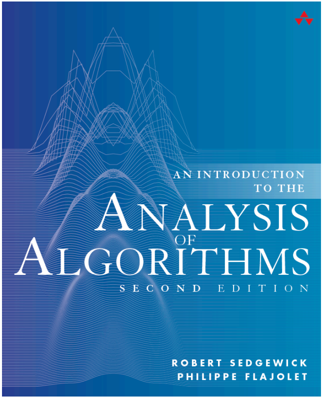
Asymptotics, Recurrences, and Divide and Conquer

Hengfeng Wei

hfwei@nju.edu.cn

April 17, 2018


AN INTRODUCTION TO THE
ANALYSIS OF ALGORITHMS
SECOND EDITION
ROBERT SEDGEWICK
PHILIPPE FLAJOLET

Problem P Algorithm A

Problem P Algorithm A

Inputs: $\mathcal{X}_n$ of size n

Problem P Algorithm A

Inputs: $\mathcal{X}_n$ of size n

$$W(n) = \max_{X \in \mathcal{X}_n} T(X)$$

Problem P Algorithm A

Inputs: $\mathcal{X}_n$ of size n

$$W(n) = \max_{X \in \mathcal{X}_n} T(X)$$

$$B(n) = \min_{X \in \mathcal{X}_n} T(X)$$

Problem P Algorithm A

Inputs: $\mathcal{X}_n$ of size n

$$W(n) = \max_{X \in \mathcal{X}_n} T(X)$$

$$B(n) = \min_{X \in \mathcal{X}_n} T(X)$$

$$A(n) = \sum_{X \in \mathcal{X}_n} T(X) \cdot P(X)$$

Problem P Algorithm A

Inputs: $\mathcal{X}_n$ of size n

$$W(n) = \max_{X \in \mathcal{X}_n} T(X)$$

$$B(n) = \min_{X \in \mathcal{X}_n} T(X)$$

$$A(n) = \sum_{X \in \mathcal{X}_n} T(X) \cdot P(X) = \mathop{\mathbb{E}}_{X \in \mathcal{X}_n} [T(X)]$$

Average-case Time Complexity (Problem 1.8)

$$\text{Input} : r \in [1, n], \;\; r \in \mathbb{Z}^+$$

$$P\{r = i\} = \begin{cases} \frac{1}{n}, & 1 \le i \le \frac{n}{4} \\ \frac{2}{n}, & \frac{n}{4} < i \le \frac{n}{2} \\ \frac{1}{2n}, & \frac{n}{2} < i \le n \end{cases}$$

Average-case Time Complexity (Problem 1.8)

$$\text{Input}: r \in [1, n], \quad r \in \mathbb{Z}^+$$

$$P\{r = i\} = \begin{cases} \frac{1}{n}, & 1 \le i \le \frac{n}{4} \\ \frac{2}{n}, & \frac{n}{4} < i \le \frac{n}{2} \\ \frac{1}{2n}, & \frac{n}{2} < i \le n \end{cases}$$

$$\begin{aligned} A &= \sum_{X \in \mathcal{X}} T(X) \cdot P(X) \\ &= T(1)P(1) + T(2)P(2) + \cdots + T(n)P(n) \\ &= \frac{n}{4} \times 10 \times \frac{1}{n} + \frac{n}{4} \times 20 \times \frac{2}{n} + \frac{n}{4} \times 30 \times \frac{1}{2n} + \frac{n}{4} \times n \times \frac{1}{2n} \\ &= \frac{1}{8}n + \frac{65}{4} \end{aligned}$$

Average-case Analysis of Quicksort

$$A(n) = n - 1 + \frac{1}{n} \sum_{i=0}^{i=n-1} \left(A(i) + A(n-i-1) \right)$$

$$A(n) = \mathop{\mathbb{E}}_{X \in \mathcal{X}_n} [T(X)] = \sum_{X \in \mathcal{X}_n} T(X) \cdot P(X)$$

Average-case Analysis of Quicksort

$$A(n) = n - 1 + \frac{1}{n} \sum_{i=0}^{i=n-1} \left(A(i) + A(n-i-1) \right)$$

$$A(n) = \mathop{\mathbb{E}}_{X \in \mathcal{X}_n} [T(X)] = \sum_{X \in \mathcal{X}_n} T(X) \cdot P(X)$$

$$\begin{aligned}
A(n) &= \mathbb{E}[T(X)] \\
&= \mathbb{E}[\mathbb{E}[T(X) | I]] \\
&= \sum_{i=0}^{i=n-1} P(I = i) \, \mathbb{E}[T(X) \mid I = i] \\
&= \sum_{i=0}^{i=n-1} \frac{1}{n} [n - 1 + A(i) + A(n-i-1)]
\end{aligned}$$

Mathematical Induction

Horner's rule (Problem 1.5)

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

```
procedure HORNER(A[0...n], x)          ▷ A : {a_0 ... a_n}
    p ← A[n]
    for i ← n − 1 downto 0 do
        p ← px + A[i]
    return p
```

Horner's rule (Problem 1.5)

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

```
procedure HORNER(A[0...n], x)        ▷ A : {a_0 ... a_n}
    p ← A[n]
    for i ← n − 1 downto 0 do
        p ← px + A[i]
    return p
```

Loop invariant (after the k-th loop):

Horner's rule (Problem 1.5)

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n$$

```
procedure HORNER(A[0...n], x)                ▷ A : {a_0 ... a_n}
    p ← A[n]
    for i ← n − 1 downto 0 do
        p ← px + A[i]
    return p
```

Loop invariant (after the k-th loop):

$$\boxed{\mathcal{I}: \; p = \sum_{i=n}^{i=n-k} a_i x^{k-(n-i)}}$$

$$\boxed{\mathcal{I}:\ p = \sum_{i=n}^{i=n-k} a_i x^{k-(n-i)}}$$

$$\boxed{\mathcal{I}: \; p = \sum_{i=n}^{i=n-k} a_i x^{k-(n-i)}}$$

When you are in an exam:

- 20% : Finding $\mathcal{I}$
- 80% : Proving $\mathcal{I}$ by PMI

$$\boxed{\mathcal{I}: \ p = \sum_{i=n}^{i=n-k} a_i x^{k-(n-i)}}$$

Proof.

Prove by mathematical induction on non-negative integer k, the number of loops.

$$\boxed{\mathcal{I}: \; p = \sum_{i=n}^{i=n-k} a_i x^{k-(n-i)}}$$

Proof.

Prove by mathematical induction on non-negative integer k, the number of loops.

Basis:

$$k = 0 : p = a_n = \mathcal{I}_0$$

Inductive Hypothesis:

Inductive Step:

□

Integer Multiplication (Problem 1.6)

```
procedure INT-MULT(y, z)
    if z = 0 then
        return 0
    return INT-MULT(cy, ⌊z/c⌋) + y(z mod c)
```

$$\textbf{procedure } \textsc{Int-Mult}(y, z)$$
$$\textbf{if } z = 0 \textbf{ then}$$
$$\textbf{return } 0$$
$$\textbf{return } \textsc{Int-Mult}(cy, \lfloor \tfrac{z}{c} \rfloor) + y(z \mod c)$$

Integer Multiplication (Problem 1.6)

```
1: procedure INT-MULT(y, z)
2:     if z = 0 then
3:         return 0
4:     return INT-MULT(cy, ⌊z/c⌋) + y(z mod c)
```

Proof.

Prove by mathematical induction on non-negative integer z. $\square$

2-tree; full binary tree (Problem 2.5)

$$n_0 = n_2 + 1$$

Proof.

2-tree; full binary tree (Problem 2.5)

$$n_0 = n_2 + 1$$

Proof.

Prove by mathematical induction on *the structure of binary tree.*

$\square$