配点：(1) 15 点，(2) 24 点，(3) 24 点，(4-1) 13 点，(4-2) 12 点，(4-3) 12 点

図 1 の ANSI-C 準拠である C 言語のプログラムは，53 行目の整数型（int 型）配列（array）dat で与えられる入力に対して，昇順（ascending order）に整列（sort）するプログラムである．以下の各問に答えよ．

(1) このプログラムで実現されている整列方式（sorting algorithm）の名称を答えよ．

(2) このプログラムの 4 行目の display 関数を，1 回目に呼び出すときと，3 回目に呼び出すときの，配列 dat の内容を記述せよ．

(3) このプログラムは，入力を指定する 53 行目の配列 dat によって動作が変化する．49 行目の sort 関数の呼び出しにおいて，常に k と b が等しくなり，かつ 25 行目の処理が行われないように，配列 dat を並べ替えよ．

(4) 49 行目の sort 関数の呼び出しにおいて，常に k と b が等しくなる配列 dat が与えられたときについて，以下の各小問に答えよ．

(4-1) 配列 dat の要素数を n としたとき，プログラム実行の全体における sort 関数が呼び出される回数を求めよ．

(4-2) このような状況における，このプログラムが示す整列方式の時間計算量（time complexity）を，整列要素数 n を用いてオーダー表記（order notation）で答えよ．

(4-3) これは，このプログラムが示す整列方式にとってどのような状況であるか，時間計算量の観点から簡潔に述べよ．

```
1   #include <stdio.h>
2   #define MAX 8
3
4   void display(int *arr) {
5       int i;
6
7       for(i=0; i<MAX; i++) printf("%d ", arr[i]);
8       printf("¥n");
9   }
10
11  int arrange(int a, int b, int *arr) {
12      int left, right, bd, temp;
13
14      left=a; right=b; bd=arr[(a+b)/2];
15
16      while(1) {
17          while(arr[left]<bd) {
18              left++; if(left>b) break;
19          }
20          while(arr[right]>=bd) {
21              right--; if(right<a) break;
22          }
23
24          if(right<a) {
25              arr[(a+b)/2]=arr[a]; arr[a]=bd;
26              return a+1;
27          } else if(left<=right) {
28              temp=arr[left];
29              arr[left]=arr[right]; arr[right]=temp;
30
31              left++; right--;
32          } else {
33              break;
34          }
35      }
36
37      return(left);
38  }
39
40  void sort(int a, int b, int *arr) {
41      int k;
42
43      if(b<=a) return;
44
45      k=arrange(a, b, arr);
46      display(arr);
47
48      sort(a, k-1, arr);
49      sort(k, b, arr);
50  }
51
52  int main(void) {
53      int dat[]={30, 50, 70, 40, 20, 80, 60, 10};
54
55      sort(0, MAX-1, dat);
56      display(dat);
57      return 0;
58  }
```

図1　プログラム

(1) クイックソート (Quick Sort)

(2) 1回目: {30, 10, 20, 40, 70, 80, 60, 50}

3回目: {10, 20, 30, 40, 70, 80, 60, 50}

(3) {20, 40, 60, 80, 10, 30, 50, 70}

(4) (4-1) このとき、sort関数が呼び出される回数は $n+(n-1)=2n-1$ 回.

(4-2) このような状況における、時間計算量は $O(n^2)$

(4-3) これは、クイックソート (Quick Sort) にとって最悪の状況である.

ピボット (pivot) は常に区間の最大値となる、分割は極端に不均衡である.