# データ構造とアルゴリズム

## 演習課題 3

提出日 2024/07/01

HI4 45 号 山口惺司

1. 問題

挿入ソートについて、ダブル挿入ソートという改良アルゴリズムがある。
この方法について{2, 8, 3, 4, 7, 6, 1, 5}が与えられたとき、ダブル挿入ソートを利用してソートせよ。

2. アルゴリズムの説明

以下のような経過でダブル挿入ソートが行われる。

① {2, 8, 3, 4, 7, 6, 1, 5} が与えられた(偶数個の場合のみ)
② {2, 8, 3, (4, 7), 6, 1, 5} 中央の 2 つを大小順に入れ替え
③ {2, 8, 3, (4, 7), 6, 1, 5} 中央部の両側の 2 つを大小順に入れ替え
④ {2, 8, (3, 4, 7), 6, 1, 5} 中央部の左側を挿入ソートで中央部が昇順になるよう挿入
⑤ {2, 8, (3, 4, 6, 7), 1, 5} 中央部の右側を挿入ソートで中央部が昇順になるよう挿入
⑥ {2, 1, (3, 4, 6, 7), 8, 5} 操作③
⑦ {2, (1, 3, 4, 6, 7), 8, 5} 操作④
⑧ {2, (1, 3, 4, 6, 7, 8), 5} 操作⑤
⑨ {2, (1, 3, 4, 6, 7, 8), 5} 操作③
⑩ {(1, 2, 3, 4, 6, 7, 8), 5} 操作④
⑪ {(1, 2, 3, 4, 5, 6, 7, 8)} 操作⑤
⑫ {(1, 2, 3, 4, 5, 6, 7, 8)} 終了

3. フローチャート

ダブル挿入ソートのフローチャートを図 1 に、挿入ソート関数のフローチャートを図 2 に示す。

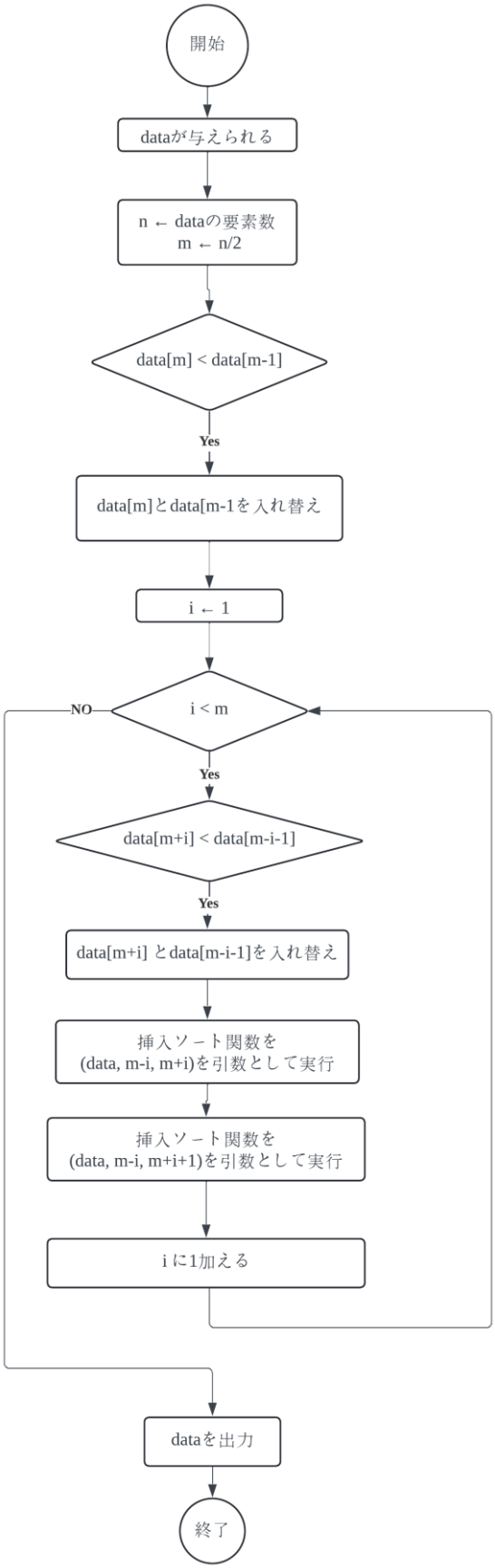


図 1 ダブル挿入ソートのフローチャート

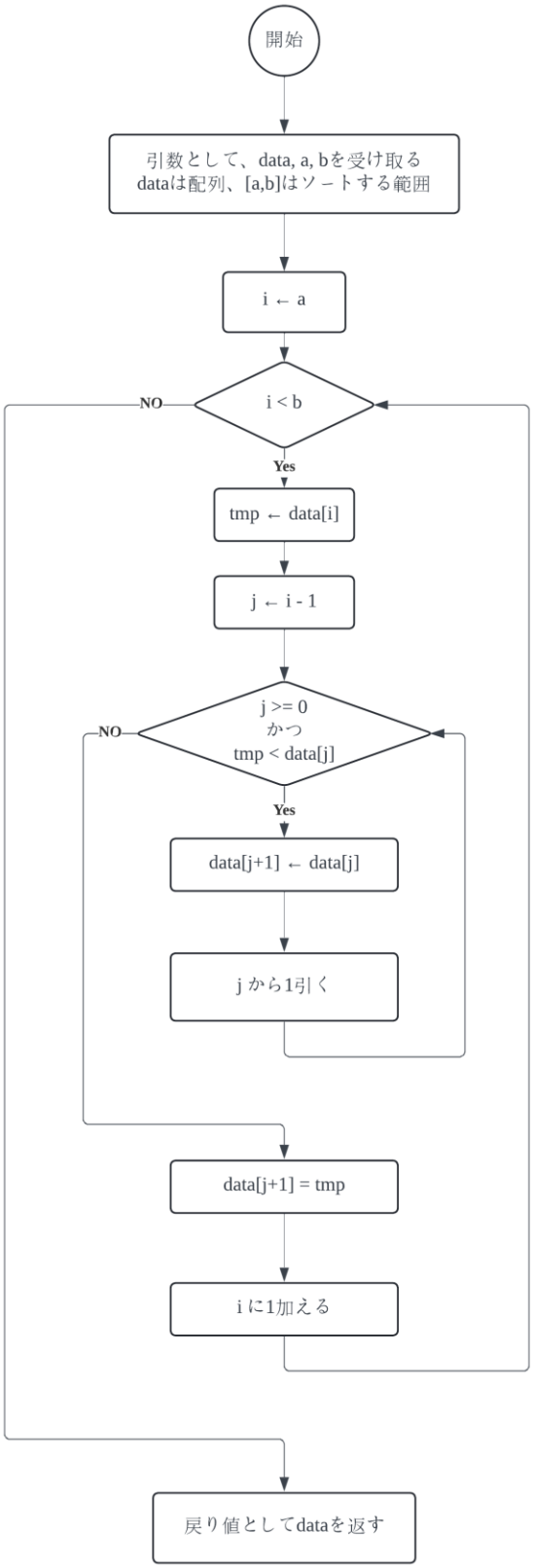


図 2 挿入ソート関数のフローチャート

## 4. 疑似コード

ダブル挿入ソートと挿入ソート関数の疑似コードを以下に示す。

ダブル挿入ソート：

① data が与えられる。
② n=data の要素数、m=data の要素数の半分
③ data[m] < data[m-1]であれば data[m]と data[m-1]を入れ替え
④ i = 1
⑤ i < m でなければ⑩へ進む
⑥ data[m+i] < data[m-i-1]であれば data[m+i] と data[m-i-1]を入れ替え
⑦ 挿入ソート関数に data, [m-i, m+i]を渡す
⑧ 挿入ソート関数に data, [m-i, m+i+1]を渡す
⑨ i に 1 加えて⑤に戻る
⑩ data を出力する

挿入ソート関数：

① data, 範囲[a, b]が与えられる
② i = a
③ i < b でなければ⑧へ進む
④ tmp = data[i], j = i -1
⑤ j >= 0 かつ j >= a-1 でなければへ進む
⑥ tmp < data[j]であれば data[j+1] = data[j], j から 1 減らし、⑤へ戻る、そうでなければ⑦へ進む
⑦ data[j+1] = tmp, ③へ戻る
⑧ data を返す

## 5. プログラム

作成したフローチャートと疑似コードを基に、比較回数と交換回数をカウントする処理を加えた Python プログラムを以下に示す。

```
def double_insertion_sort(data):  #ダブル挿入ソート関数
    chcount, cocount = 0, 0 #比較回数と交換回数
    n = len(data)
    m = int(n / 2)
    if data[m] < data[m - 1]:
        data[m], data[m - 1] = data[m - 1], data[m]
        chcount += 1 # 交換回数を増やす

    for i in range(1, m):
        if data[m + i] < data[m - i - 1]:
```

```
            data[m + i], data[m - i - 1] = data[m - i - 1],
data[m + i]
            chcount += 1 # 交換回数を増やす

        a, b = insertion_sort(data, m - i, m + i)
        chcount += a # 交換回数を増やす
        cocount += b # 比較回数を増やす
        a, b = insertion_sort(data, m - i, m + i + 1)
        chcount += a # 交換回数を増やす
        cocount += b # 比較回数を増やす

    print("交換回数:", chcount)
    print("比較回数:", cocount)

    return data

def insertion_sort(data, a, b): #挿入ソート関数
    chcount, cocount = 0, 0
    for i in range(a, b):
        tmp = data[i]
        j = i - 1

        while j >= 0 and j >= a - 1:
            cocount += 1  # 比較回数を増やす
            if tmp < data[j]:
                data[j + 1] = data[j]
                j -= 1
                chcount += 1  # 交換回数を増やす
            else:
                break

        data[j + 1] = tmp

    return chcount, cocount

def main(): #main 関数
    data = [2, 8, 3, 4, 7, 6, 1, 5] #与えられたデータ
    print("元のデータ:", data)
    sorted_data = double_insertion_sort(data)
    print("ソート後のデータ:", sorted_data)

if __name__ == "__main__":
```

```
    main()
```

## 6. 実行結果

実行結果を図 3〜図 7 に示す。

```
元のデータ: [2, 8, 3, 4, 7, 6, 1, 5]
交換回数: 6
比較回数: 31
ソート後のデータ: [1, 2, 3, 4, 5, 6, 7, 8]
```

図 3 [2, 8, 3, 4, 7, 6, 1, 5]を与えたときの実行結果

```
元のデータ: [1, 2, 3, 4, 5, 6]
交換回数: 0
比較回数: 14
ソート後のデータ: [1, 2, 3, 4, 5, 6]
```

図 4 [1, 2, 3, 4, 5, 6]を与えたときの実行結果

```
元のデータ: [0, -1, 2, 3, -2, 2, 9, 4]
交換回数: 7
比較回数: 30
ソート後のデータ: [-2, -1, 0, 2, 2, 3, 4, 9]
```

図 5 [0, -1, 2, 3, -2, 2, 9, 4]を与えたときの実行結果

```
元のデータ: [8, 3, 4, 1, 2, 3, 4, 2]
交換回数: 7
比較回数: 29
ソート後のデータ: [1, 2, 2, 3, 3, 4, 4, 8]
```

図 6 [2, 8, 3, 4, 7, 6, 1, 5]を与えたときの実行結果

```
元のデータ: [3, 5, 2, 1, 0]
交換回数: 3
比較回数: 6
ソート後のデータ: [1, 2, 3, 5, 0]
```

図 7 [3, 5, 2, 1, 0]を与えたときの実行結果

## 7. 考察

図 3~6 では正しくソートできていることがわかる。
図 7 では与えられたデータの要素数が奇数であったため、正しくソートできなかった。
また、図 3, 4, 6 を見ると、与えられたデータが同じ要素数でも比較回数と交換回数が違うことが分かった。

## 8. 感想

コードが挿入ソートと比べて明らかに長くなっているが、挿入ソートよりダブル挿入ソートの方がより効率化されていることが驚きである。