

2009 年度ソフトウェア工学 再 試験問題解答例

1. 漸近量評価

(1) 次の関数を O 記法で示せ。(出来るだけ漸近計算量が小さいもので表すこと。)

(10 点、各 2 点)

(a) $T_a(n) = 2n^3 - 5n^2 + 7n$

最高次数のみ注目し、係数を除けば (係数を 1 と見なせば) よい。

$$T_a(n) = O(n^3)$$

(b) $T_b(n) = 3(\sqrt[5]{n^3} - 2)(\sqrt[5]{n^2} + 3) + 7(\sqrt[3]{n} + 5)(\sqrt[3]{n} - 3)$

項別に評価すると良い。

$$\text{第 1 項 } O\left(n^{\frac{3}{5}}\right) \times O\left(n^{\frac{2}{5}}\right) = O\left(n^{\frac{3}{5} + \frac{2}{5}}\right) = O(n)$$

$$\text{第 2 項 } O\left(n^{\frac{1}{3}}\right) \times O\left(n^{\frac{1}{3}}\right) = O\left(n^{\frac{1}{3} + \frac{1}{3}}\right) = O\left(n^{\frac{2}{3}}\right)$$

よって、 $T_b(n) = O(n)$

(c) $T_c(n) = (\log n)^{10} + n^{\frac{1}{10}}$

対数とべき乗では、べき乗の方が漸近的な増加速度は大きい。

すなわち、どんな大きな数 $N > 1$ と小さい数 $\varepsilon > 0$ であっても、

$$O\left((\log n)^N\right) < O(n^\varepsilon)$$

が成り立つ。

よって、

$$T_c(n) = O\left(n^{\frac{1}{10}}\right)$$

(d) $T_d(n) = 8^{\log n} + n^2 + (\sqrt{n})^5$

第 1 項 :

$$8^{\log n} = 2^{3 \log n} = (2^{\log n})^3 = n^3 \text{ なので、第 1 項は } O(n^3)$$

第 3 項 :

$(\sqrt{n})^5 = n^{\frac{5}{2}} = n^{2.5}$ なので、第3項は $O(n^{2.5})$

以上より、

$$T_d(n) = O(n^3)$$

$$(e) \quad T_e(n) = \frac{(3\sqrt{n}+5)(2\sqrt{n^3}-2)}{\sqrt{n+5}} + \frac{5(\sqrt{n}+5)(\sqrt{n^3}+5)}{\sqrt[3]{n^2-4}} + \frac{2(\sqrt{n}+5)(n+5)}{\sqrt{\sqrt{n}+3}}$$

第1項：

$$\frac{O\left(n^{\frac{1}{2}}\right) \times O\left(n^{\frac{3}{2}}\right)}{O\left(n^{\frac{1}{2}}\right)} = O\left(n^{\frac{1}{2} + \frac{3}{2} - \frac{1}{2}}\right) = O\left(n^{\frac{3}{2}}\right)$$

第2項：

$$\frac{O\left(n^{\frac{1}{2}}\right) \times O\left(n^{\frac{3}{2}}\right)}{O\left(n^{\frac{2}{3}}\right)} = O\left(n^{\frac{1}{2} + \frac{3}{2} - \frac{2}{3}}\right) = O\left(n^{\frac{4}{3}}\right)$$

第3項：

$$\frac{O\left(n^{\frac{1}{2}}\right) \times O(n^1)}{O\left(n^{\frac{1}{2} \times \frac{1}{2}}\right)} = O\left(n^{\frac{1}{2} + 1 - \frac{1}{4}}\right) = O\left(n^{\frac{5}{4}}\right)$$

以上より、 $\frac{3}{2} > \frac{4}{3} > \frac{5}{4}$ なので、 $T_e(n) = O\left(n^{\frac{3}{2}}\right)$

(2) 次の C 言語風擬似コードの時間計算量を O 記法で示せ。

(15 点、各 5 点)

ただし、以下のコードでは引数 n が入力サイズとし、すべて $n = 2^k$ の形であるとする。

(d) 時間計算量 $T_d(n)$

アルゴリズム D:

```
void algoD(int n){
    for(i=0;i<n;i++){
        for(j=i;j<n;j++){
            (定数  $c_d$  時間の処理)
        }
    }
    return;
}
```

内側のループは、外側のループの実行回数に応じて、繰り返し回数が定まる。

よって、次のように計算できる。

$$\begin{aligned} T_d(n) &= \sum_{i=0}^{n-1} (n-i) \\ &= n + (n-1) + \cdots + 2 + 1 \\ &= \frac{n(n+1)}{2} \\ &= O(n^2) \end{aligned}$$

(e) 時間計算量 $T_e(n)$

アルゴリズム E:

```
void algoE(int n){
    for(i=1;i<n;i=2*i){
        for(j=0;j<n;j++){
            (定数  $c_e$  時間の処理)
        }
    }
    return;
}
```

外側のループカウンタ p は繰り返し回数 p に応じて、次のように増加する。

繰り返し回数 p	1	2	3	4	...	p
i の値	1	2	4	8	...	2^{p-1}

よって、外側のループは $O(\log n)$ 回繰り返される。

内側のループは外側のループ 1 回実行される都度、 n 回繰り返される。よって、全体では、 $n \times O(\log n) = O(n \log n)$ の時間計算量である。

(f) 漸近計算量 $T_f(n)$

アルゴリズム F :

```
void algoF(int n){
    if(n<=1){
        return;
    }else{
        (定数  $c_f$  時間の処理)
        algoF(n/2);
        return;
    }
}
```

次のような漸化式が成り立つ。

$$T_h(n) = \begin{cases} c_0 & n \leq 1 \\ T_h\left(\frac{n}{2}\right) + c_h & n > 1 \end{cases}$$

この漸化式を解く。

$T_h(n) = T_h(2^k) = T'_h(k)$ とおく。

$$T'_h(k) = \begin{cases} c_0 & k \leq 0 \\ 2T'_h(k-1) + c_h & k > 0 \end{cases}$$

$$\begin{aligned} T'_h(k) &= T'_h(k-1) + c_h \\ &= \{T'_h(k-2) + c_h\} + c_h = T'_h(k-2) + 2c_h \\ &= \{T'_h(k-3) + c_h\} + 2c_h = T'_h(k-3) + 3c_h \\ &\dots \\ &= T'_h(0) + kc_h \\ &= c_0 + kc_h \end{aligned}$$

よって、

$$T_h(n) = O(\log n)$$

2. べき乗のアルゴリズム (25 点)

次の C 言語の関数 `pow(int n, double x)` は、 $n = 2^k$ に対して、 $f(n, x) = x^n = \prod_{i=1}^n x$ を計算する。

ここで、 n は n に、 x は x に対応する。

```
double pow(int n, double x){
    double f; /*べき乗の関数値*/
    int i;
    f= 1.0; /*初期化*/
    for(    (ア)    ){
        f=f*x;
    }
    return f;
}
```

このとき、次の設問に答えよ。

(1) (5 点)

(ア) を適切に埋めよ。(5 点)

ループの繰り返しの度に x が 1 回乗じられるので、 n 回の繰り返しが必要である。

for 文の典型的な n 回繰り返しを記述する。

`(i=0; i<n; i++)`

(2) (5 点)

このプログラムの時間計算量 $T_{\text{slow}}(n)$ を求めよ。

For ループが n 回繰り返されるだけなので、

$T_{\text{slow}}(n) = O(n)$

である。

(3) (10 点)

$f(n, x) = x^n = \prod_{i=1}^n x$ を計算するより高速なアルゴリズム (関数)

`double fast_pow(int n, double x)` を示せ。(ここで、入力サイズは次数 n とする。)

次のように倍倍で計算すると良い。

```
double fast_pow(int n, double x){
    f=x; /*初期化*/
    for(i=0; i<k; i++){
        f=f*f;
    }
    return f;
}
```

(4) (5点)

(3)のアルゴリズムの時間計算量 $T_{fast}(n)$ を求めよ。

$k = \log n$ 回の繰り返しが行われる。

よって、

$$T_{fast}(n) = O(k) = O(\log n)$$

3. クイックソート

配列 A に次のようにデータが蓄えられているとする。

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
A	38	52	12	21	73	41	15	27

この配列をソートするために以下に示すようなクイックソートを用いる。

```
void qsort(int left, int right){
    int pos ; /*分割位置*/
    if(left>=right){
        return;
    }else{
        pos=partition(left,right);
        print_array(left,right);
        qsort(left,pos-1);
        qsort(pos+1,right);
    }
}
```

ここで、partition はピボット (基準値) の位置を求める関数で、部分配列 A[left]-A[right] をピボットより小さい要素、ピボット、ピボットより大きい要素の順に並べかえて、ピボット位置 (配列の添え字) を返す関数である。なお、ピボットは並び替え前における部分配列の右端 A[right] が選ばれるとする。また、print_array(left,right) は、部分配列 A[left]-A[right] を表示する関数である。

このとき、クイックソートにおける動作について答えよ。

(1)(10 点)

print_array によって表示される配列の内容をすべて示せ。

次のように階層的に表示される。

```
qsort(0,7):15,21,12;27;73,41,38,52
qsort(0,2):12;21,15   qsort(4,7):38,41;52;73
qsort(1,2):15;21   qsort(4,5):38;41   qsort(7,7):-
qsort(2,2):21       qsort(4,4):38
```

(2) (5 点)

$n = \text{right} - \text{left}$ とし、partition の時間計算量を $T_p(n)$ 、クイックソートの時間計算量を $T_q(n)$ とする。このとき $T_q(n)$ が成り立つべき漸化式を以下の形式で示せ。ただし、表示部分の

print_array は $T_q(n)$ の時間計算量に含めないようにせよ。

$$T_q(n) = \begin{cases} c(\text{定数}) & n \leq 0 \\ (\text{right, left, pos を用いた再帰式}) & n > 0 \end{cases}$$

部分配列の要素数を考慮すると次式が導出できる。

$$T_q(n) = \begin{cases} c(\text{定数}) & n \leq 0 \\ \underbrace{T_p(\text{right} - \text{left})}_{\text{partition 分}} + \underbrace{T_q(\text{pos} - \text{left})}_{\text{前半の再帰呼び出し}} + \underbrace{T_q(\text{right} - \text{pos})}_{\text{後半の再帰呼び出し}} & n > 0 \end{cases}$$

$$= \begin{cases} c(\text{定数}) & n \leq 0 \\ \underbrace{(\text{right} - \text{left} + 1)}_{\text{partition 分}} + \underbrace{T_q(\text{pos} - \text{left})}_{\text{前半の再帰呼び出し}} + \underbrace{T_q(\text{right} - \text{pos})}_{\text{後半の再帰呼び出し}} & n > 0 \end{cases}$$

(3) (5 点)

クイックソートの最悪時間計算量 $T_q^{\text{worst}}(n)$ を O 記法で示せ。

$n = \text{right} - \text{left}$ 個の配列の分割が、左の部分配列 $n-1$ 個、ピボット 1 個、右の部分配列 0 個のように分割されるのが再悪である。このとき、

$$T_q(n) = \begin{cases} c(\text{定数}) & n \leq 0 \\ n + T_q(n-1) & n > 0 \end{cases}$$

という漸化式が成り立つ。

この漸化式を解く。

$$\begin{aligned} T_q(n) &= n + T_q(n-1) \\ &= n + (n-1) + T_q(n-2) \\ &\dots \\ &= n + (n-1) + \dots + 1 + T_q(0) \\ &= \sum_{i=1}^n i + c \\ &= \frac{n(n+1)}{2} + c \\ &= O(n^2) \end{aligned}$$

(4) (5 点)

データ数 $n=8$ において時間計算量が最大となる入力例を一つ示せ。

$n = \text{right} - \text{left}$ 個の配列の分割が、左の部分配列 $n-1$ 個、ピボット 1 個、右の部分配列 0 個のように分割される入力例を与えればよい。

例えば、

A: 1, 2, 3, 4, 5, 6, 7, 8

のような入力例が考えられる。

4．線形探索

次のC言語の関数 `linear_search(double key,int n,double A[])` は、 n 個の要素の配列 A 中に key があるか探索を行う。配列中に key が無い場合には、 -1 を返し、配列中に key が存在する場合にはその添え字を返す。このとき、以下の設問に答えよ。

```
int linear_search(double key,int n,double A[]){
    int i=0;
    A[n]=( ア );    /* 番兵の設定      */
    while(key!=A[i]){
        i++;
    }
    if(i==n){
        return -1; /*key は配列 A に存在しない*/
    }else{
        return ( イ ); /*key は配列 A に存在する*/
    }
}
```

(1)(6点、各3点)

(ア)(イ)を適切に埋めよ。

番兵を用いた線形探索では、探索範囲の最後に求める値 (key) を置く。このことにより、探索範囲中に必ず key があることが保証される。比較回数の軽減と、バッファオーバーランの防止の効果がある。

ア： key

イ： i

(2)(4点)

このプログラムで最も比較回数が大きくなる場合を答えよ。

探索中に key 値が存在しない場合。この場合には探索範囲をすべて key 値と比較しながら走査する必要がある。

(3)(5点)

(2)の場合での比較回数 $H_{\text{worst}}(n)$ を求めよ。ただし、 $H_{\text{worst}}(n)$ は O 記法ではなくて、厳密な関数の形で答える事。

配列に key が存在しない場合、 n 個の配列要素比較後に最後の番兵と比較する。よって、
 $H_{\text{worst}}(n) = n + 1$

(4) (10 点)

配列 A 中に key が必ず含まれ、各 $A[i]$ に key が存在する確率 $P(i)$ が次式で表せるとする。

$$P(i) = \begin{cases} \frac{3}{2n} & 0 \leq i < \frac{n}{2} \\ \frac{1}{2n} & \frac{n}{2} \leq i < n \end{cases}$$

このときの平均比較回数 $H_{\text{ave}}(n)$ を求めよ。ただし、 $H_{\text{ave}}(n)$ は O 記法ではなくて、厳密な関数の形で答える事。

i 番目 key がある時の比較回数を $h(i)$ とする。また、

$$H_1 = \sum_{i=0}^{\frac{n}{2}-1} P(i)h(i)$$

$$H_2 = \sum_{i=\frac{n}{2}}^{n-1} P(i)h(i)$$

とする。

このとき平均比較回数 $H_{\text{ave}}(n)$ は次式を満たす。

$$H_{\text{ave}}(n) = \sum_{i=0}^{n-1} P(i)h(i) = H_1 + H_2$$

一方、線形探索のプログラムより、全ての i で

$$h(i) = i + 1$$

である。

よって、

$$\begin{aligned} H_1 &= \frac{3}{2n} \sum_{i=0}^{\frac{n}{2}-1} (i+1) \\ &= \frac{3}{2n} \cdot \left\{ 1 + 2 + \cdots + \frac{n}{2} \right\} \\ &= \frac{3}{2n} \cdot \left\{ \frac{\frac{n}{2} \left(\frac{n}{2} + 1 \right)}{2} \right\} \\ &= \frac{3}{16} \cdot (n+2) \end{aligned}$$

$$\begin{aligned}
H_2 &= \frac{1}{2n} \sum_{i=\frac{n}{2}}^{n-1} (i+1) \\
&= \frac{1}{2n} \bullet \left\{ \left(\frac{n}{2} + 1 \right) + \left(\frac{n}{2} + 2 \right) + \cdots + (n-1) + n \right\} \\
&= \frac{1}{2n} \bullet \left\{ \frac{\frac{n}{2} \left(\frac{3n}{2} + 1 \right)}{2} \right\} \\
&= \frac{1}{16} \bullet (3n+1)
\end{aligned}$$

$$\begin{aligned}
H_{ave}(n) &= H_1 + H_2 \\
&= \frac{3}{16} \bullet (n+2) + \frac{1}{16} \bullet (3n+1) \\
&= \frac{(3n+6) + (3n+1)}{16} \\
&= \frac{6n+7}{16}
\end{aligned}$$