データ構造とアルゴリズム

レポート課題 5

提出日: 2024/11/08

HI4 45 号山口惺司

1. 課題内容

乱択法を用いてクイックソートを改良しなさい.

2. アルゴリズム

1. クイックソートについて

クイックソートは効率的な比較ソートアルゴリズムの 1 つで，大規模なデータセットを効率的に並べ替えるために用いられる.
また，「分割統治法」に基づいており，配列を再帰的に分割し，それぞれをソートしていくことで全体をソートさせる.

2. クイックソートの手順

① 配列の先頭をピボットとして選び，配列を「ピボットより小さい要素」と「ピボットより大きい要素」に分ける.
② ピボットを基に①のように分割し新たな配列を作成する.
③ ②で分割された配列それぞれに，再び①②のようにクイックソートを適用する．これを再帰的に繰り返すことで，配列全体がソートされる.
④

3. フローチャート

クイックソートのフローチャートを図 1 に示す.

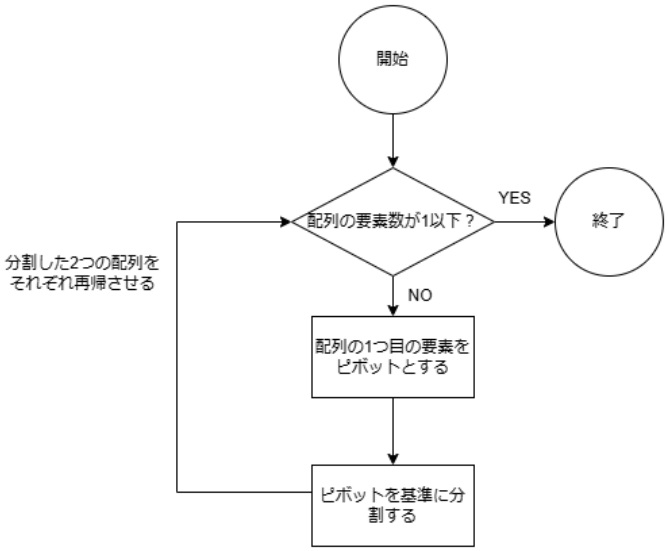

図 1 クイックソートのフローチャート

4. クイックソートの改良法

教科書に書いてある quick_sort.py を改良するには，ピボットを最初の要素ではなく，ランダムに決める

という方法がある.
ピボットをランダムな要素にすることによって, 平均的に高速にすることができる.
このようにランダム化によって平均的によい性能を期待できるアルゴリズムを**乱択アルゴリズム**という.

3. スクリプト

1. クイックソート

改良前のクイックソートのスクリプトを以下に示す.

```
def quicksort(A):
    if len(A) < 2:
        return A
    p = A[0]
    X, Y = divide(p, A[1:])
    return np.concatenate([quicksort(X), [p], quicksort(Y)])

def divide(p, A):
    A = np.array(A)
    X = A[A < p]
    Y = A[A >= p]
    return X, Y
```

2. 改良版クイックソート

改良後クイックソートのスクリプトを以下に示す.

```
def randomized_quicksort(A):
    if len(A) < 2:
        return A
    pivot_index = random.randint(0, len(A) - 1)

    p = A[pivot_index]
    A[0], A[pivot_index] = A[pivot_index], A[0]
    X, Y = divide(p, A[1:])
    return np.concatenate([randomized_quicksort(X), [p], randomized_quicksort(Y)])

def divide(p, A):
    A = np.array(A)
    X = A[A < p]
    Y = A[A >= p]
    return X, Y
```

3. 元のクイックソートと改良版クイックソートを比較

今回は元のクイックソートと改良版クイックソートを比較するため，以下の手順でソートを行う．

① 0〜99 までの整数をランダムに 100 個生成し data リストに入れる
② data をクイックソートし，実行時間を計測する．
③ ①②を 100 回繰り返す．
④ 実行時間の分散を調べる
⑤ ③④を 1000 回繰り返し，実行時間の分散をグラフにする．
⑥ それぞれのソートの平均実行時間と実行時間の分散平均の値を表示する．

以上の手順でソートを行うためのスクリプトを以下に示す．

```
import random
import time
import matplotlib.pyplot as plt
import numpy as np

# 従来のクイックソート
def quicksort(A):
    if len(A) < 2:
        return A
    p = A[0]
    X, Y = divide(p, A[1:])
    return np.concatenate([quicksort(X), [p], quicksort(Y)])

# 乱択法を用いたクイックソート
def randomized_quicksort(A):
    if len(A) < 2:
        return A
    pivot_index = random.randint(0, len(A) - 1)

    p = A[pivot_index]
    A[0], A[pivot_index] = A[pivot_index], A[0]
    X, Y = divide(p, A[1:])
    return np.concatenate([randomized_quicksort(X), [p], randomized_quicksort(Y)])

# 分割関数
def divide(p, A):
    A = np.array(A)
    X = A[A < p]
    Y = A[A >= p]
    return X, Y

# main 関数
def main():
```

```
#実行時間と分散を入れるリスト
data_num = [[], []] #ave var
data_num_rand = [[], []] #ave var

for j in range(1000):
    data_time = []
    data_time_rand = []
    for i in range(100):
        random.seed()
        data = np.random.randint(0, 100, 100)  #配列の作成

        start_time = time.time()
        quicksort(data.copy())
        end_time = time.time()
        data_time.append(end_time - start_time)

        start_time = time.time()
        randomized_quicksort(data.copy())
        end_time = time.time()
        data_time_rand.append(end_time - start_time)

    data_num[0].append(np.average(data_time))
    data_num_rand[0].append(np.average(data_time_rand))
    data_num[1].append(np.var(data_time))
    data_num_rand[1].append(np.var(data_time_rand))

print(data_num[0])
print(data_num_rand[0])
print(data_num[1])
print(data_num_rand[1])

print("改良前平均実行時間: ", np.average(data_num[0]))
print("改良後平均実行時間: ",np.average(data_num_rand[0]))
print("改良前実行時間の分散平均: ",np.average(data_num[1]))
print("改良後実行時間の分散平均: ",np.average(data_num_rand[1]))

l = len(data_num[0])

#グラフ
plt.scatter(range(0, l), data_num[1], color="orange", label="Before", s = 10)
plt.scatter(range(0, l), data_num_rand[1], color="blue", label="After", s = 10)
```

```
        plt.xlabel("Run Count")
        plt.ylabel("Variance of Runtime")
        y_max = np.average(data_num[1])*3
        plt.ylim(0, y_max)
        plt.legend()
        plt.show()

    if __name__ == "__main__":
        main()
```

4. 実行結果

まず、元のクイックソートと改良版のクイックソートが正しくソートされているかを確認した実行結果を図2~4に示す.

また、実行したコードは以下のとおりで 100 個のランダムな要素を持った配列をソート関数に渡している。

```
data = np.random.randint(0, 100, 100)
print("改良前クイックソート: ", quicksort(data.copy()), sep = "¥n")
print("改良後クイックソート: ", randomized_quicksort(data.copy()), sep = "¥n")
```

```
改良前クイックソート:
[ 0  1  2  2  3  5  5 11 12 12 14 15 15 15 16 16 23 23 24 25 26 26 27 30
 30 31 31 31 32 34 36 38 38 39 39 39 41 41 41 42 43 43 46 47 48 49 50 52
 53 55 56 57 58 61 63 64 65 66 67 68 71 71 71 72 73 76 76 77 77 78 79 79
 79 80 81 83 83 84 84 85 86 86 86 88 88 90 90 90 92 92 93 93 93 95 96 96
 97 98 98 99]
改良後クイックソート:
[ 0  1  2  2  3  5  5 11 12 12 14 15 15 15 16 16 23 23 24 25 26 26 27 30
 30 31 31 31 32 34 36 38 38 39 39 39 41 41 41 42 43 43 46 47 48 49 50 52
 53 55 56 57 58 61 63 64 65 66 67 68 71 71 71 72 73 76 76 77 77 78 79 79
 79 80 81 83 83 84 84 85 86 86 86 88 88 90 90 90 92 92 93 93 93 95 96 96
 97 98 98 99]
```

図 2 実行結果 1：正しくソートが行われているかの確認

```
改良前クイックソート:
[ 0  1  1  1  1  1  2  3  4  7  8  8 10 11 12 12 16 17 17 17 19 21 21 21
 22 24 26 28 28 30 31 32 34 34 34 35 35 36 36 37 40 40 40 41 41 43 43 43
 44 44 46 47 48 48 49 49 50 50 51 51 52 53 54 56 58 58 58 58 63 63 64 65
 66 66 68 68 69 71 72 73 74 76 76 78 79 80 82 84 85 86 86 87 88 89 91 93
 93 96 97 98]
改良後クイックソート:
[ 0  1  1  1  1  1  2  3  4  7  8  8 10 11 12 12 16 17 17 17 19 21 21 21
 22 24 26 28 28 30 31 32 34 34 34 35 35 36 36 37 40 40 40 41 41 43 43 43
 44 44 46 47 48 48 49 49 50 50 51 51 52 53 54 56 58 58 58 58 63 63 64 65
 66 66 68 68 69 71 72 73 74 76 76 78 79 80 82 84 85 86 86 87 88 89 91 93
 93 96 97 98]
```

図 3 実行結果 2：正しくソートが行われているかの確認

```
改良前クイックソート:
[ 2  3  3  3  3  4  4  4  4  5  7  7 10 12 12 12 12 13 13 14 14 15 15 16
 17 18 18 20 23 24 26 26 26 27 27 27 28 30 31 32 32 32 33 34 35 36 36 36
 37 40 43 43 44 44 45 45 47 48 49 51 52 52 53 53 54 56 57 59 59 60 62 62
 62 63 63 63 65 66 66 71 72 74 74 75 80 80 82 82 86 89 89 90 91 93 93 94
 97 98 99 99]
改良後クイックソート:
[ 2  3  3  3  3  4  4  4  4  5  7  7 10 12 12 12 12 13 13 14 14 15 15 16
 17 18 18 20 23 24 26 26 26 27 27 27 28 30 31 32 32 32 33 34 35 36 36 36
 37 40 43 43 44 44 45 45 47 48 49 51 52 52 53 53 54 56 57 59 59 60 62 62
 62 63 63 63 65 66 66 71 72 74 74 75 80 80 82 82 86 89 89 90 91 93 93 94
 97 98 99 99]
```

図 4 実行結果 3：正しくソートが行われているかの確認

次に、改良版クイックソートが正しく改良されているかを確認した実行結果を図 5~10 に示す。
横軸が実行回数，縦軸が実行時間の分散である．

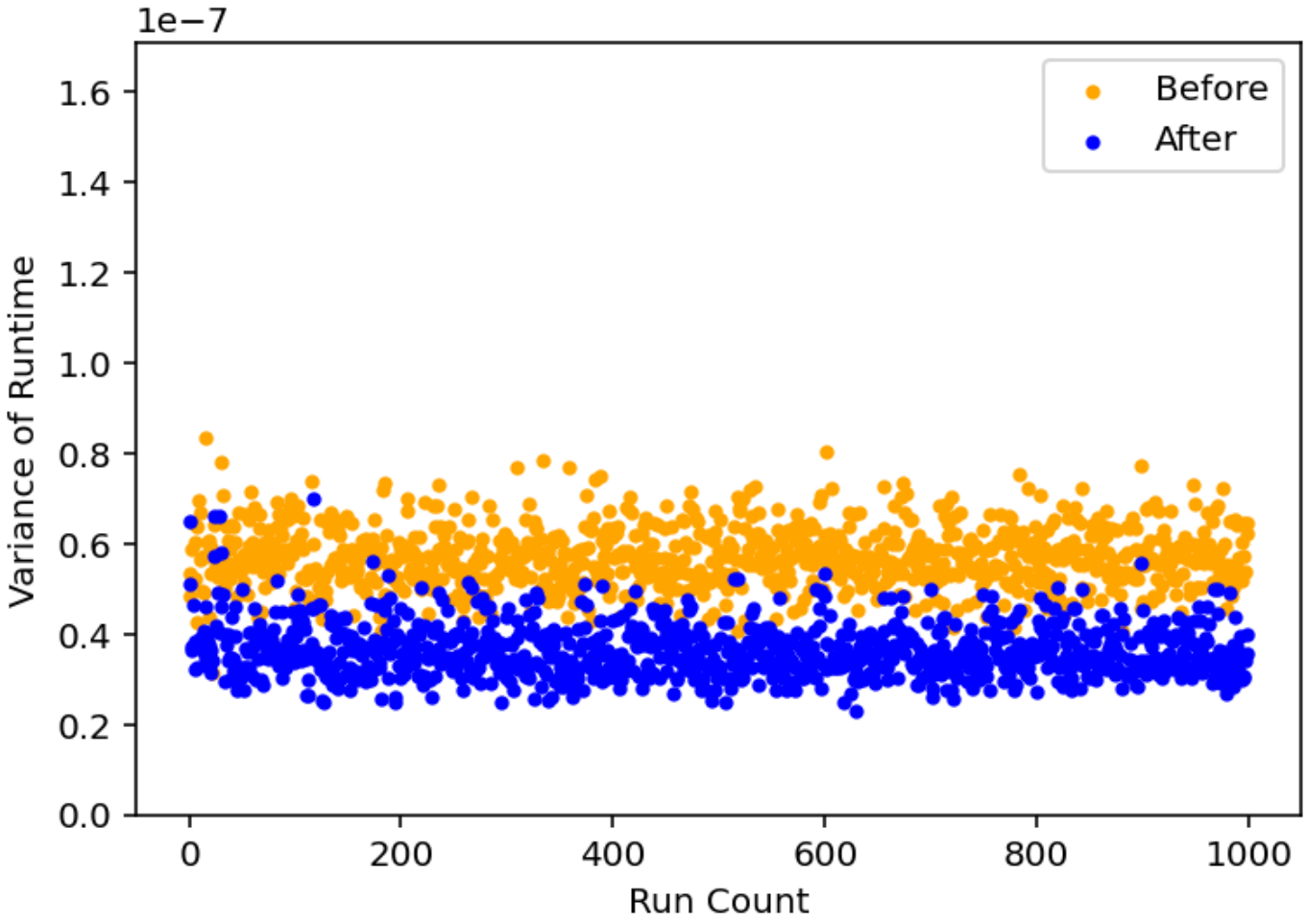

図 5 実行結果 1：実行時間の分散のグラフ

```
改良前平均実行時間:   0.0004244818282127381
改良後平均実行時間:   0.0005087746548652649
改良前実行時間の分散平均:   5.69839546808737e-08
改良後実行時間の分散平均:   3.6381477488004066e-08
```

図 6 実行結果 1：それぞれのソートの平均実行時間と実行時間の分散平均

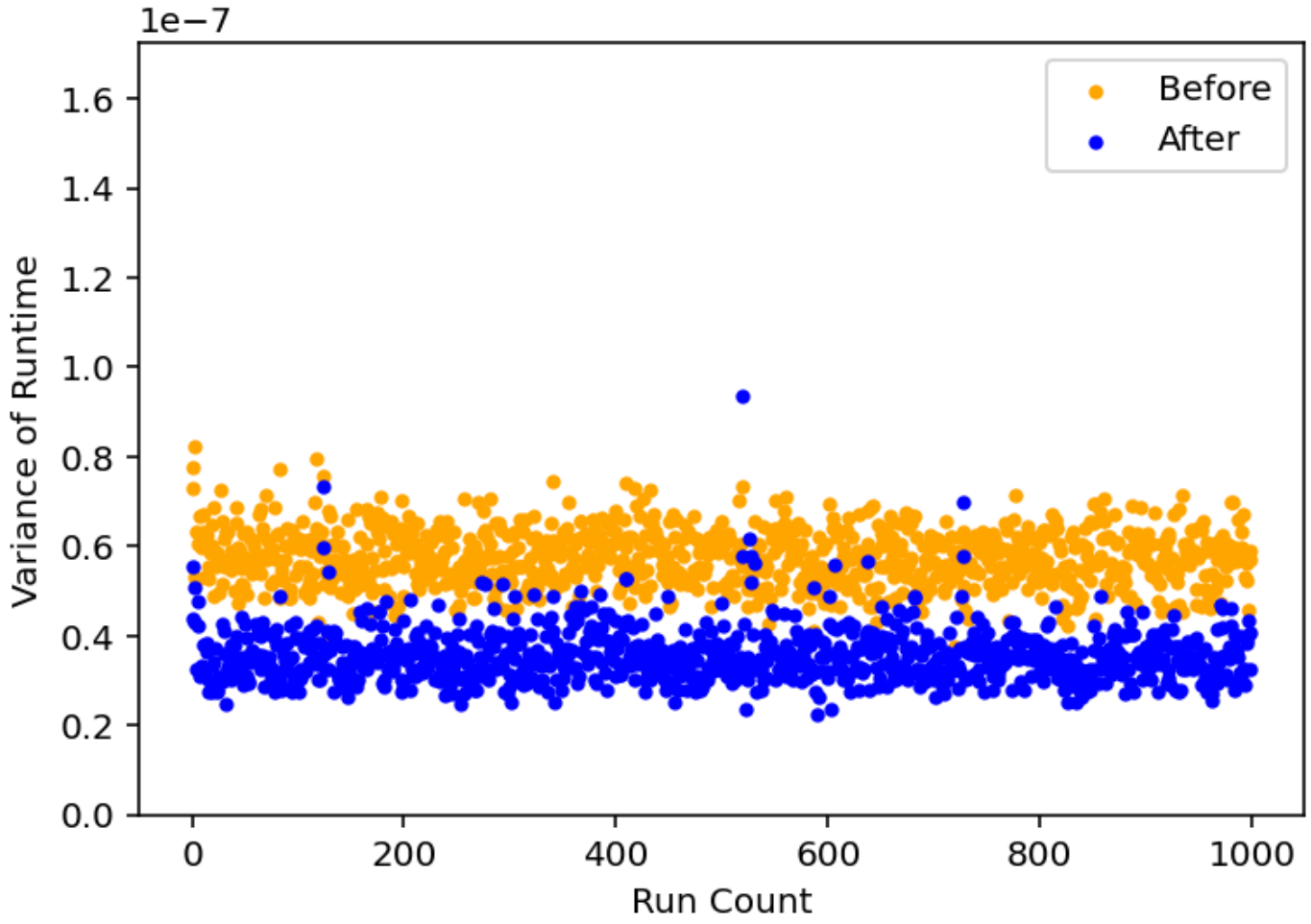

図 7 実行結果 2：実行時間の分散のグラフ

```
改良前平均実行時間:  0.0004225851845741272
改良後平均実行時間:  0.0005067645835876465
改良前実行時間の分散平均:  5.749770259913589e-08
改良後実行時間の分散平均:  3.562804824639443e-08
```

図 8 実行結果 2：それぞれのソートの平均実行時間と実行時間の分散平均

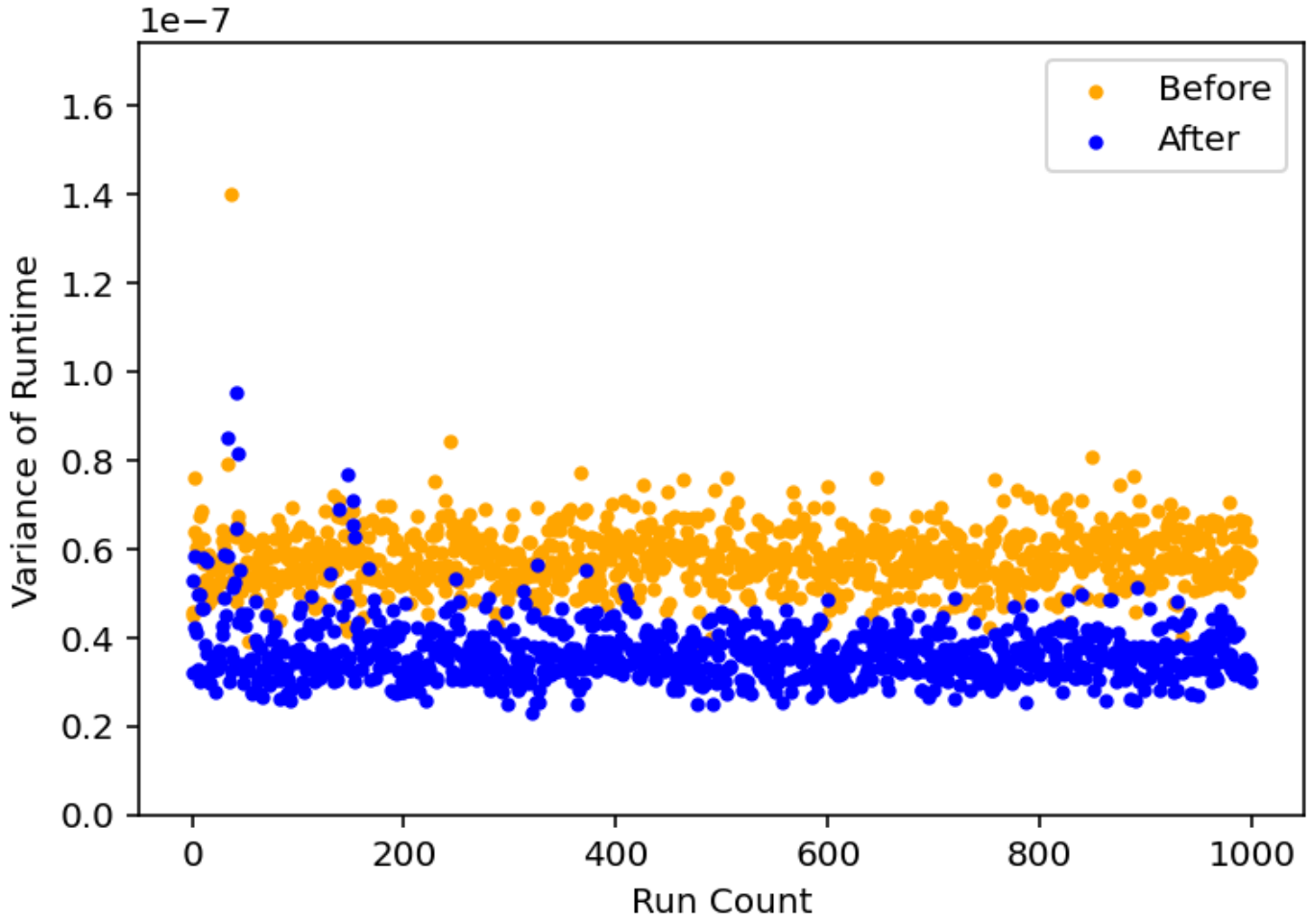

図 9 実行結果 3：実行時間の分散のグラフ

```
改良前平均実行時間:   0.0004247214746475219 4
改良後平均実行時間:   0.0005075844097137452
改良前実行時間の分散平均:   5.8055747079345106e-08
改良後実行時間の分散平均:   3.655747722502838e-08
```

図 10 実行結果 3：それぞれのソートの平均実行時間と実行時間の分散平均

5. 考察

図 2~4 より、どちらのクイックソートも正しくソートができていると言える。
図 5~10 より、改良後のクイックソートは改良前より平均実行時間が大きくなっているが、実行時間の分散平均が改良前より小さくなっているため、改良することによって安定化できていると言える。
また、改良後のクイックソートの平均実行時間が遅くなっている理由としてはピボットを選択するときに、ランダム関数を使っていて、その処理に時間がかかっているのではないかと推測する。