PH502: Scientific Programming Concepts

Irish Centre for High End Computing (ICHEC)

September 23, 2020

Overview

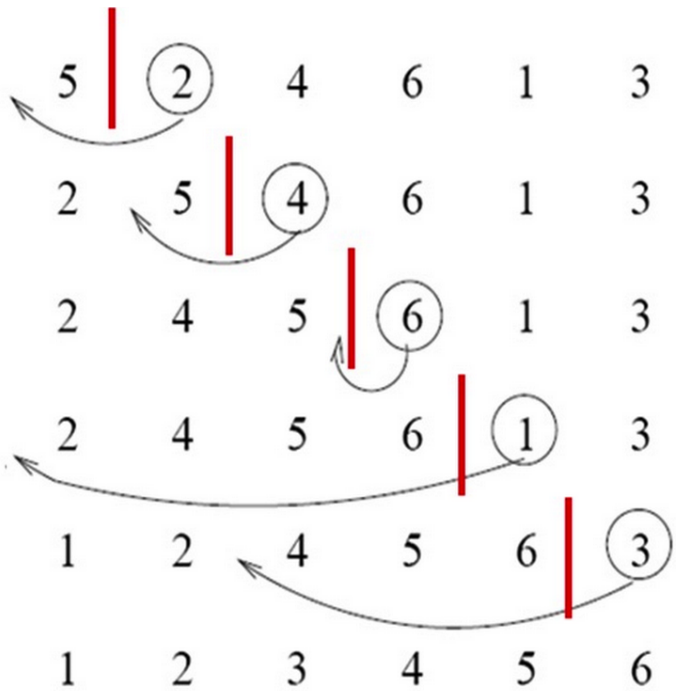
- If the last lecture was a bit theoretical we will continue with some examples.

Example: Insertion Sort

Input: Sequence of numbers: $A = \{a_1, a_2, ..., a_n\}$

Output: Permutation of this numbers: $a_1\prime, a_2\prime, ..., a_n\prime$ such that $a_1\prime \leq a_2\prime \leq ... \leq a_n\prime$

```
for j ← 2 to n do
    key ← A[j]
    i ← j − 1
    while i > 0 and A[i] > key do
        swap A[i + 1] and A[i]
        i ← i − 1
    end while
    A[i + 1] ← key
end for
```



Analysis of Insertion Sort

- Worst-Case: Reverse Sorted

$$\sum_{j=2}^{n} \Theta(n) = O(n^2)$$

- Best-Case: Already Sorted list

$$\sum_{j=2}^{n} \Theta(1) = \Omega(n)$$

For small n, moderately fast. Not at all for large n. Use *Heap Sort* or *Merge Sort* for larger n.
(`http://en.wikipedia.org/wiki/Sorting_algorithm`)

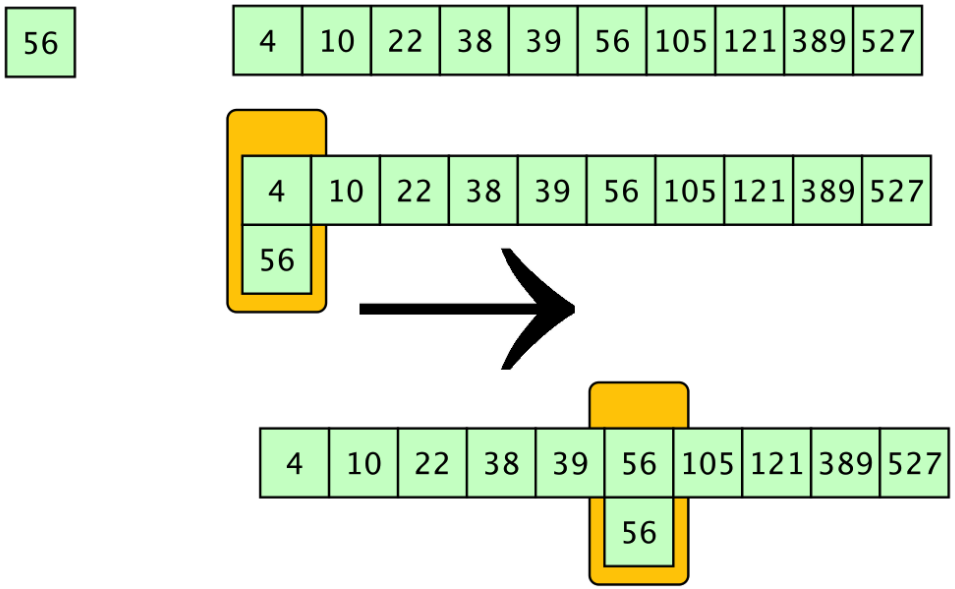
Example: Linear Search

Input: A list of unsorted integers $A = \{a_1, a_2, ..., a_n\}$, A single integer to search for: k

Output: True is the integer is found; False otherwise.

Procedure: Scan the list from beginning to end, comparing the integer to find with every single element.

```
for i ← 1 to n do
    if k=A[i] then
        return TRUE
    end if
end for
return FALSE
```



Analysis of Linear Search

- Worst-Case: Element searched in the last one

$$O(n)$$

- Best-Case: element searched is the first

$$\Omega(1)$$

Binary search is faster (Worst case: $O(logn)$) however it requires the list to be sorted.
(`http://en.wikipedia.org/wiki/Binary_search_algorithm`)

Three Classes

- Three classes of problems:
  1. **P:** solve the problem in polynomial-time. Ex.: Most searching and sorting algorithms
  2. **NP:** solving problem is slower than P but solution is verifiable in polynomial-time. Ex. traveling salesman problem.
  3. **NPC:** (complete) in solving one, of a set of NP problems, the solution is transferrable in P.