デジタル構造とアルゴリズム　中間レポート

福富隆大　学籍番号３５７１４１２１

2024 年 6 月 21 日

1　実験結果

前半 5 個が通常のデータ、後半 5 個が 90 %ソートされたデータである。

10000	バブルソート	213, 255, 225, 227, 213, 138, 114, 117, 120, 119
10000	マージソート	4, 3, 4, 3, 4, 3, 3, 3, 3, 10
10000	基数ソート	4, 4, 4, 4, 4, 7, 4, 4, 4, 4
50000	バブルソート	5229, 5267, 5527, 5284, 5456, 3779, 3775, 3791, 3060, 3098
50000	マージソート	7, 7, 7, 33, 21, 3, 4, 3, 4, 11
50000	基数ソート	7, 7, 7, 33, 21, 3, 4, 3, 4, 11
100000	バブルソート	22452, 19734, 31895, 33828, 33421, 34889, 34389, 34241, 33989, 33904
100000	マージソート	62, 35, 20, 14, 14, 8, 8, 7, 8, 7
100000	基数ソート	47, 23, 23, 23, 24, 23, 23, 25, 22, 24
500000	バブルソート	978848, 935280, 991861, 967596, 927330, 334265, 342269, 808978, 818900, 806195
500000	マージソート	91, 86, 89, 89, 89, 43, 44, 48, 44, 137
500000	基数ソート	117, 116, 109, 119, 116, 121, 116, 118, 120, 118
1000000	バブルソート	979136, 801236, 891136, 899136, 981236, 900124, 800136, 890113, 890124, 890136
1000000	マージソート	232, 207, 199, 192, 187, 127, 117, 97, 117, 127
1000000	基数ソート	243, 203, 236, 213, 223, 243, 205, 243, 207, 223

2　実験結果の考察

今回の課題をバイトの時間の関係で cse ではなく、自分のパソコンで実行した。
パソコンは M3 MacBook Pro で、プロセッサは M3 pro チップ、メモリは 36GB である。
実験結果を見てみると、バブルソート、マージソート、基数ソートの順で実行時間が短くなっていることがわかる。
これは最悪実行時間がそれぞれ $O(n^2)$、$O(n \log n)$、$O(n)$ であることにも一致している。
しかし、データ数が 10 倍になると、バブルソートの実行時間はおおよそ 100 倍になるはずだが、実際にはそれよりも大きくなっている。これは、大量のデータを処理する方はハードウェア的な問題が影響しているのではないかと考えた。

3　プログラムの説明

このプログラムはデータが入っているディレクトリをコマンドライン引数で取得し、そのディレクトリに入っているデータのパスと名前を配列に入れる。
その後、バブルソート、マージソート、基数ソートの順でソートをデータの数だけ行い、データのファイル名、実行時間、result.tex に出力している。

4　ソースコード

```
1   import java.io.FileWriter;
2   import java.io.BufferedReader;
3   import java.io.BufferedWriter;
4   import java.io.File;
5   import java.io.FileReader;
6   import java.io.IOException;
7   import java.util.ArrayList;
8   import java.util.Arrays;
9   import java.util.Comparator;
10  import java.util.List;
11
12  public class Main {
13      public static void main(String[] args) {
14          ArrayList<String> dataText = new ArrayList<String>();// データのファイルの名前を格納
15          ArrayList<String> dataPath = new ArrayList<String>();// データのファイルのパスを格納
16          String dirname = args[0];
17          try (BufferedWriter writer = new BufferedWriter(new
                FileWriter("result.txt", true))) {
18              writer.write(dirname + "\n");
19          } catch (IOException e) {
20              e.printStackTrace();
21          }
22          inputdata(dirname, dataText, dataPath);// ディレクトリの中にあるデータのファイルをに格納dataText
23          for (int index = 0; index < 3; index++) {// バブルソート、マージソート、基数ソート0:1:2:
24              try (BufferedWriter writer = new BufferedWriter(new
                    FileWriter("result.txt", true))) {
```

```
25                      switch (index) {
26                          case 0:
27                              writer.write("バブルソート");
28                              break;
29                          case 1:
30                              writer.write("マージソート");
31                              break;
32                          case 2:
33                              writer.write("基数ソート");
34                              break;
35                          default:
36                              break;
37                      }
38                      writer.write("\n");
39                  } catch (IOException e) {
40                      e.printStackTrace();
41                  }
42                  String[] results = new String[dataPath.size()];
43                  int[] resultsdata = new int[dataPath.size()];
44                  for (int i = 0; i < dataPath.size(); i++) {// データのファイル数だけ繰り返す
45                      String singleDataPath = dataPath.get(i);
46                      String singleDataText = dataText.get(i);
47                      List<Integer> numbers = new ArrayList<>();
48                      try (BufferedReader br = new BufferedReader(new FileReader(singleDataPath))) {// ファイルを読み込む
49                          String line;
50                          while ((line = br.readLine()) != null) {
51                              numbers.add(Integer.parseInt(line));// データのリストを作る
52                          }
53                      } catch (IOException e) {
54                          e.printStackTrace();
55                      }
56                      // レポート用の出力
57                      switch (index) {
58                          case 0:
59                              long start = System.currentTimeMillis();
60                              bubbleSort(numbers);
61                              long end = System.currentTimeMillis();
```

```
62                          results[i] = (singleDataText + "  " + (end
                                - start) + "ms");
63                          resultsdata[i] = (int) (end - start);
64                          break;
65                      case 1:
66                          start = System.currentTimeMillis();
67                          mergeSort(numbers);
68                          end = System.currentTimeMillis();
69                          results[i] = (singleDataText + "  " + (end
                                - start) + "ms");
70                          resultsdata[i] = (int) (end - start);
71                          break;
72                      case 2:
73                          // 桁数をそろえたり、最大値を求めたりする部分は実行時間時間
                                に含めない
74                          List<Integer> formattedNumbers = new
                                ArrayList<>();// 桁数を揃えた数字を
                                格納
75                          for (int num : numbers) {
76                              formattedNumbers.add(Integer.parseInt(
                                    String.format("%08d", num)));
77                          }
78                          start = System.currentTimeMillis();
79                          for (int digit = 1; digit <= 8; digit ++) {
                                // exp桁
                                数:
80                              radixSort(formattedNumbers, digit);
81                          }
82                          end = System.currentTimeMillis();
83                          results[i] = (singleDataText + "  " + (end
                                - start) + "ms");
84                          resultsdata[i] = (int) (end - start);
85                          break;
86
87                      default:
88                          break;
89                  }
90              }
91              try (BufferedWriter writer = new BufferedWriter(new
                    FileWriter("result.txt", true))) {
```

```
92                      Arrays.sort(results, Comparator.comparingInt(String
                            ::length));
93                      for (String str : results) {
94                          writer.write(str + "\n");
95                      }
96                      // 分散を求める
97                      double sum = 0;
98                      for (int num : resultsdata) {
99                          sum += num;
100                     }
101                     double mean = sum / resultsdata.length;// 平均
102                     double squaredDifferenceSum = 0;
103                     for (int num : resultsdata) {
104                         squaredDifferenceSum += Math.pow(num - mean, 2)
                                ;
105                     }
106                     double variance = squaredDifferenceSum /
                            resultsdata.length;
107                     writer.write("分散:" + variance + "\n");// 分散
108                 } catch (IOException e) {
109                     e.printStackTrace();
110                 }
111             }
112         }
113
114         public static void inputdata(String dirname, ArrayList<String>
                dataText, ArrayList<String> dataPath) {// ディレクトリの中にある
                データをに格納dataText
115             File dir = new File(dirname);
116             File[] files = dir.listFiles();
117             if (files != null) {
118                 for (File file : files) {
119                     if (file.isFile()) {
120                         if (!isSortedFile(file)) {// ソートされたファイル以外
                                を格納
121                             dataText.add(file.getName());
122                             dataPath.add(file.getAbsolutePath());
123                         }
124                     } else if (file.isDirectory()) {
125                         // 再帰的にサブディレクトリを処理する
```

```
126                         inputdata(file.getAbsolutePath(), dataText,
                                dataPath);
127                     }
128                 }
129             }
130         }
131
132         public static boolean isSortedFile(File file) {
133             if (file == null || file.getName() == null) {
134                 return false;
135             }
136             return file.getName().contains("sorted");
137         }
138
139         public static void bubbleSort(List<Integer> arr) {// バブルソート
140             int n = arr.size();
141             for (int i = 0; i < n - 1; i++) {
142                 for (int j = 0; j < n - i - 1; j++) {
143                     if (arr.get(j) > arr.get(j + 1)) {
144                         int temp = arr.get(j);
145                         arr.set(j, arr.get(j + 1));
146                         arr.set(j + 1, temp);
147                     }
148                 }
149             }
150         }
151
152         public static void mergeSort(List<Integer> arr) {// マージソート
153             if (arr.size() > 1) {
154                 int mid = arr.size() / 2;
155
156                 ArrayList<Integer> left = new ArrayList<>(arr.subList
                        (0, mid));
157                 ArrayList<Integer> right = new ArrayList<>(arr.subList(
                        mid, arr.size()));
158
159                 mergeSort(left);
160                 mergeSort(right);
161
```

```
162             int i = 0, j = 0, k = 0;// i左の配列のインデックス、:j右の配列のインデックス、:k元の配列のインデックス:
163             while (i < left.size() && j < right.size()) {// 二つの配列を小さいものから順にマージ
164                 if (left.get(i) < right.get(j)) {
165                     arr.set(k, left.get(i));
166                     i++;
167                 } else {
168                     arr.set(k, right.get(j));
169                     j++;
170                 }
171                 k++;
172             }
173
174             while (i < left.size()) {// 一つの配列しか残っていない場合
175                 arr.set(k, left.get(i));
176                 i++;
177                 k++;
178             }
179
180             while (j < right.size()) {// 一つの配列しか残っていない場合
181                 arr.set(k, right.get(j));
182                 j++;
183                 k++;
184             }
185         }
186     }
187
188     public static void radixSort(List<Integer> arr, int digit) {// 基数ソート
189
190         int N = arr.size();
191         int[][] buf = new int[10][N];
192         int[] ctr = new int[10];// それぞれの列にいくつ入っているか例： ctrなら列につのデータが入っている[0]=303
193
194         for (int i = 0; i <= 9; i++) {
195             ctr[i] = 0;
196         }
197
```

```
198             for (int i = 0; i < N; i++) {
199                 int k = val(arr.get(i), digit);
200                 buf[k][ctr[k]++] = arr.get(i);
201             }
202 
203             int t = 0;
204             for (int i = 0; i <= 9; i++) {
205                 for (int j = 0; j <= ctr[i] - 1; j++) {
206                     arr.set(t++, buf[i][j]);
207                 }
208             }
209         }
210 
211         private static int val(int num, int digit) {
212             return (int) (num / Math.pow(10, digit - 1)) % 10;
213         }
214     }
```