Computing Time – Insertionsort

Textbook Reading:

Chapter 2, Subsection 2.6.5 Pages 41-46.

Chapter 3, Subsection 3.6.3 Pages 117-118.

Insertion Sort – recursive strategy

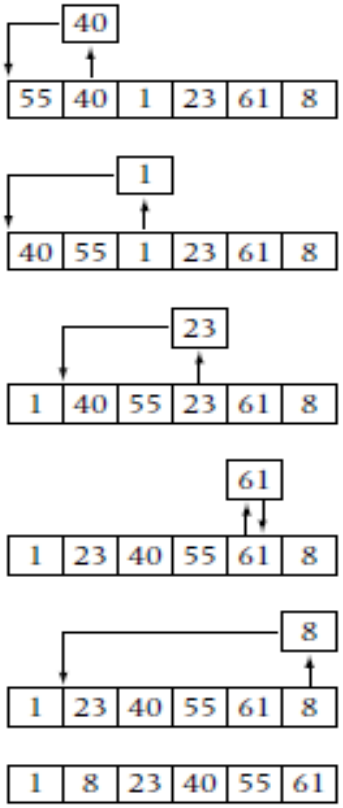
- Start with the a priori sorted sublist consisting of $L[0]$.

- Perform $n - 1$ passes, $i = 1, 2, \ldots, n - 1$.
  - insert $L[i]$ in its sorted position in already sorted sublist $L[0:i - 1]$.

Pseudocode for Insertion Sort where list is implemented using an array

```
procedure InsertionSort(L[0:n – 1])
Input:       L[0:n – 1] (a list of size n)
Output:   L[0:n – 1] (sorted in increasing order)
        for i ← 1 to n – 1 do //insert L[i] in its proper position in L[0:i – 1]
                    Current ← L[i]
                    position ← i – 1
                while position ≥ 0 .and. Current < L[position] do
                                                          //Current must precede L[position]
                        L[position +1] ← L[position]          //bump up L[position]
                        position ← position – 1
                endwhile
              //position + 1 is now the proper position for Current = L[i]
                     L[position +1] ← Current
              endfor
end InsertionSort
```

Action of Insertion Sort for sample list
55 40 1 23 61 8
where list is implemented using an array



Best-case analysis of Insertion Sort

- Take basic operation to be comparison. The best-case complexity $B(n)$ is realized for an already sorted list L. Since $L[i]$ is larger than any element in $L[0{:}i-1]$, to insert it into $L[0{:}i-1]$ involves only making a single comparison.
- Since a single comparison is done to insert for $i = 1, 2, \ldots, n-1$, the total number of comparisons performed is $n-1$. Thus,

$$B(n) = n - 1.$$

Worst-case analysis of Insertion Sort

- The worst-case complexity $W(n)$ is realized for list sorted in reverse order.
- To insert the i^{th} element in the already sorted sublist $L[0:i-1]$ involves a i comparisons, i.e., $L[i]$ is compared with every element in the sublist $L[0:i-1]$, $i = 1, 2, \ldots, n-1$.
- It follows that the total number of comparisons performed is given by

$$W(n) = 1 + 2 + \ldots + n - 1 = n^2/2 - n/2.$$

Average Complexity

We first compute the average complexity to insert *L*[*i*] in its sorted position in the already sorted sublist *L*[0:i – 1]. To motivate, let's first consider an example with inserting a sixth element into a list of 5 elements.

__10__ 23__ 30__ 70__81__

- For this sample sublist L[0:4] of size 5, there are 6 positions to insert L[5] shown above.
- The number of comparisons inserting in last position is 1, the second last position 2, third from last position 3, the forth from last position 4, the fifth from last (second) position 5 and the sixth from last (first) position also 5, but for convenience we will approximate it with 6.
- Since, we are assuming a uniform distribution the probability of inserting into any given position is $\frac{1}{6}$. Therefore, the expected number of comparisons to insert *L*[5] is

$$= \sum_{i=1}^{6} i \frac{1}{6}$$

$$= \tfrac{1}{6} \times (1 + 2 + \ \ldots + 6) = \tfrac{1}{6} \times \ \frac{6 \times 7}{2} = \frac{7}{2}.$$

Expected number of comparisons to insert $L[i]$

- There are i positions to insert $L[i]$ into the sorted sublist $L[0{:}i-1]$.
- The number of comparisons inserting in the j^{th} from last position is j, $j = 1, 2, \ldots, i-1$, and the number of comparison to insert in the first position is also $i-1$, but for convenience, we approximate it with i.
- Since, we are assuming a uniform distribution the probability of inserting into any given position is $1/i$.
- Therefore, the expected number of comparisons to insert $L[i]$ is

$$\frac{1}{i} \times 1 + \frac{1}{i} \times 2 + \ldots + \frac{1}{i} \times i = \frac{1}{i} \times (1 + 2 + \ldots + i) = \frac{1}{i} \times \frac{i(i+1)}{2} = \frac{i+1}{2}.$$

Computing $A(n)$

$A(n)$ = sum of the number of comparisons to insert $L[i]$, $i = 1, \ldots, n-1$.

Using the result from previous slide

$$A(n) = \sum_{i=1}^{n} \frac{i+1}{2}$$

$$= \frac{1}{2} \sum_{2}^{n+1} i = \frac{1}{2} \frac{(n+1)(n+2)}{2} - 1$$

$$= \frac{n^2}{4} + \frac{3n}{4} - \frac{1}{2}$$

It follows that average complexity of Insertion Sort is about half its worst-case complexity.

- Insertion Sort is **quadratic** on average and in the worst-case, which is not optimal compared to other sorting algorithms such as Mergesort. However, it is linear in the best-case and when the list is close to being sorted, which is better than Mergesort.
- Insertion is effective as an **online** algorithm, i.e., if the list elements are received over time.
- For example, if a class of students has already been sorted, another student can be added to the class in linear time. This is not true of other algorithms like Mergesort and Quicksort, which we will discuss later. They would require resorting the entire list.

Implementing Insertion Sort using a linked list

PSN. Show pictorially the action of Insertionsort implemented using a linked list for the same sample list 55 40 1 23 61 8.

Array vs. Linked List implementations

- In the array implementation we scan the list from back to front to determine the position to insert, but with the linked list implementation we scan from front to back.
- $B(n)$, $A(n)$, $W(n)$ remain the same using the linked list implementation.
- In the case of an array $B(n)$ is realized for a sorted list and $W(n)$ for a list in reverse order. The opposite is true for the linked list implementation.

Why should you never use a tool for sorting big and small fences?

Answer:
It's a fence-sieve