門外不出

かねじゅん模試

アルゴリズムとプログラミング

作成者はこの模試に関する一切の責任を負わない

次の、下に示すプログラムの説明文を読み、それに関する以下の問いに答えよ。

このプログラムは、配列 data 内の数値を(A1)に出力するプログラムである。関数(B1)の整列アルゴリズムは、かねじゅんによる(C1)の改良版である。以下でアルゴリズムの動作を具体的に説明する。

データを整列するにあたり、(ア)配列の先頭の隣のデータから順に検査をしてゆく。まず、(イ)検査対象となったデータを、配列の先頭から対象データまでの間のどこに挿入するべきか調べる。これを行うのが関数(B2)であり、この関数の探索アルゴリズムは一般に(C2)と呼ばれている。その後、その位置にデータを挿入するために、データをシフトする作業が必要となる。このとき配列の先頭と最後尾を接続したリングバッファで考え、データの今ある位置と、(ウ)挿入先までの(i)データが少ない方向を判断し、その方向にデータをシフトする。通常の(C1)と同じ、先頭から最後尾方向である(A2)側にシフトする処理を実現しているのが関数(B3)であり、逆側にシフトする処理をしているのが関数(B4)である。初めて関数(B4)を呼び出した後は、配列の先頭は必ずしも配列の添字が(D1)のデータではなくなる。データをシフトし終わったあと、(ii)適切な位置にデータを挿入し、検査対象を隣のデータに移す。

このアルゴリズムが元のアルゴリズムと比べて優れている点は 2 つある。1 つ目は、データの挿入先を(C2)で探索している点である。(C3)で探索する元のアルゴリズムに比べデータ数が膨大なときに有効なよく知られた高速化手法である。また、元のアルゴリズムでは、最悪の場合、最後尾のデータを先頭に持ってくるために、配列の要素数より(D2)少ない個数のデータをシフトする必要があった。しかし、改良版アルゴリズムでは、左右のシフト量の少ない方にデータをずらすので、シフト量は最悪でも配列の要素数の半分程度である。

(1) プログラムの説明文における空欄 A1 から D2 に当てはまる適切な語句を語群から選べ。

語群A　昇順、降順、左、右

語群B　show、search、shift_r、shift_l、sort、main

語群C　バブルソート、選択ソート、挿入ソート、クイックソート、マージソート、計数ソート、逐次探索法、二分探索法、ハッシュ法

語群D　0、1、2、3

(2) 下線部アからウのデータのある位置は、プログラム中でどのように表現されているか、配列の添字として使う式を示して答えよ。ただし「配列の添字として使う式」とは、そのデータを呼び出す際に data[x]と指定する場合の x のことである。なお、式中には、プログラム中の定数 N と、関数 sort 内の変数を用いて良い。

(3) 下線部 i と ii の操作を行っているのはプログラムの何行目か答えよ。

(4) このプログラムが終了するまでに、関数 shift_r と関数 shift_l が呼び出される順番を示せ。ただし、それぞれの関数の呼び出しを R、L で表すこと。

(5) プログラムの実行が 53 行目まで到達した際の、配列 data の内容を data[0]から順に示せ。

(6) この整列アルゴリズムの安定性を検証し、その結果と根拠を答えよ。

(7) 元のアルゴリズムを高速化するよく知られた手法がもう 1 つある。この手法では、データ構造を工夫することによって、ある操作の必要をなくしている。どのようなデータ構造を用いると、どの操作が不要になるのか、理由とともに答えよ。

以下、プログラム

```
1   #include <stdio.h>
2   #define N 20
3
4   void show(int data[], int top){
5       for(int i = 0; i < N; i++) {printf("%d ",data[(top+i)%N]);}
6       printf("¥n");
7   }
8
9   int search(int data[], int a, int left, int right){
10      int mid = (left+right)/2;
11      if(right < left) {return left;}
12      else if(a == data[mid%N]) {return mid+1;}
13      else if(a < data[mid%N]) {return search(data, a, left, mid-1);}
14      else {return    search(data, a, mid+1, right);}
15  }
```

```
16      void shift_r(int data[], int left, int right){
17          while(left < right){
18              data[right%N] = data[(right-1)%N];
19              right--;
20          }
21      }
22      void shift_l(int data[], int left, int right){
23          while(left < right+N){
24              data[left%N] = data[(left+1)%N];
25              left++;
26          }
27      }
28
29      int sort(int data[]){
30          int offset, p, insert, tmp, top = N;
31          for (offset = 1; offset < N; offset++){
32              p = top+offset;
33              insert = search(data, data[p%N], top, p-1);
34              if(insert != p){
35                  tmp = data[p%N];
36                  if(p-insert <= N/2){
37                      shift_r(data, insert, p);
38                  }else{
39                      shift_l(data, p, insert-1);
40                      insert--; top--;
41                  }
42                  data[insert%N] = tmp;
43              }
44          } return top;
45      }
46
47      int main(){
48          int data[N] = {12, 3, 8, 14, 17, 7, 9, 10, 11, 5, 16, 19, 13, 4, 15, 6, 2, 18, 20, 1};
49          show(data, sort(data));
50          return 0;
51      }
```

解答

(1) A1: 昇順、A2: 右
B1: sort、B2: search、B3: shift_r、B4: shift_l
C1: 挿入ソート、C2: 二分探索法、C3: 逐次探索法
D1: 0、D2: 1

(2) (ア) top%N、(イ) p%N、(ウ)insert%N

(3) (i) 36 行目、(ii) 42 行目

(4) R R R R R R R R L R L L R L

(5) 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 1 2 3 4

(6) 安定である。12 行目で同じ値の数値を見つけたときは、その後ろを挿入先に指定しているため元の順番が保持される。また、シフト操作においてもデータ順は保持される。

(7) データの格納に配列でなく連結リストを用いる。連結リストは、ポインタの繋ぎかえにより、まさしくデータの“挿入”が行えるため、データをずらすシフト操作が不必要になる。