# 2008年度ソフトウェア工学㊞試験問題解答例

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）

$(a)\quad T_a(n)=2n^5-9n^3-7n$

単純な多項式の漸近計算量は、最高次で定まる。（多項式の漸近的な振る舞いは、最高次数の項だけで定まる）

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

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

指数法則に注意する。

第１項は、$\frac{1}{3}+\frac{1}{3}=\frac{2}{3}$次の式。

第２項は、$\frac{1}{2}+\frac{1}{2}=\frac{2}{2}=1$次の式。

第３項は、$1+1=2$次の式。

以上より、

$T_b(n)=O(n^2)$

$(c)\quad T_c(n)=\left(\log n+\sqrt{n}\right)^5+\left(2^{\log n}+\sqrt{n}\right)^3$

$O(\log n)\subset O\left(\sqrt{n}\right)\subset O\left(n\right)$と、$2^{\log n}=n$に注意する。

第１項、$\frac{1}{2}\times 5=\frac{5}{2}$次の式。

第２項、$1\times 3=3$次の式。

よって、$\frac{5}{2}\le 3$より、$T_c(n)=O(n^3)$

$(d)\quad T_d(n)=4^{\log n}+\sqrt{n^3}+\left(\log n\right)^5$

どんな大きな$k$とどんな小さな$\varepsilon$に対しても、$O\left((\log n)^k\right)\subset O\left(n^{\varepsilon}\right)$。

また、$4^{\log n}=\left(2^2\right)^{\log n}=2^{2\times\log n}=\left(2^{\log n}\right)^2=n^2$に注意する。

第１項、$O(n^2)$。第２項、$O(n^{\frac{3}{2}})$。第３項$O\left((\log n)^5\right)$。

よって、$\frac{3}{2} \le 2$ より、$T_d(n) = O(n^2)$。

$(e)$ $T_e(n) = \frac{3(\sqrt{n}-8)(\sqrt[3]{n^2}-2)}{\sqrt{n^3+5}} + \frac{2(\sqrt[5]{n^3}-4)(\sqrt{n^2}-7)}{\sqrt[3]{n^5+3}}$

第１項は、$\frac{1}{2}+\frac{2}{3}-\frac{3}{2} = \frac{3+4-9}{6} = -\frac{2}{6} = -\frac{1}{3}$ より、$O\left(n^{-\frac{1}{3}}\right)$ の式。

第２項は、$\frac{3}{5}+\frac{2}{2}-\frac{5}{3} = \frac{9+15-25}{15} = -\frac{1}{15}$ より、$O\left(n^{-\frac{1}{15}}\right)$ の式。

よって、$-\frac{1}{3} < -\frac{1}{15}$ より、$T_e(n) = O\left(n^{-\frac{1}{15}}\right) = O\left(\frac{1}{\sqrt[15]{n}}\right)$。

（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。（１２点、各３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。

$(a)$ 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<i;j++){
            (定数 c_a 時間の処理)
        }
    }
    return;
}
```

（粗い解析）

外側のループは、$n$ 回繰り返される。

外側のループが１回実行されるとき、内側のループは $i$ 回繰り返される。ここで、$i < n$ という関係を用いると、外側のループが１回実行されるとき、内側のループは“高々” $n$ 回実行される。よって、内側のループは“高々” $n^2$ 回実行される。よって、$T_A(n) = O(n^2)$

（細かい解析）

外側のループカウンタの増加に伴い、内側の繰り返し回数が増加することに注意する。

最も内側の部分の実行回数は次式で与えられる。

$$T_A(n) = \sum_{i=1}^{n} i \times c_a$$
$$= c_a \left(1 + 2 + \cdots + n\right)$$
$$= c_a \bullet \frac{n(n+1)}{2}$$
$$= O(n^2)$$

なお、この場合粗い解析と細かい解析では、係数部分に$\frac{1}{2}$の差異があるが、O記法では同一となる。

(*b*) 漸近計算量$T_B(n)$

アルゴリズム B :

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

（粗い解析）

繰り返し回数が 1 回増加するごとにカウンタが 2 倍になることに注意して、外側のループの繰り返し回数$l$とカウンタ$i$の関係を調べる。

| $l$ | 0 | 1 | 2 | $\cdots$ | $k = \log n$ |
|---|---|---|---|---|---|
| $i$ | 1 | 2 | 4 | $\cdots$ | $2^k = n$ |

よって、外側のループは$O(\log n)$回実行される。ここで、$i < n$という関係を用いると、外側のループが 1 回実行されるとき、内側のループは“高々” $n$回実行される。よって、内側のループは“高々” $n \log n$回実行される。よって、$T_B(n) = O(n \log n)$。

（細かい解析）

外側のループカウンタの増加に伴い、内側の繰り返し回数が変化することに注意する。

最も内側の部分の実行回数は次式で与えられる。

$$
\begin{aligned}
T_B(n) &= c_b \sum_{l=1}^{k} 2^l \\
&= c_b \left(1 + 2 + 4 + \cdots + 2^k\right) \\
&= c_b \bullet \left(2^{k+1} - 1\right) \\
&= c_b \bullet \left(2n - 1\right) \\
&= O(n)
\end{aligned}
$$

この場合、粗い解析と細かい解析では、求められる漸近計算量が異なる。よって、粗い解析の場合は減点とする。

(*c*) 漸近計算量$T_C(n)$

アルゴリズム C :

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

アルゴリズムより、 $n$ に関して次の漸化式が成り立つ。

$$T_C(n)=\begin{cases} c & n=1 \\ T_C\left(\frac{n}{2}\right)+c_c & n\geq 2 \end{cases}$$

$n=2^k$ より、 $k$ に関して次ぎの漸化式が成り立つ。

$$T'_C(k)=\begin{cases} c & k=0 \\ T'_C(k-1)+c_c & k>0 \end{cases}$$

この漸化式を解く。

$$\begin{aligned} &T'_C(k)=T'_C(k-1)+c_c \\ &=\left(T'_C(k-2)+c_c\right)+c_c=T'_C(k-2)+2c_c \\ &=\left(T'_C(k-3)+c_c\right)+2c_c=T'_C(k-3)+3c_c \\ &\vdots \\ &=T'_C(0)+kc_c \\ &=c+kc_c \\ &=O(k) \end{aligned}$$

よって、 $T_C(n)=O(\log n)$ 。

(*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{
        algoD(n-1);
        (定数c_d 時間の処理)
        algoD(n-1);
      return;
    }
}
```

アルゴリズムより、$n$ に関して次の漸化式が成り立つ。

$$T_D(n)=\begin{cases}c & n=1\\ T_D(n-1)+c_d+T_D(n-1) & n\geq 2\end{cases}$$

$$=\begin{cases}c & n=1\\ 2T_D(n-1)+c_d & n\geq 2\end{cases}$$

この漸化式を解く。

$$T_D(n)=2T_D(n-1)+c_d$$

$$=2\left(2T_D(n-2)+c_d\right)+c_d=2^2T_D(n-2)+2c_d+c_d$$

$$=2^2\left(2T_D(n-3)+c_d\right)+2c_d+c_d=2^3T_D(n-3)+c_d\left(2^2+2^1+2^0\right)$$

$$\vdots$$

$$=2^iT_D(n-i)+c_d\sum_{l=0}^{i}2^l$$

$$\vdots$$

$$=2^{n-1}T_D(1)+c_d\sum_{l=0}^{n-1}2^l$$

$$=2^{n-1}c+c_d\left(2^n-1\right)$$

$$=\left(\frac{c}{2}+c_d\right)2^n-c_d$$

$$=O(2^n)$$

よって、$T_D(n)=O(2^n)$

２．データ構造（ヒープ）（２８点）

配列 $A$ を利用してデータ構造のヒープを作成する。ヒープは次の条件を満たすように構築されているとする。

- $A[0]$ を根 $r$ とし、根 $A[0]$ には最小値が蓄えられる。
- 根以外の各頂点 $v$ に対して、親の方が小さい要素を蓄えているとする。（すなわち、頂点 $v$ を根とする部分木中の最小の値が頂点 $v$ に蓄えられる。）
- 頂点 $v$ が $A[i]$ に蓄えられているとき、頂点 $v$ の左の子は $A[2i+1]$ に、右の子は $A[2i+2]$ に蓄えられるとする。

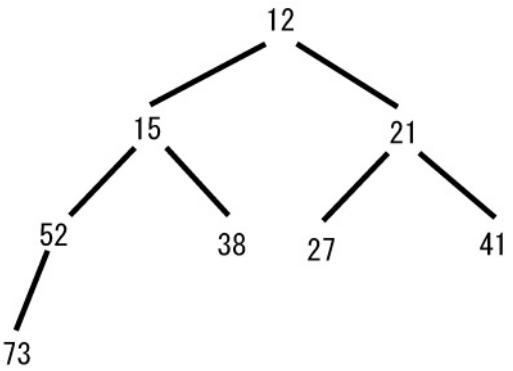
このとき、次の設問に答えよ。

（１）配列が次の状態であるとき、ヒープの木の形状を図示せよ。

| | A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] | A[9] | A[10] | A[11] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 12 | 15 | 21 | 52 | 38 | 27 | 41 | 73 | | | | |

次のような形状となる。

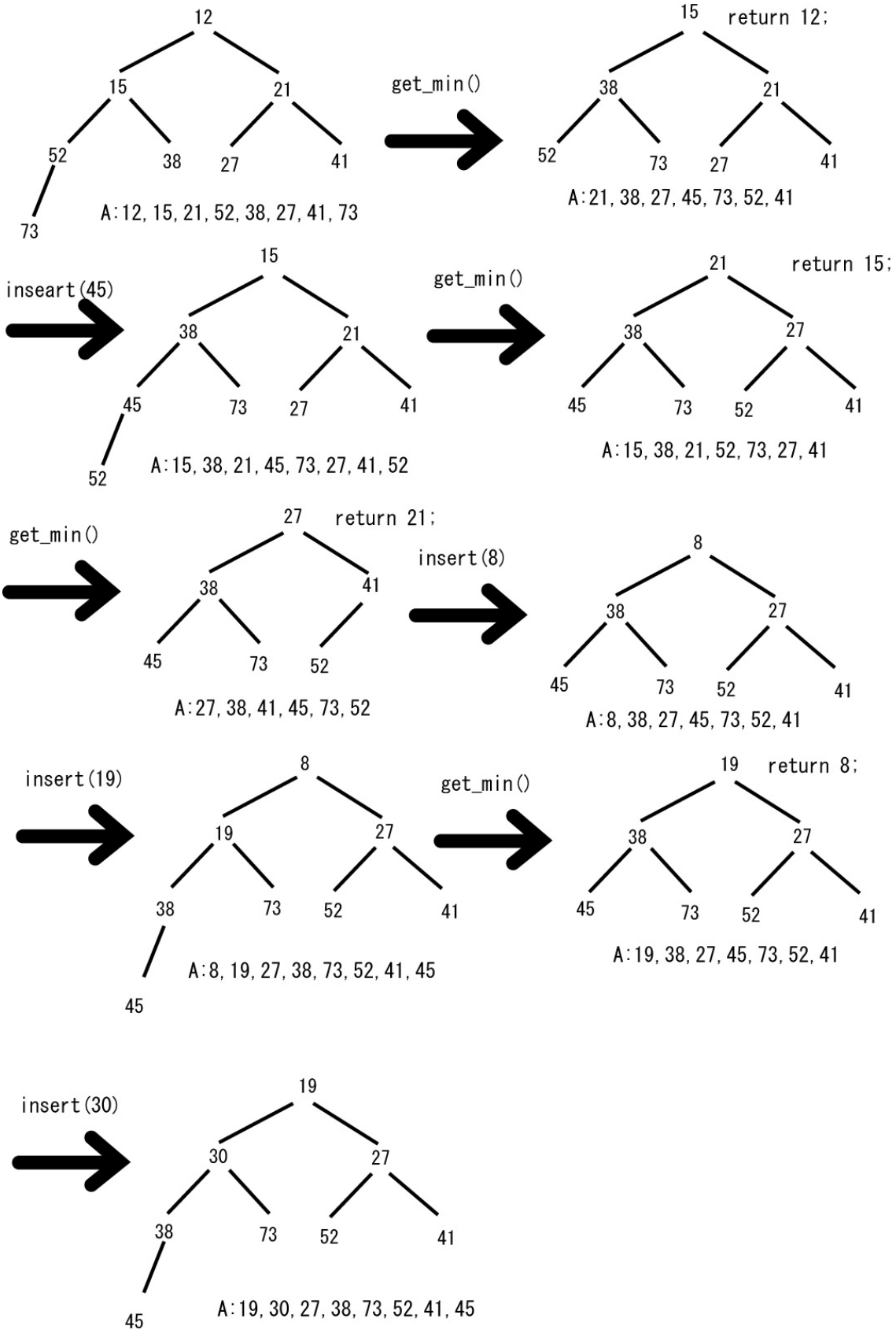
A:12, 15, 21, 52, 38, 27, 41, 73




（２）（１）の状態から次のように操作を行うとする。このとき、配列の状態とヒープの木をそれぞれ示せ。

$$get\_\min() \rightarrow insert(45) \rightarrow get\_\min() \rightarrow get\_\min()$$
$$\rightarrow insert(8) \rightarrow insert(19) \rightarrow get\_\min() \rightarrow insert(30)$$

次のように変更される。


12
15
21
52
38
27
41
73
A:12, 15, 21, 52, 38, 27, 41, 73
get_min()
15
return 12;
38
21
52
73
27
41
A:21, 38, 27, 45, 73, 52, 41
inseart(45)
15
38
21
45
73
27
41
52
A:15, 38, 21, 45, 73, 27, 41, 52
get_min()
21
return 15;
38
27
45
73
52
41
A:15, 38, 21, 52, 73, 27, 41
get_min()
27
return 21;
38
41
45
73
52
A:27, 38, 41, 45, 73, 52
insert(8)
8
38
27
45
73
52
41
A:8, 38, 27, 45, 73, 52, 41
insert(19)
8
19
27
38
73
52
41
45
A:8, 19, 27, 38, 73, 52, 41, 45
get_min()
19
return 8;
38
27
45
73
52
41
A:19, 38, 27, 45, 73, 52, 41
insert(30)
19
30
27
38
73
52
41
45
A:19, 30, 27, 38, 73, 52, 41, 45

（３）一般に$n$個のデータが蓄えられているときに、操作１回に必要な最悪時間計算量を求O 記法で示せ。ただし、結果だけでなく、理由も記述すること。
最悪の場合、insert(x)も get_min()もヒープの高さ$h$に比例する時間計算量が必要になる。高さ$h$のヒープでの最大の要素数$n_{\max}$は、完全２分木状になっているときであり、次式が成り立つ。

$$n_{\max} = 2^h - 1$$

一方、高さ$h$のヒープでの最小の要素数$n_{\min}$は、高さ$h-1$の完全２分木に１要素が加わったときであり、次式が成り立つ。

$$n_{\min} = \left(2^{h-1} - 1\right) + 1 = 2^{h-1}$$

よって、要素数$n$に関して、次式が成り立つ。

$$n_{\min} \le n \le n_{\max}$$
$$\therefore 2^{h-1} \le n \le 2^h - 1 < 2^h$$
$$\therefore h - 1 \le \log n < h$$
$$\therefore \log n < h \le \log n + 1$$

よって、各操作の最悪時間計算量は、$O(h) = O(\log n)$である。

３．選択ソート

配列$B$に$n$個のデータが蓄えられている。このとき、最大値を求めるアルゴリズムを利用した選択ソートにより、配列$B$を昇順にソートしよう。このようなアルゴリズムを下に示す。

```
int find_max(int left,int right){
    int j,max=left;
    for(j=left+1;j<=right;j++){
           if(B[max]<B[j]){
                  max=j;
           }
    }
    return max;
}
void select_sort(){
    int i,max;
    for(i=n-1;i>0;i--){
           max=find_max( （ア） );
           swap( （イ） );
    }
    return;
}
```

ここで、find_max は配列 $B$ の $B[left]-B[right]$ 中の最大値を保持している配列要素 $B[j]$ の添え字 $j$ を返す。

(1)　選択ソートが適切に動作するように、(ア)、(イ) を適切に埋めよ。

ア：find_max(0,i);

イ：swap(&B[i],&B[min])

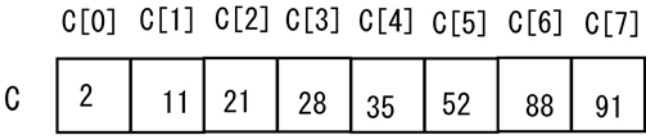
(2) 1 回の find_max 呼び出しに必要な最悪時間計算量 $T_{\max}(n)$ を示せ。

(3)アルゴリズム全体における交換回数 $N_{swap}(n)$ を示せ。なお、$N_{swap}(n)$ は O 記法ではなく、厳密な式で答えること。

(4)アルゴリズム全体の最悪時間計算量 $T_{sort}(n)$ を O 記法で示せ。

４．２分探索

配列$C$に次のようなデータが蓄えられているとする。このとき、下に示すプログラムで２分探索を行おうとしている。




```
int binary_search(int key,int left,int right){
  int mid;/*中央の添え字*/
  if(left>right){return -1;/*存在しない*/}
  else{
    mid=(left+right)/2;
    if(C[mid]==key){
       return mid;
    }else if(key < C[mid]){
         return binary_search( （ア）  );/*小さい方*/
    }else{
         return binary_search( （イ）  );/*大きい方*/
  }
}
```

⑴２分探索が適切に動作するよういに、(ア)、（イ）に入る適切な引数を示せ。

⑵次の引数でこの関数を呼び出したとき、調べられる配列の要素を順に示せ。

⒜`binary_search(21,0,7);`

⒝`binary_search(65,0,7);`

⑶データ数が一般の$n$であるとき、上のプログラムの時間計算量$T(n)$が満たすべき漸化式を示し、最悪時間計算量$T(n)$を$O$記法で答えよ。