2007 年度ソフトウェア工学試験解答例

1.漸近量評価（２２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）

O 記法では、関数の増加量に気をつける。特に多項式においては、最高次数に注目して係数を省略した式になる。

$(a)\quad T_a(n)=\frac{1}{3}n^2-5n+8$

$T_a(n)=\frac{1}{3}n^2-5n+8=O(n^2)$

$(b)\quad T_b(n)=3(n+1)(n-2)(n-3)+7(n+5)(n-3)+8(n+5)$

展開するまでもない。第１項は三次式、第２項は２次式、第３項は１次式。

よって、

$T_b(n)=3(n+1)(n-2)(n-3)+7(n+5)(n-3)+8(n+5)=O(n^3)$

$(c)\quad T_c(n)=\sqrt{n}+\log n+\sin n$

増加量の変化に注意する。

$\sin n \le 1$ より、増加量は最小である。

$\sqrt{n}$ と $\log n$ では、$\log n$ の方が増加量が小さい。

例えば、$n=2^6=64$ とすると、

$\sqrt{2^6}=2^{\frac{6}{2}}=2^3=8$

$\log 2^6=6$

であるので、

$\log 2^6<\sqrt{2^6}$ である。

一般に、任意 $\varepsilon(0<\varepsilon<1)$ に対して、n^{ε} より $\log n$ の方が増加量が小さい。

以上より、

$T_c(n)=\sqrt{n}+\log n+\sin n$

$=O\left(n^{\frac{1}{2}}\right)$

$=O\left(\sqrt{n}\right)$

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

$8^{\log n}$

$= 2^{3\log n}$

$= \left(2^{\log n}\right)^3$

$= n^3$

よって、$8^{\log n}$ は n^5 より増加率が小さい。また、多項式は指数関数より増加率が小さい。

以上より、

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

$$= O\left(2^{\frac{n}{5}}\right)$$

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

$$(e)\quad T_e(n) = \frac{5(n+1)(n^2-2)}{\sqrt{n^2+5}} + \frac{2(n+5)(n-3)}{n-4} + \frac{7(n+5)(2n-1)}{\sqrt{n+3}}$$

これは、次数の関係だけに注目する。

第１項：分子３次式、分母１次式→２次式

第２項：分子２次式、分母１次式→１次式

第３項：分子２次式、分母$\frac{1}{2}$次式→$\frac{3}{2}$次式

よって、第１項が増加率最大であり、２次式である。

以上より、

$$T_e(n) = \frac{5(n+1)(n^2-2)}{\sqrt{n^2+5}} + \frac{2(n+5)(n-3)}{n-4} + \frac{7(n+5)(2n-1)}{\sqrt{n+3}} = O(n^2)$$

（２）次の C 言語風擬似コードの漸近計算量を O 記法で示せ。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。（１２点、各３点）
多重ループにおいては、内側のループと外側のループの繰り返し回数により考察する。
(a) 漸近計算量 $T_A(n)$
アルゴリズム A:

```
void algoA(int n){
    for(i=0;i<n;i++){
        for(j=0;j<10;j++){
            (定数 c_a 時間の処理)
        }
    }
    return;
}
```

内側のループは定数回（１０回）しか実行されないので、内側のループの実行時間は $O(1)$ 時間である。また、外側のループは n 回実行される。よって、全体では、$O(n)$ 時間である。
ループだから無条件に O 記法の次数が増加する訳ではない。
(b) 漸近計算量 $T_B(n)$
アルゴリズム B :

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

外側のループのループカウンタの増加に伴い、内側のループ回数が変化するアルゴリズム。
ループカウンタが１増加すると、繰り返し回数も１増加する。
これらの総和が時間計算量である。
よって、

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

(c) 漸近計算量$T_C(n)$

アルゴリズム C :

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

内側のループはいつもn回実行されるので、内側のループの時間計算量は$c_c n$ である。

外側のループの回数を見積もる。

i=2*i でループカウンタが更新されるので、

繰り返しに応じて、i の値は次の系列になる。

繰り返し回数	1	2	3	…	k	…
i	1	2	4	…	2^k	…

よって、外側の繰り返し回数は、$k-1 = \log n - 1$ である。

（倍々ゲームの原理である。繰り返し回数に応じて、カウンタ i が２のべき乗で増加する。）

以上より、全体の時間計算量は$T_C(n) = O(n \log n)$ である。

(d) 漸近計算量$T_D(n)$

アルゴリズム D：

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

アルゴリズムより次の漸化式が成り立つ。

$$T_D(n)=\begin{cases}1 & n=1\text{のとき}\\ T_D\left(\frac{1}{2}\right)+c_b & n>1\text{のとき}\end{cases}$$

このとき、$n=2^k$ より形式的に次数 k に注目して対応する漸化式をたてると良い。

$$T_D'(k)=\begin{cases}1 & k=0\text{のとき}\\ T_D'(k-1)+c_b & k>0\text{のとき}\end{cases}$$

この漸化式を解く。繰り返し漸化式を代入して計算する。

$$\begin{aligned}
&T_D'(k)=T_D'(k-1)+c_b\\
&=\{T_D'(k-2)+c_b\}+c_b=T_D'(k-2)+2c_b\\
&=\{T_D'(k-3)+c_b\}+2c_b=T_D'(k-3)+3c_b\\
&\vdots\\
&=T_D'(k-i)+ic_b\\
&\vdots\\
&=T_D'(0)+kc_b=kc_b+1
\end{aligned}$$

よって、$T_D'(k)=O(k)$ である。

$k=\log n$ より、

$T_D(n)=T_D'(k)=O(k)=O(\log n)$ である。

２．多項式の計算

２．多項式の計算（２５点）

下のアルゴリズムは多項式 $f(x)=a_0+a_1x^1+\cdots+a_nx^n$ の関数値を計算するホーナーの方法である。ホーナーの方法では、

$$f(x)=a_0+a_1x^1+\cdots+a_nx^n=\left(a_0+\left(\cdots\left(a_{n-1}+\left(a_n\right)x\right)\cdots\right)x\right)$$

の順序で関数値を計算する。各係数 a_i は（グローバル変数の）配列 $A[i]$ に入っているとし、最高次数は N とマクロ定義されているとする。この方法について以下の問いに答えよ。

ホーナーの方法:

```
/*多項式を計算するホーナーの方法（再帰的関数）
引数 i:内側から i 番目の括弧内の計算
引数 x:f(x)の x*/
1. double rec_horner(int i,int x){
2.   double fx;
3.   if(i==0){
4.     fx=A[N];
5.   }else{
6.     fx=A[N-i]+rec_horner(i-1,x)*x;
7.   }
8.   printf("%f¥n",fx);
9.   return fx;
10. }
```

(1) 関数 $f(x)=8-4x+3x^2-7x^3+5x^4-x^5$ の $x=2$ における関数値 $f(2)$ を上のアルゴリズムで求めるとき、８行目の printf 文で表示される系列を示せ。（５点）

再帰呼び出しにより、再帰の深さごとに printf 文が実行され、系列となって出力されることに注意する。

i	式	値
0:	(-1)	$=-1$
1:	$5+(-1)\times 2$	$=3$
2:	$-7+(3)\times 2$	$=-1$
3:	$3+(-1)\times 2$	$=1$
4:	$-4+(1)\times 2$	$=-2$
5:	$8+(-2)\times 2$	$=4$

以上より、$-1,3,-1,1,-2,4$ という系列が出力される。

なお、以下計算によっても関数値が正しいことが確認できる。

$$f(2)=8-4\cdot 2+3\cdot 2^2-7\cdot 2^3+5\cdot 2^4-2^5=8-8+12-56+80-32=4$$

(2) 上の疑似コードの時間計算量を$T_h(n)$とする。このとき、$T_h(n)$が満たすべき漸化式を示せ。ただし、入力サイズは最高次数のnとする。（５点）

アルゴリズムより、次の漸化式が成り立つ。

$$T_h(n)=\begin{cases}c_1 & n=0\\ T_h(n-1)+c_2 & n>0\end{cases}$$

ここで、c_1, c_2は定数である。

(3)(2)の漸化式を解き、最悪時間計算量$T_h(n)$をO記法で示せ。（５点）

漸化式を繰り返し適用し、代入して計算する。

$$T_h(n)=T_h(n-1)+c_2$$
$$=\{T_h(n-2)+c_2\}+c_2=T_h(n-2)+2c_2$$
$$\vdots$$
$$=T_h(n-i)+ic_2$$
$$\vdots$$
$$=T_h(0)+c_2n$$
$$=c_2n+c_1$$

以上より、

$T_h(n)=O(n)$時間である。

(4)ホーナーの方法は再帰ではなくて、繰り返しを用いても実現できる。ホーナーの方法を繰り返しで実現するアルゴリズム for_horner(int x)を C 言語風の疑似コードにより示せ。（１０点）

例えば、次のようになる。

```
/*多項式を計算するホーナーの方法(繰り返し版)
引数 x:f(x)の x*/
1. double for_horner(int x){
2. int i;/*ループカウンタ,内側から i 番目の括弧内を表す*/
3. double fx＝A[N]; /*関数値f(x)を表す*/
4.   for(i=1;i<N;i++){
5.     /*printf(“%f¥n”,fx);ここに printf 文を入れると上と同じ動作をする。*/
6.     fx=A[N-i]+fx*x;
7.   }
8.   return fx;
9. }
```

３．マージソート（２５点）

配列 A には n 個の要素 $a_0, a_1, \cdots, a_{n-1}$ が $A[0], A[1], \cdots, A[n-1]$ にそれぞれ蓄えられているとする。この配列 A をマージソートにより昇順にソートすることを考える。マージソートのアルゴリズムを下に擬似コードで示す。$l < m < r$ に対して、関数 merge は昇順の部分配列 $A[l] \cdots A[m]$ と昇順の部分配列 $A[m+1] \cdots A[r]$ から、部分配列 $A[l] \cdots A[r]$ を昇順にする。関数 `print_array(l,r,A)`は部分配列 $A[l] \cdots A[r]$ の内容を表示する。このとき、以下の問いに答えよ。

マージソート：

```
1. void m_sort(int l,int r,double A[N]){
2.   int m=(l+r)/2;/*中央の添え字*/
3.   if(l>=r){
4.     return;
5.   }else{
6.       /*print_array(l,r,A);*/
7.     m_sort( (ア) );/*前半のソート*/
8.     m_sort( (イ) );/*後半のソート*/
9.       /*print_array(l,r,A);*/
10.    merge(l,m,r,A);
11.      /*print_array(l,r,A);*/
12.    return;
13.  }
14.}
```

(1) マージソートが動作するように、(ア)、（イ）に入る適切な引数を示せ。（１０点、各５点）

(ア)(l,m,A),

(イ)(m+1,r,A)

引数の呼び出しの形と、重複がないように注意する。問題文の merge の説明にヒントが隠されている。

(2) $n = 8$ とし、配列 A に下のように値が蓄えられているとする。`m_sort(0,7,A)`を実行したときに 6,9,11 行目の各 `print_array(0,7,A)`により表示される配列の内容を示せ。（５点）

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]
A	18	46	7	83	21	75	39	55

６行目はソート前の系列、
9 行目は、前半と後半のソートが終了した系列、
11 行目は、マージによって全体がソートされた系列
になっていることに注意する。よって、次のような系列が出力される。

6行目: 18, 46, 7, 83, 21, 75, 39, 55
9行目: 7, 18, 46, 83, 21, 39, 55, 75
11行目: 7, 18, 21, 39, 46, 55, 75, 83

(3)マージソートの時間計算量を$T_m(n)$と表す。このとき、$T_m(n)$が満たすべき漸化式を示せ。ただし、関数 merge の時間計算量は$O(r-l)=O(n)$時間であるとし、表示用関数 `print_array` の計算量は含めないとする。（５点）
c_1, c_2を定数として、アルゴリズムより、次の漸化式が成り立つ。

$$T_m(n) \le \begin{cases} c_1 & n=1\text{のとき} \\ 2T_m\left(\frac{n}{2}\right)+c_2 n & n>1\text{のとき} \end{cases}$$

(4)(3)の漸化式を解き、マージソートの最悪時間計算量$T_m(n)$を O 記法で示せ。ただし、$n=2^k$であるとして良い。（５点）
$n=2^k$ より、 $k=\log n$ に関する漸化式をたてる。

$$T_m'(k) \le \begin{cases} c_1 & k=0\text{のとき} \\ 2T_m'(k-1)+c_2 2^k & k>0\text{のとき} \end{cases}$$

繰り返し代入することにより、漸化式を解く。

$$\begin{aligned}
&T_m'(k) \le 2T_m'(k-1)+c_2 2^k \\
&\le 2\left\{2T_m'(k-2)+c_2 2^{k-1}\right\}+c_2 2^k = 2^2 T_m'(k-2)+c_2\left(2\bullet 2^k\right) \\
&\le 2^2\left\{2T_m'(k-3)+c_2 2^{k-2}\right\}+c_2 2\bullet 2^k = 2^3 T_m'(k-3)+c_2\left(3\bullet 2^k\right) \\
&\vdots \\
&\le 2^i T_m'(k-i)+c_2\left(i\bullet 2^k\right) \\
&\vdots \\
&\le 2^k T_m'(0)+c_2\left(k\bullet 2^k\right) \\
&= c_1\left(2^k\right)+c_2\left(k\bullet 2^k\right)
\end{aligned}$$

よって、$T_m'(k)=O\left(k\bullet 2^k\right)$。 $n=2^k$ 、 $k=\log n$ であるので、

$$T_m(n) = T_m'(k) = O\left(k \cdot 2^k\right) = O(n \log n)$$

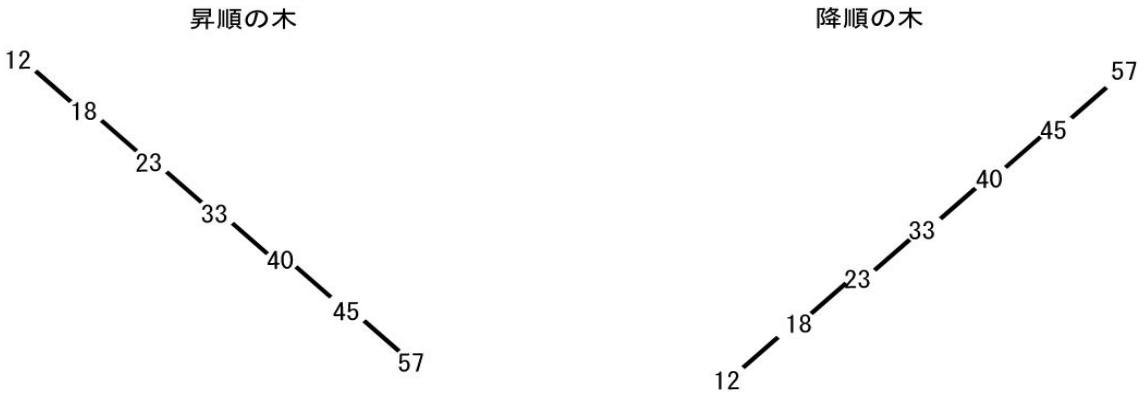
以上より、最悪時間計算量$T_m(n) = O(n \log n)$時間のアルゴリズムである。

４.２分探索木（２８点）

２分探索木Tに集合$S = \{12, 18, 23, 33, 40, 45, 57\}$を蓄えることを考える。２分探索木$T$にデータ x を挿入する操作を `insert(x)`で表わし、２分探索木Tからデータ x を削除する操作を `delete(x)`と表す。ただし、`delete(x)`において、削除される頂点の場所には、左の部分木中の最大の要素で置き換わる。（このTは単なる２分探索木で AVL 木ではない。）このとき、以下の問いに答えよ。

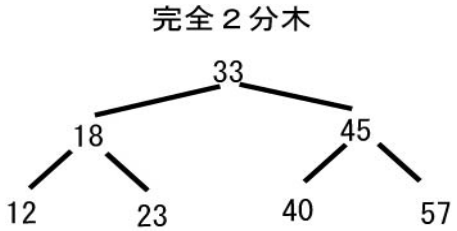
(1) 集合Sを蓄える２分探索木Tで高さが最大となるもの図示せよ。（１種類示せばよい。）（２点）

次のような２分探索木が最悪である。



(2) 集合Sを蓄える２分探索木Tで高さが最小となるもの図示せよ。（１種類示せばよい。）（２点）

完全２分木が高さ最小。これは１種類しか存在しない。次のようになる。

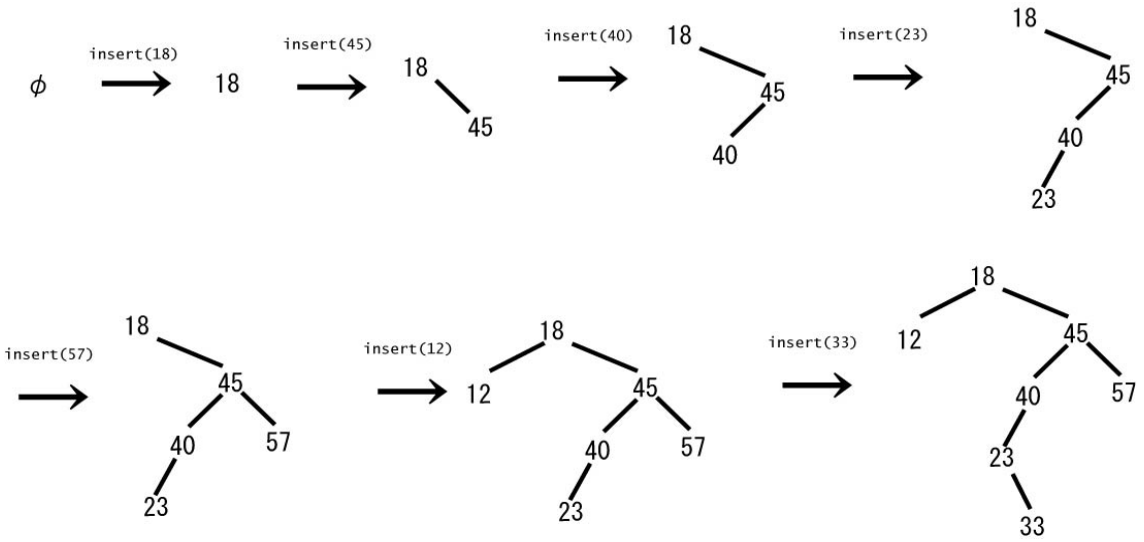


(3) 空の（要素数０の）２分探索木Tに次の順序で操作を行ったとき、２分探索木Tの状態の変化を図示せよ。（７点、各１点）

$\text{insert}(18) \to \text{insert}(45) \to \text{insert}(40) \to \text{insert}(23) \to \text{insert}(57) \to \text{insert}(12) \to \text{insert}(33)$

２分探索木への挿入では、既にある木に挿入要素が葉として“接ぎ木”されるだけである。次のように作成される。

挿入による２分探索木の作成

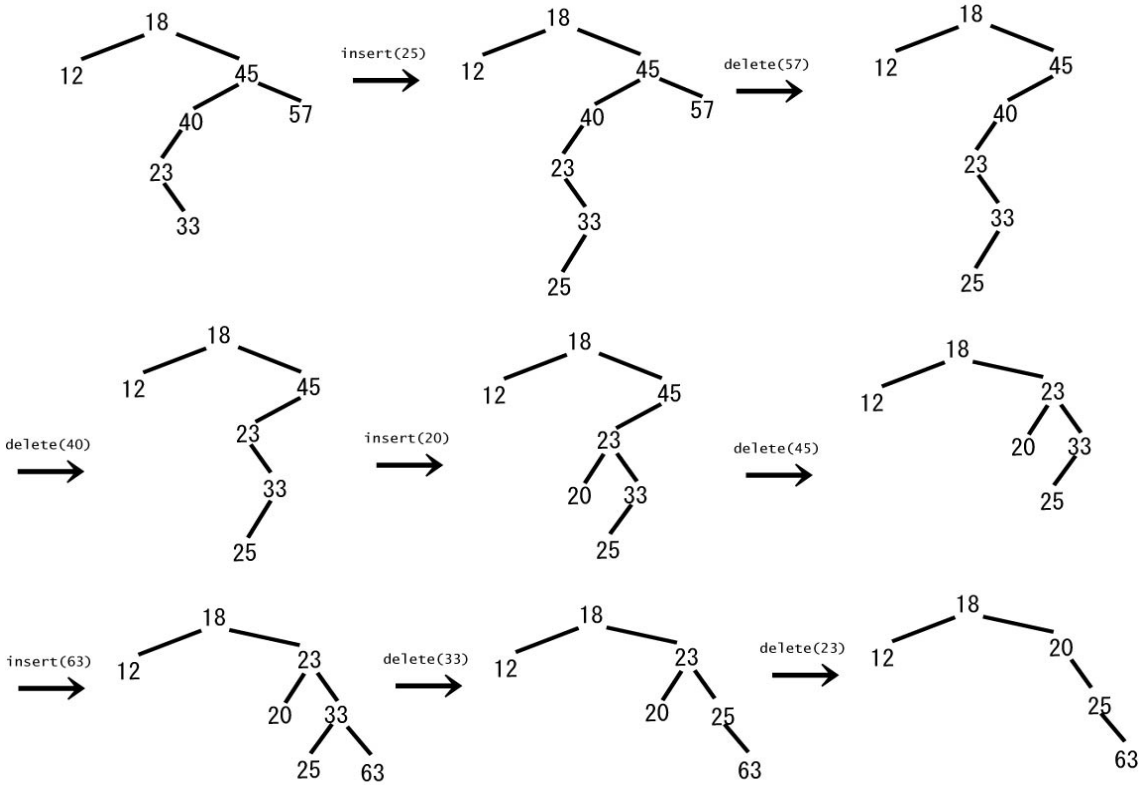


(4)(3)の木にさらに次の順序で操作を行ったとき、２分探索木Tの状態の変化を図示せよ。
(13 点、各 insert　1 点、各 delete２点)

insert(25) → delete(57) → delete(40) → insert(20)
→ delete(45) → insert(63) → delete(33) → delete(23)

delete の動作に注意を要する。次のように動作する。

２分探索木への操作



(5) 空の２分探索木Tにn回挿入を繰り返して２分探索木Tを作成するときの最悪時間計算量$T_{BT}(n)$を示せ。（４点）

Tに昇順にデータを挿入することを考える。

この場合$i,(1 \le i \le n)$番目のデータ挿入直前のTの高さは$i-1$であり、挿入直後のTの高さはiである。よって、i番目のデータの挿入に必要な時間計算量t_iは、定数cを用いて、$t_i = ci$と表せる。これらの総和が２分探索木作成に必要な時間計算量である。よって、

$$
\begin{aligned}
T_{BT}(n) &= \sum_{i=1}^{n} t_i \\
&= \sum_{i=1}^{n} ci \\
&= c\sum_{i=1}^{n} i \\
&= c(1+2+3+\cdots+n) \\
&= c\frac{n(n+1)}{2}
\end{aligned}
$$

と計算できる。以上より、２分探索木作成にひつような最悪時間計算量は、$T_{BT}(n) = O(n^2)$である。

なお、最悪時間計算量を実現する挿入例は次の図のようになる。

最悪の２分探索木構築