Ch. 2 Introduction

Insertion Sort

```
Insertion-Sort(A, n)
  for j <- 2 to n do
    key <- A[j]
    i <- j - 1
    while i > 0 and A[i] > key do
      A[i + 1] <- A[i]
      i <- i - 1
    A[i + 1] <- key
```

- `A[k]` for `k` $<$ `key` are sorted
- Insert `A[key]` to the right place in `A[k]` for `k` $<$ `key`

Correctness

- "An alg is *correct*" $\equiv$
  "It *halts* w/ the *correct output* for every input instance"

Loop Invairants

- Conds and relps satisfied by the vars and ds at the end of *every iteration* of the loop
- Explain why an alg is correct
- We need to show 3 things about a loop invariant
  - *Initialization*: True prior to the 1st iteration
  - *Maintenance*: If true before an iteration, also true before the next iteration
  - *Termination*

Correctness of Insertion Sort

- Loop invariant
  - At the start of each iteration of the loop, `A[1..j - 1]` consists of the elems orginally in `A[1..j - 1]` but in sorted order.
- Initialization
  - When `j = 2`, `A[1..j - 1]` $\equiv$ `A[1]`
- Maintenance
  - Move `A[j - 1]`, `A[j - 2]`, ... by one pos to the right until the proper pos for `A[j]` is found
- Termination
  - Ends when `j = n + 1`, `A[1..n]` consists of the elems originally in `A[1..n]` but in sorted order

Kinds of Analysis on Running Time

1. Worst-case (usually)

    - $T(n)$: Max time of alg on any input of size n
    - Usually interested in because
        - Gives an upper bound
        - Occurs often for some algs
        - Avg case is often bas as the worst case
2. Avg-case (sometimes)
    - $T(n)$: Expected time of alg over all inputs of size n
3. Best-case (bogus)

Asymptotic Analysis

- Look only at the leading term of the formula for running time
    - $an^2 + bn + c$ $\rightarrow$ $\Theta(n^2)$
    - Ignore machine-dependent constants
- "One alg is more efficient than another" $\equiv$
  "Its worst-case running time has a smaller order of growth"
- $T(n)$: *Basic computer steps* needed in the alg
    - Basic computer steps: branching, loading, storing, comparisons, simple arithmetic, ...
    - Assumption: Basic computer steps take a constant amount of time.
    - $T(n) = \Theta(g(n))$: $T(n) \in \Theta(g(n))$, technically

Θ-Notation

- $\Theta(g(n))$ $\equiv$
  ${~ f(n) ~|~ \exist c_1,~ c_2,~ n_0 ~\text{s.t.}~ 0 \leq c_1 g(n)\leq f(n) \leq c_2 g(n) ~\text{for}~ \forall n \geq n_0 ~}$
    - $g(n)$: Asymptotic tight bound for $f(n)$

Insertion Sort Analysis

Worst Case

- Input reverse sorted
- $T(n) = \sum_{j = 2}^{n} \Theta(j) = \Theta(n^2)$

Avg Case

- All permutations equally likely
- $T(n) = \sum_{j = 2}^{n} \Theta(j / 2) = \Theta(n^2)$

Fibonacci Numbers

- Recursive definition
  $ F_n = \begin{cases} 0 & \text{if}~ n = 0 \ 1 & \text{if}~ n = 1 \ F_{n - 1} + F_{n - 2} & \text{if}~ n \geq 2 \end{cases} $

- Naive recursive algorithm

```
function fib1(n)
  if n = 0 do
    return 0
  if n = 1 do
    return 1
  return fib1(n - 1) + fib1(n - 2)
```

- Analysis
  $ T(n) \begin{cases} \leq 2 & \text{for}~ n \leq 1 \ = T(n - 1) + T(n - 2) + 3 & \text{for}~ n > 1 \end{cases} $

  - $T(n) \geq F_n$
  - $F_n \approx 2^{0.694n}$
  - $T(n)$ is *exponential* in n

- Better algorithm

```
function fib2(n)
  if n = 0 do
    return 0
  f[0...n]
  f[0] <- 0
  f[1] <- 1
  for i <- 2 to n do
    f[i] <- f[n - 1] + f[n - 2]
  return f[n]
```

- Analysis
  - `f[n - 1] + f[n - 2]` doesn't take a constant time as they are not small
  - $T(n) = n \times \Theta(n) = \Theta(n^2)$

Asymptotic Growth

- $c < \log(n) < \sqrt{n} < n < n \log(n) < n^2 < n^3 < c n^c < n^{\log(n)} < 2^{n} < 2^{2^n} < 2^{2...^{2^n}}$

Θ-Notation

- $\Theta(g(n))$ $\equiv$
  ${~ f(n) ~|~ \exist c_1,~ c_2,~ n_0 > 0 ~\text{s.t.}~ 0 \leq c_1 g(n)\leq f(n) \leq c_2 g(n) ~\text{for}~ \forall n \geq n_0 ~}$
  - $g(n)$: Asymptotic tight bound for $f(n)$
- $\lim_{n \rightarrow \infin} f(n)/g(n) = c, ~c \in \R^+$ $\rightarrow$ $f \in \Theta(g)$
- $n^2 / 2 - 2n = \Theta(n^2)$

O-Notation

- $O(g(n))$ $\equiv$ ${~ f(n) ~|~ \exist c ,~ n_0 > 0 ~\text{s.t.}~ 0 \leq f(n) \leq c g(n) ~\text{for}~ \forall n \geq n_0 ~}$
  - $g(n)$: Asymptotic upper bound for $f(n)$
- $\lim_{n \rightarrow \infin} f(n)/g(n) = c, ~c \in \R^*$ $\rightarrow$ $f \in O(g)$
  - $\R^*$: Set of non-negative real numbers
- $2n^2 = O(n^3)$

Ω-Notation

- $O(g(n))$ $\equiv$ ${~ f(n) ~|~ \exist c ,~ n_0 > 0 ~\text{s.t.}~ 0 \leq c g(n) \leq f(n) ~\text{for}~ \forall n \geq n_0 ~}$
  - $g(n)$: Asymptotic lower bound for $f(n)$
- $ \lim_{n \rightarrow \infin} f(n)/g(n) = \begin{cases} c > 0 \ \infin \end{cases} $ $\rightarrow$ $f \in \Omega(g)$
  - $\R^*$: Set of non-negative real numbers
- $\sqrt{n} = \Omega(\lg(n))$

o-Notation

- $o(g(n))$ $\equiv$ ${~ f(n) ~|~ \forall c > 0,~ \exist n_0 > 0 ~\text{s.t.}~ 0 \leq f(n) < c g(n) ~\text{for}~ \forall n \geq n_0 ~}$
  - $f(n)$ is asumptotically smaller than $g(n)$
- $\lim_{n \rightarrow \infin} f(n)/g(n) = 0$ $\rightarrow$ $f \in o(g)$
- $10^{10} n^2 + 10^5 n + 10^9 = o(n^3)$

ω-Notation

- $o(g(n))$ $\equiv$ ${~ f(n) ~|~ \forall c > 0,~ \exist n_0 > 0 ~\text{s.t.}~ 0 \leq c g(n) < f(n) ~\text{for}~ \forall n \geq n_0 ~}$
  - $f(n)$ is asumptotically larger than $g(n)$
- $\lim_{n \rightarrow \infin} f(n)/g(n) = \infin$ $\rightarrow$ $f \in \omega(g)$
  - $\R^*$: Set of non-negative real numbers
- $n^3 - 9 = \omega(n)$

Theorems

- $g(n) = o(f(n))$ $\leftrightarrow$ $f(n) = \omega(g(n))$
- $f(n) = \Theta(g(n))$ $\leftrightarrow$ $f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$
- $\log(n) = o(n^\alpha)$ for any $\alpha >0$
- $n^k = o(2^n)$ for any $k > 0$

> There is exercise in the lecture note!!