10. Insertion Sort

Insertion Sort

It is said that insertion sort is how many sort a hand of cards. An empty hand is sorted. A single, first card, is sorted. As every other card is dealt, insert it into the previously sorted hand.

The functional `insertion_sort` code below uses a helper function to perform the insertion.

```haskell
Listing 28: Functional Insertion Sort
⟨Functional insertion sort 115a⟩≡
insertion_sort :: Ord a => [a] -> [a]
insertion_sort [] = []
insertion_sort [x] = [x]
insertion_sort (x:xs) = insert (insertion_sort xs)
where insert [] = [x]
insert (y:ys) | x <= y = x : y : ys
| otherwise = y : insert ys
```

An imperative implementation is given below. It assumes a sentinel, `A[0]`, that is smaller every other element in the array. The invariant of the algorithm is that after the `i`th step, the array `A[0..i]` is sorted.

```c
Listing 29: Imperative Insertion sort
⟨Insertion sort 115b⟩≡
void InsertionSort(int A[], int n) {
    int i, j, v;
    for (i = 2; i < n; i++) {
        v = A[i];
        j = i;
        while (A[j-1] > v) {
            A[j] = A[j-1];
            j = j - 1;
        }
        A[j] = v;
    }
}
```

To insert sort a list `x:xs`,
- Insert sort the tail `xs`.
- Insert `x` at the head if it is smaller than the head of the insertion sorted list `(y:ys)`.
- Otherwise, leave `y` as the head of the sorted list and insert `x` in the tail `ys`. 
Insertion Sort – Analysis of Complexity

The time complexity of the imperative algorithm can be computed using these observations:

- The outer for loop on i is executed n – 1 times
- There are two initial assignment in this outer loop, a while loop, and a final assignment
- The comparison in the while loop Boolean expression may execute as many as i times and as few as 0 times
- In the best case (the data is sorted in ascending order), insertion sort will execute 3n – 3 assignments and n – 1 tests of the Boolean expression in the while loop. Thus, the time complexity is O(n)
- In the worst case (the data is in descending order), the while loop executes i times making i evaluations of the Boolean expression and 2 assignments on each pass of the loop. This occurs for every value of i from 2 to n, thus the total complexity is

\[
3n - 3 + \sum_{i=2}^{n} 3i = 3n - 3 + 3 \left[ \frac{n(n + 1)}{2} - 1 \right]
\]

The time complexity is O(n²)
- For the average case, we need the probability that k compares are made in the while test for k = 1 to i
- For given k = 1, ..., i there will be k compares if and only if

\[
\]

and

\[
A[i - k] \leq A[i]
\]

That is, A[i] is the kth largest element in the array

\[
A[1], A[2], ..., A[i - 1], A[i]
\]

- The probability of this is 1/i: There is one out of i positions to place the kth largest element
• Thus the average number of comparisons is

\[
\sum_{i=2}^{n} \sum_{k=1}^{i} \frac{1}{i} = \sum_{i=2}^{n} \frac{i+1}{2} = \frac{(n+1)(n+2)}{4} - \frac{3}{2}
\]

• The average case complexity is \(O(n^2)\)

<table>
<thead>
<tr>
<th>Example: Insertion sort operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i = 2)</td>
</tr>
<tr>
<td>-----------</td>
</tr>
<tr>
<td>0 2 5 7</td>
</tr>
<tr>
<td>0 2 7 5</td>
</tr>
<tr>
<td>0 5 2 7</td>
</tr>
<tr>
<td>0 5 7 2</td>
</tr>
<tr>
<td>0 7 2 5</td>
</tr>
<tr>
<td>0 7 5 2</td>
</tr>
</tbody>
</table>

For \(i = 2\), there are \(\frac{9}{6}\) comparisons, on average
For \(i = 3\), there are \(\frac{12}{6}\) comparisons, on average
The average number of comparisons, over all \(i\), is

\[
21/6 = 7/2 = \frac{(3+1)(3+2)}{4} - \frac{3}{2} = 5 - \frac{3}{2}
\]

Exercises

1. Consider the selection sort algorithm below. Explain the code and analyze its time complexity.

Listing 30: Functional Selection Sort

\[
\text{select\_sort : (Ord a) => [a] -> [a]}
\]

\[
\text{select\_sort [] = []}
\]

\[
\text{select\_sort xs = let x = maximum xs}
\]

\[
\text{in select\_sort (remove x xs) ++ [x]}
\]

\[
\text{where remove _ [] = []}
\]

\[
\text{remove a (x:xs)}
\]

\[
\mid x == a = xs
\]

\[
\mid \text{otherwise} = x : \text{remove a xs}
\]

2. Consider the selection sort algorithm below.
Listing 31: Imperative selection sort

```c
Imperative selection sort ≡
void selectionsort(int A[], int n)
{
    int i, j, min;
    for (i = 1; i < n; i++) {
        min = i;
        for (j = i+1; j < n; j++) {
            if (A[j] < A[min]) {
                min = j;
                swap(A[min], A[i]);
            }
        }
    }
}
```

(a) Compute the average number of swaps on the 6 permutations of \{2, 5, 7\}. Construct a table, similar to the table in example to show your computations.

(b) Find a general formula for the average number of comparisons in the insertion sort algorithm.
Bibliography


Corman, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, third edition. [page 9], [page 14], [page 18], [page 21], [page 31], [page 33], [page 39], [page 53], [page 93], [page 105], [page 135], [page 141], [page 163], [page 169], [page 178], [page 187], [page 193], [page 201]


Index

Haskell programming language, 20, 31, 44, 116, 137
Java programming language, 31

Ackermann function, 174
Axioms
  Peano, 192
Bézout, Étienne, 47

C programming language, 30, 31, 42, 43, 55, 109, 152, 153
Calendar, 7
Cartesian product, 142
Catalan numbers, 156, 161
Computability, 191
Computable function, 28
Computer Language Benchmark Game, 30
Contradiction, 193

Dijkstra’s algorithm, 183
DNA, 146

Empty
  String λ, 61

Fibonacci, 44, 46, 142
Floating point
  Machine epsilon, 50
Function
  Ackermann, 174
Functional programming, 31

Gödel, Kurt, 192
Golden ratio, 142
Greatest common divisor, 47
Gutenberg Project, 84

Hash Tables, 93
Hofstadter, Douglas, 25

Kleene, Stephen, 53
Knuth, Donald, 30

Lamé, Gabriel, 44
λ calculus, 26, 28
Linear congruence, 47
Listing
  C
    Matrix Chain Multiplication, 158
Listings
  C
    A Minimum Algorithm, 135
    Aho-Corasick, 89
    Boyer-Moore Pattern Matching, 72
    Brute-force Left-to-Right Pattern Matching, 56
    Bubble Sort, 109
    Build a Heap, 126
    Counting Sort, 128
    Direct Address Table Operations, 94
    Edit Distance, 152
    Find Set with Path Compression, 177
    Greatest Common Divisor, 42
    Hash Table with Chaining, 95
    Heap Sort, 127
    Heapify a List, 126
    Inner Product, 155
    Insertion Sort, 112
    Knuth–Morris–Pratt Pattern Matching, 67
    Maximum Subsequence Sum (Cubic), 34
    Maximum Subsequence Sum (Linear), 35
    Merge Sort, 118
    Morris–Pratt Pattern Matching, 58
    Partition about a pivot, 120
    Quick Sort, 120
    Radix Exchange Sort, 132
    Radix Sort, 129
    Randomized Partition, 136
    Randomized Selection, 138
    Rational Knapsack, 167
    Selection Sort, 114
    Shell Sort, 116
    Union by rank, 177

Haskell
  Ackermann function, 174
  Apply a Function Repeatedly, 50
  Brute-force Left-to-Right Pattern Matching, 57
  Bubble Function, 108
  Bubble Sort, 109
  Extended Euclidean Algorithm, 48
  Fold from the left, 37
  Greatest Common Divisor, 44
  Inner Product, 155
  Insertion Sort, 112
  Matrix Multiplication, 155
  Maximum Subsequence Sum, 37
  Merge Sort, 117
  Merging Sorted Lists, 117
  Newton’s Square Root Method, 51
  Newton’s Square Root Recurrence, 50
  Quicksort, 120
  Randomized Partition, 137
  Relative Error Convergence Test, 50
  Sample Code Header, 31
  Second of a pair, 36
  Selection Sort, 114
  Sorted Decision Problem, 106
Splitting a List, 117
Swapping head with Another Element, 137

Pseudocode
Activity Selection, 170
Bucket Sort, 130
Dijkstra’s Algorithm, 186
Kruskal Minimum Spanning Tree, 172
Prim Minimal Spanning Tree, 181

Lucas, Édouard, 44
Memorization, 141
Metric space, 152
Newton’s method, 13
Newton, Issac, 49
Noweb, 30, 55

Object-oriented programming, 31

Pattern Matching, 53, 54
Boyer-Moore, 71
Brute Force, 55

Knuth–Morris–Pratt, 67
Morris–Pratt, 58
Right-to-Left Brute Force, 70
ϕ, 142
Presburger arithmetic, 191

RAM, 26, 29

Sequence, 19
Lucas, 44
Sets
Cartesian product, 142
Simpson’s rule, 51

Sorting Algorithms
Bubble Sort, 107
Bucket Sort, 129
Counting Sort, 128
Heap Sort, 124
Insertion Sort, 111
Merge Sort, 116
Quick Sort, 119
Radix Sort, 129
Selection Sort, 114
Shell Sort, 115

Time complexity, 127

String
Border, 61
Period, 61
Prefix, 61
Suffix, 61

Syllabus, 5

Theorems
Bezout’s, 47
Euclidean division, 41
Lamé’s, 45
Taylor’s, 49, 51

Turing
Machine, 9, 26, 28, 192

Wikipedia, 201

xkcd
Algorithms by Complexity, 25
Calendar of meaningful dates, 7
Haskell, 30
Ineffective Sorts, 105
Traveling Salesman Problem, 141