2009 年度ソフトウェア工学試験解答例

1.漸近量評価（２５点）

（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）

$(a)\quad T_a(n)=\frac{2}{3}n^5-\frac{7}{2}n^3+\frac{1}{5}n$

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。

$T_a(n)=O(n^5)$

$(b)\quad T_b(n)=2(\sqrt[3]{n^2}+1)(\sqrt[3]{n^2}-7)+7(\sqrt{n}+5)(\sqrt{n}-3)$

項別に評価すると良い。

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

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

よって、$T_b(n)=O\left(n^{\frac{4}{3}}\right)$

$(c)\quad T_c(n)=(\log n)^5+\left(\sqrt{n}\right)^3$

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、

$T_c(n)=O\left(\left(\sqrt{n}\right)^3\right)=O\left(n^{\frac{3}{2}}\right)$

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

第１項：

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

第３項：

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

以上より、

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

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

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：

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

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。

(f) 時間計算量 $T_f(n)$

アルゴリズム F:

```
void algoF(int n){
    for(i=0;i<n;i++){
        for(j=0;j<i;j++){
            (定数 c_f 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。

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

(g) 時間計算量 $T_g(n)$

アルゴリズム G：

```
void algoG(int n){
    for(i=1;i<n;i=2*i){
            (定数 c_g 時間の処理)
    }
    return;
}
```

ループカウンタ i は繰り返し回数 j に応じて、次のように増加する。

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

$i \geq n$ のとき終了するので、繰り返し回数 j は次のように求められる。

$i \geq n$

$\therefore 2^{j-1} \geq 2^k$

$j \geq k+1 = \log n + 1$

以上より、$T_g(n) = O(\log n)$

(h) 漸近計算量$T_h(n)$

アルゴリズム H：

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

次のような漸化式が成り立つ。

$$T_h(n) = \begin{cases} c_0 & n \leq 1 \\ T_h\left(\frac{n}{2}\right) + T_h\left(\frac{n}{2}\right) + c_h & n > 0 \end{cases}$$

この漸化式を解く。

$T_h(n) = T_h(2^k) = T'_h(k)$ とおく。

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

$T'_h(k) = 2T'_h(k-1) + c_h$

$= 2\{2T'_h(k-2) + c_h\} + c_h = 2^2 T'_h(k-2) + \{2+1\}c_h$

$= 2^2\{2T'_h(k-3) + c_h\} + \{2+1\}c_h = 2^3 T'_h(k-3) + \{2^2 + 2 + 1\}c_h$

…

$= 2^k T'_h(0) + c_h \sum_{j=0}^{k-1} 2^j$

$= c_h(2^k - 1) + c_0 2^k$

よって、

$T_h(n) = c_h(n-1) + c_0 n$

$= (c_h + c_0)n - c_h$

$= O(n)$

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= ( ア ) /*初期化*/
       for(i=0;i<n;i++){
          f=f*x;
        }
       retrun f;
}
```

このとき、次の設問に答えよ。

（1）（5 点）

（ア）を適切に埋めよ。

掛け算の初期値として、 $f=x^0=1$ を設定する。

```
f= 1.0;
```

(2) （５点）

このプログラムの時間計算量 $T_{slow}(n)$ を求めよ。

For ループが n 回繰り返されるだけなので、

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

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）（10 点）

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;
        }
       retrun f;
}
```

(4)　(3)のアルゴリズムの時間計算量$T_{fast}(n)$を求めよ。（5 点）

$k=\log n$ 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（25 点）

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	12	4	5	11	8	2	15	7

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。（10 点）

次のように階層的に表示される。

$qsort(0,7):2,4,5;7;8,12,15,11$

$qsort(0,2):2,4;5 \quad qsort(4,7):8;11;12,15$

$qsort(0,1):2;4 \quad qsort(3,2):- \quad qsort(4,4):- \quad qsort(6,7):12;15$

$qsort(0,0):- \qquad qsort(6,6):-$

（２）（5 点）

$n = right - left$ 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。

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

部分配列の要素数を考慮すると次式が導出できる。

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

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

（３） (5 点)

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

$n=right-left$ 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、

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

という漸化式が成り立つ。

この漸化式を解く。

$$T_q(n)=n+T_q(n-1)$$
$$=n+(n-1)+T_q(n-2)$$
$$\cdots$$
$$=n+(n-1)+\cdots+1+T_q(0)$$
$$=\sum_{i=1}^{n}i+c$$
$$=\frac{n(n+1)}{2}+c$$
$$=O\left(n^2\right)$$

（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。

$n=right-left$ 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(25 点)

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{
      retrun  (  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、バッファー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 $H_{worst}(n)$ を求めよ。ただし、$H_{worst}(n)$ 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、

$H_{worst}(n) = n+1$

(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 $P(i) = \frac{1}{n}$ と表せるとする。このときの平均比較回数 $H_{ave}(n)$ を求めよ。ただし、$H_{ave}(n)$ 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。
このとき平均比較回数 $H_{ave}(n)$ は次式を満たす。

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

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。
以上より、

$$H_{ave}(n) = \frac{1}{n}\sum_{i=0}^{n-1}(i+1)$$
$$= \frac{1}{n}\bullet\left\{\frac{n(n+1)}{2}\right\}$$
$$= \frac{n+1}{2}$$