1] are sorted
- **Maintenance (inductive step):**
  - Selection of min/max from A[i:end] and sort that while A[0:i] is sorted
- **Termination:**
  - The outer loop ends when there is not element left unsorted i.e. i=end

Loop Invariant: Bubble Sort

- Proving loop invariants works like mathematical induction
- **Initialization (base case):**
  - Before iteration 2 the elements from A[end-1:end] are sorted
- **Maintenance (inductive step):**
  - Selection of min/max from A[0:i] and sort that while A[end-i:end] is sorted
- **Termination:**
  - The outer loop ends when there is not element left unsorted i.e. end-i=0

Loop Invariant: Merge Sort

- Proving loop invariants works like mathematical induction
- **Initialization (base case):**
  - A contains a single element (which is trivially “sorted”)
  - A[p] is the smallest element of L and R

Loop Invariant: Merge Sort

- **Maintenance (inductive step):**
  - Assume that Merge satisfy the loop invariant property until k
  - Next value to be inserted is the smallest one remaining in L and R
  - This value is large than those previously inserted in A
  - Loop invariant property satisfied for k+1

Loop Invariant: Merge Sort

- **Termination:**
  - Merge terminates when k=r, thus when r-p+1 elements have been inserted in A
    ⇒ All elements are sorted

Loop Invariant: Merge Sort

- Proving loop invariants works like mathematical induction
- **Initialization (base case):**
  - A contains a single element (which is trivially “sorted”) A[p]
- **Maintenance (inductive step):**
  - Assumption for k and prove for k+1
- **Termination:**
  - Merge terminates when k=r, thus when r-p+1 elements have been inserted in A ⇒ All elements are sorted

4.6 Master Theorem

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$
- $T(n) = a^2T\left(\frac{n}{b^2}\right) + af(n/b) + f(n)$
- $T(n) = a^3T\left(\frac{n}{b^3}\right) + a^2f(n/b^2) + af(n/b) + f(n)$
- $T(n) = a^3T\left(\frac{n}{b^3}\right) + \sum_{j=0}^{3-1} a^j f(n/b^j)$
- …
- $T(n) = a^iT\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f(n/b^j)$

4.6 Master Theorem

- $T(n) = a^i T\left(\frac{n}{b^i}\right) + \sum_{j=0}^{i-1} a^j f(n/b^j)$
- $f(n) = n^{\log_b a - \epsilon}$