問題用紙は 6 ページある。

問題 4 は 4 ページから始まるので注意。

問題 1.

$$\begin{cases} a_0 = 0, \\ a_1 = 1, \\ a_n = a_{n-2} + a_{n-1} \qquad n \geq 2 \text{ のとき} \end{cases}$$

で定義される数列を**フィボナッチ数列**と呼ぶ。

フィボナッチ数列の第 n 項を計算する C の関数を書け。関数は計算のみを行い、入出力などそれ以外の動作はいっさい行ってはならない。関数のインターフェースは、以下の通りとせよ。

```
/* 返値はフィボナッチ数列の第 n 項である。 */
unsigned long long      fibonacci(unsigned n);
```

問題 2. 開番地法のハッシュ表で、表の大きさを N、衝突した際の増分値を P とおくと、N と P は互いに素でなくてはならない。それをふまえて、以下の問いに答えよ。

(1) ある正整数 n について $N = 2^n$ が成り立てば、$0 < P < N$ をみたす任意の奇数 P は N と互いに素である。そこで、正整数 M を引数としてとり、$N \geq M$ かつある正整数 n について $N = 2^n$ が成り立つ最小の整数 N を返す C の関数を書け。

(2) N が素数ならば、$0 < P < N$ をみたす任意の整数 P は N と互いに素である。そこで、正整数 M を引数としてとり、$N \geq M$ である最小の素数 N を返す C の関数を書け。

関数は計算のみを行い、入出力などそれ以外の動作はいっさい行ってはならない。引数も返値も `size_t` 型とせよ。いずれの関数も、引数の値の二倍が `size_t` の上限値を超えることはないものと仮定してよい。

問題 3. コード 1〜5 は、`int` 型の値を格納するさまざまなデータ構造について、最大値を持つ要素を探索するアルゴリズムを、C の関数として実装したものである。

コード 1 配列。データは小さい値から大きい値に順序づけられていると仮定する。
コード 2 配列。データの並びに制限はない。
コード 3 単方向線形リスト。データは小さい値から大きい値に順序づけられていると仮定する。
コード 4 単方向線形リスト。データの並びに制限はない。
コード 5 二分木。データは左が小さく右が大きな順で並んでいると仮定する。

(1) 空欄を埋めよ。

(2) それぞれのコードが実装するアルゴリズムについて、要素の個数を n としたときの時間計算量のオーダーをのべよ。

コード 1 最大要素へのポインタ（順序つき配列）

```
#include <stdlib.h>

int *
maxelem_ordered_array(int array[], size_t size)
{
        if (size == 0)
                return NULL;
        return [   (あ)   ];
}
```

コード 2 最大要素へのポインタ（順序なし配列）

```
#include <stdlib.h>

int *
maxelem_unordered_array(int array[], size_t size)
{
        int     *p, *p_max;
        if (size == 0)
                return NULL;
        p_max = [   (い)   ];
        for (p = [   (う)   ]; p < [   (え)   ]; p = [   (お)   ]) {
                if (*p_max < *p) {
                        p_max = p;
                }
        }
        return p_max;
}
```

コード 3 最大要素へのポインタ（順序つき単方向線形リスト）

```
#include <stdlib.h>

struct cell {
        struct cell     *next;  /* 次のセル */
        int value;
};

struct cell *
maxelem_ordered_slist(struct cell *first)
{
        struct cell     *p;
        if (first == NULL)
                return NULL;
        p = [   (か)   ];
        while ([   (き)   ] != NULL) {
                p = [   (く)   ];
        }
        return p;
}
```

コード 4 最大要素へのポインタ（順序なし単方向線形リスト）

```
#include <stdlib.h>

struct cell {
        struct cell     *next;  /* 次のセル */
        int value;
};

struct cell *
maxelem_unordered_slist(struct cell *first)
{
        struct cell     *p, *p_max;
        if (first == NULL)
                return NULL;
        p_max = (け);
        for (p = (こ); p != NULL; p = (さ)) {
                if (p_max->value < p->value) {
                        p_max = p;
                }
        }
        return p_max;
}
```

コード 5 最大要素へのポインタ（順序つき二分木）

```
#include <stdlib.h>

struct node {
        struct node     *left;  /* 左部分木 */
        struct node     *right; /* 右部分木 */
        int     label;
};

struct node *
maxelem_ordered_bintree(struct node *root)
{
        struct node     *p;
        if (root == NULL)
                return NULL;
        p = (し);
        while ((す) != NULL) {
                p = (せ);
        }
        return p;
}
```

問題 4. コード 6 とコード 7 とコード 8 は、配列のソートを C の関数 `pine_sort()` として実装したものである。いずれも、第一引数は構造体 `struct pine` の配列の先頭を指すポインタ、第二引数はその配列の大きさである。二つの引数で定められた配列を昇順にソートする。

それぞれの採用するアルゴリズムは以下の通りである。コード 6 とコード 7 は選択ソートを実装したものである。コード 6 では値の小さいものから確定していき、コード 7 では値の大きいものから確定していく。コード 8 はヒープソートを実装したものである。

以下を前提とする。

- 構造体 `struct pine` はファイル `pine.h` 内で定義されている。
- 関数 `pine_swap()` と `pine_less()` は、形式が

  ```
  void    pine_swap(struct pine *, struct pine *);
  int     pine_less(const struct pine *, const struct pine *);
  ```

  であり、外部で定義され、そのプロトタイプ宣言がファイル `pine.h` 内でなされている。
  - 関数 `pine_swap()` は、第一引数の指す先と第二引数の指す先の値を交換する。
  - 関数 `pine_less()` は、第一引数の指す先と第二引数の指す先で値を比較し、前者が小さければ 0 でない整数を返し、さもなくば 0 を返す。

(1) 空欄を埋めよ。

(2) 実用的な三つのソートアルゴリズムであるクイックソート・ヒープソート・マージソートをどう使い分ければ良いか、時間計算量と空間計算量（計算時間とメモリ消費量）の観点から解説せよ。

コード 6 選択ソートの実装（小さいほうから確定）

```
#include <stdlib.h>
#include "pine.h"

void
pine_sort(struct pine *a, size_t n)
{
        size_t  i, j, m;
        for (i = 0; i < n; i ++) {
                m = i;
                for (j = i + 1; j < n; j ++) {
                        if (pine_less(&a[  (そ)  ], &a[  (た)  ])) {
                                m = j;
                        }
                }
                pine_swap(&a[  (ち)  ], &a[  (つ)  ]);
        }
}
```

コード 7 選択ソートの実装（大きいほうから確定）

```
#include <stdlib.h>
#include "pine.h"

void
pine_sort(struct pine *a, size_t n)
{
        size_t  i, j, m;
        for (i = n; i > 0; i --) {
                m = 0;
                for (j = 1; j < i; j ++) {
                        if (!pine_less(&a[   (て)   ], &a[   (と)   ])) {
                                m = j;
                        }
                }
                pine_swap(&a[   (な)   ], &a[   (に)   ]);
        }
}
```

コード 8 ヒープソートの実装

```
#include <stdlib.h>
#include "pine.h"

static inline void
sift(struct pine *a, size_t m, size_t n)
{
        size_t  i, j;
        for (i = m; (j = i * 2 + 1) < (ぬ); i = j) {
                if (j + 1 < (ね) && pine_less(&a[j], &a[j + 1])) {
                        j ++;
                }
                if (pine_less(&a[i], &a[j])) {
                        pine_swap(&a[(の)], &a[(は)]);
                } else
                        break;
        }
}

void
pine_sort(struct pine *a, size_t n)
{
        size_t  i;
        if (n <= 1)
                return;
        for (i = n / 2; i > 0;) {
                i --;
                sift(a, (ひ), (ふ));
        }
        for (i = n - 1; i > 0; i --) {
                pine_swap(&a[(へ)], &a[(ほ)]);
                sift(a, (ま), (み));
        }
}
```