プログラミング演習I第9回レポート

学籍番号：2364902

名前：キム　ギュソク

(1) 課題番号と課題内容

A. 基本課題 2

i. クイック・ソート過程におけるデータの比較回数とデータの入れ換え回数を数え、出力する。 データを出力ファイルに出力する関数にし、整列したデータをファイルに 書き込めるようにする。なお、プログラムの実行時に出力ファイル名もコマンド引数で指定できるようにする。

(2) フローチャートまたは疑似言語によるアルゴリズムの記述

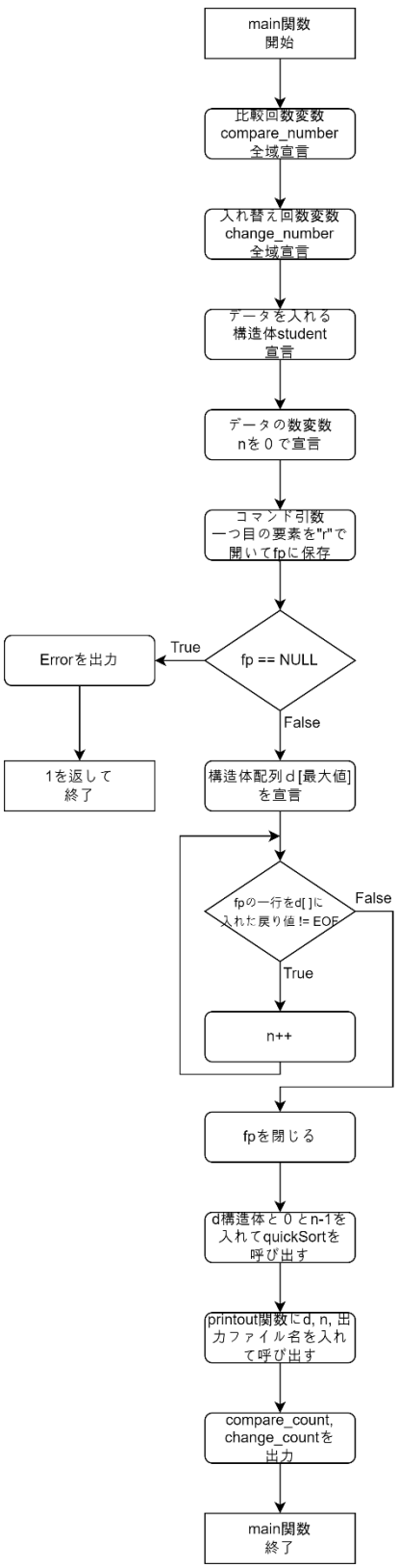


図 1。main関数フローチャート

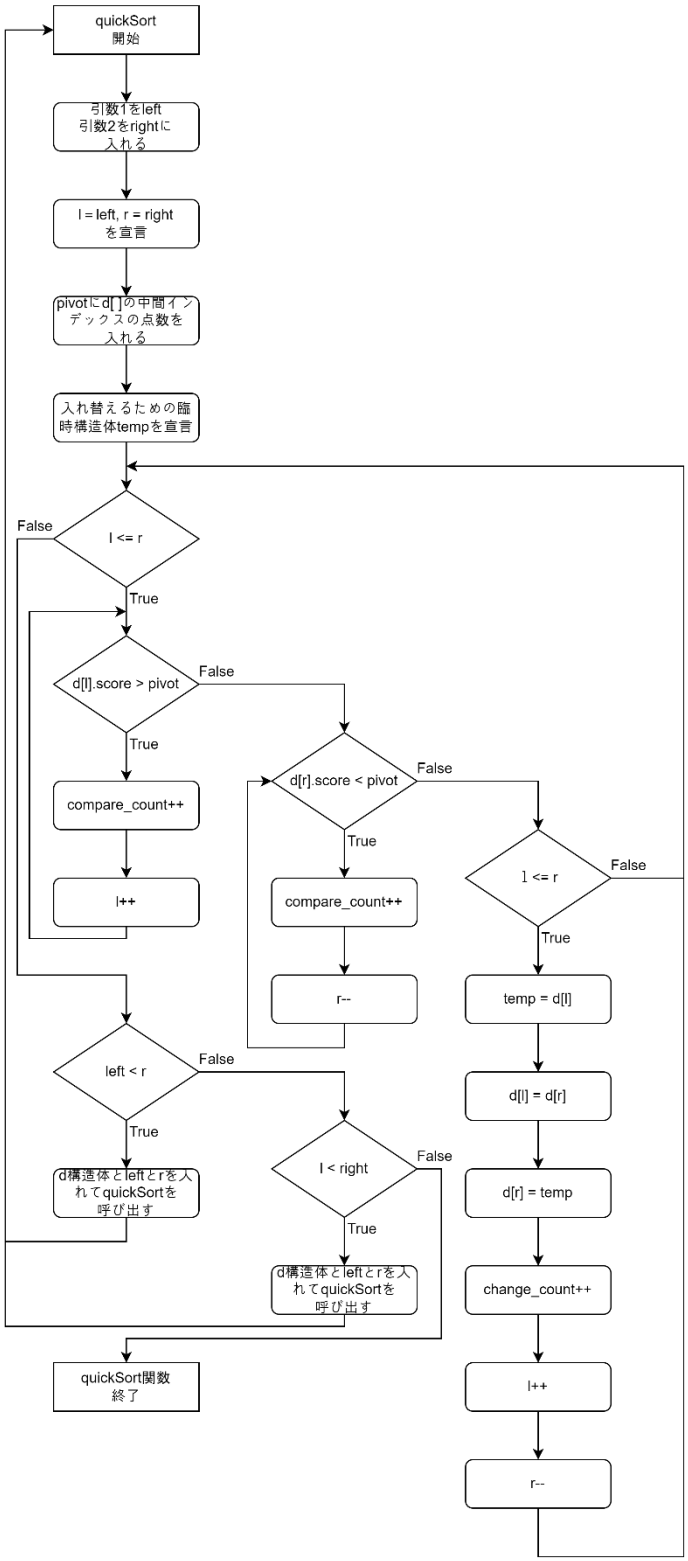


図 2。quickSort関数フローチャート

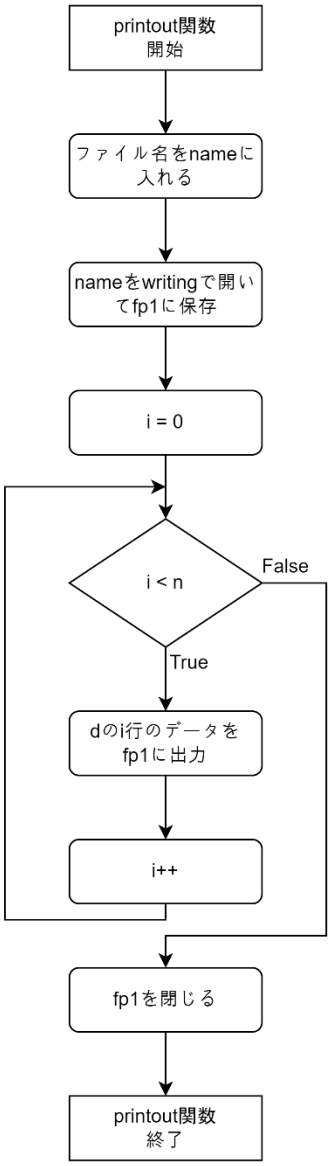


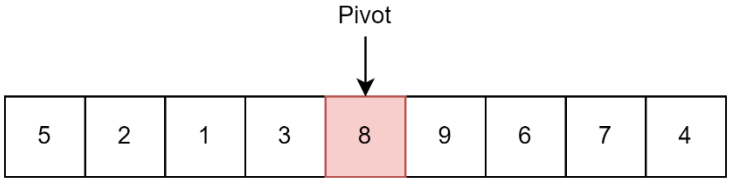
図 3。printout関数フローチャート

(3) アルゴリズムが正しいことの説明

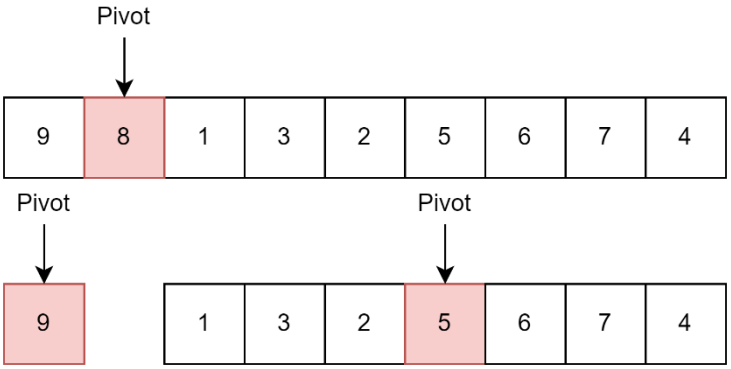
A. 正当性

i. クイック・ソートはデータの中で任意のデータを選んでそれを基準として大きいデータと小さいデータに分ける作業を再帰的に行うアルゴリズである。例えば、次のようなデータがある時に

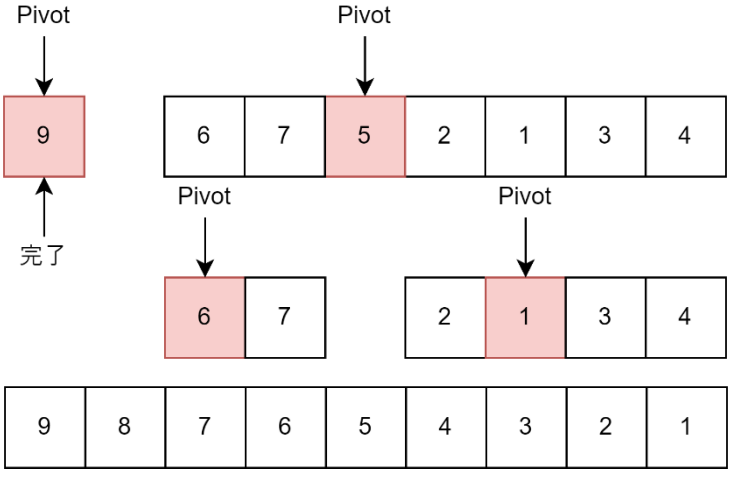
今回はその任意のデータの位置を真ん中のインデックスを設定したのでこの例では8がpivotとして働く。



その後、今回は降順で整列するのでPivotである8より大きい値をすべて左側にそして、8より小さい値を右側に集める



左側と右側を再帰してまた真ん中のインデックスをPivotとして設定して同じ作業を行う。



結果的にこのように降順整列されることが分かる。今回はファイルのデータから点数を読み取って点数を比較しながら構造体配列の順番を入れ替えているので例の単純な数字が構造体に変わるだけなので全く同じ過程である。よって、このアルゴリズは正しく整列されることがわかる。

B. 停止性

i. 今回のアルゴリズで使われているループはデータの数を数えるためのループと左側のPivotより大きいな値を探すためのループ、右側のPivotより小さい値を探すためのループ、全てのデータを右と左に揃えるための外側のループ、次の段階に入る再帰的なループがある。

1. データ数ループ
    A. ファイルを読み取るときにファイルの終わりを読み取ったらEnd Of FileであるEOFを返すので戻り値がＥＯＦである条件を立てると最終的にはファイルの最後の地点にたどり着いて停止する。
2. 左側のループ
    A. Pivotより小さい数字が出ると止まるように条件を付けるとPivotより大きい数字になった時にループは停止する。
3. 右側のループ
    A. 左側のループと反対の条件を付けることでPivotより大きいなデータを発見したときに停止する。
4. 外側のループ
    A. Pivot大きいデータは左側に小さいデータは右側に移動出来たら停止するように条件を付ける必要がある。よって、leftから初めたｌとrightから始まったｒが同じかｒが大きいときに左側にはもうPivotより小さいデータは存在しない。また、右側も同じくPivotより大きいデータは存在しないことを意味するので停止する。
5. 再帰的ループ
    A. 全てのデータが整列された時に停止する必要がある。よって、左側の出発点が右から移動したｒより大きくかつ右側の出発点が左から移動したｌより小さいときにお互いに最後の部分まで整列を行っていることを意味するので停止するようにする。

よって、全てのループが無限ループに入ることなく正しい結果が得られた時に停止するように設計されていると言える。

(4) ソース・プログラムの説明

A. コマンド引数でもらったファイル名のファイルからデータを読み取って構造体に保存するためにstudenという名前の構造体を学籍番号と名前と点数をそれぞれ型に合わせて宣言する。その後、各関数のプロトタイプを宣言してmain関数でファイルをfopenしてファイルポインタ変数fpに入れる。もし、失敗したときにわかるようにif文を利用して１を返して終了する。成功すると、構造体の各データを読み取るための構造体配列をデータの最大値で宣言する。その後、繰り返し文を利用してデータの数をnに保存する。そして、構造体dとleftインデックスとrightインデックスを引数に入れてquickSort関数を呼び出す。quickSort関数では、leftをl変数に入れて、right値をr変数に入れる。そして、引数として入ったleftとrightの真ん中インデックスのscore点数をpivotに入れて宣言する。その後、単純に数字を入れ替えることではなく構造体そのものを入れ替える必要があるので臨時構造体tempを宣言する。pivotを基準に左と右がpivotより大きい点数の構造体と小さい構造体になるためのfor文を作る。その中にpivotより小さい点数が現るまでに l を増やしてpivotより大きい点数のインデックスを順序に探す。rも同じく小さい点数を持っているインデックスを探す。その後、大きい点数のインデックスが小さい点数のインデックスより右側にある時にお互いに入れ替える。その後、次のインデックスにわたるために＋＋とーーをする。最後に、pivotに分けた二つの部分を各自再帰的に引数に入れて呼び出す。その後、コマンド引数でもらった名前のファイルを生成してそこにデータを出力する。

(5) 考察

A. 第8回で作った単純選択整列と比較してみるとquickSortアルゴリズの効率性を確認することができると思う。2240個のデータの場合単純選択整列では比較回数が2,507,680回で入れ替え回数は2,239であった。それに比べて、quickSortの場合比較回数は15,880回で入れ替え回数は7,970である。比較演算や入れ替える演算はプログラムの実行時間に大きく関わっているのでより多くの比較回数と入れ替え回数を持っているアルゴリズが非効率的で多くのデータを使うときに時間をたくさん使ってしまうことがわかる。実際に、応用課題３で行っているようにデータ数が12,800の場合単純選択整列は整列が終わるまで166.362ミリセカンドがかかった。それに比べて、quickSortは2.923ミリセカンドかかった。実行時間が約５７倍の差があることがわかる。この効率は今のようなビッグデータ社会で必要不可欠な要素である。また、このような差が出る理由について考えたいと思う。単純選択整列の場合全ての要素の中で一番max_numberを探してそれを順序に整列するので毎回最大値を探すときに全てのデータを比較する必要がある。しかし、quickSortの場合pivotより大きい数字や小さい数字が出ると比較を止めてお互いに入れ替えるので全てのデータを比較する必要がない。また、pivotを利用して二つの部分を分けて処理を行っているので並列的な処理を行うので多数の処理を同時に行うためより早く処理を完了することができると思う。実際に論理回路の

授業で習っているように段数が多い回路は効率的ではない回路である。並列的な処理を行うとこのような段数が少なくなる。よって、並列的な処理はより効率的なアルゴリズムであり、実際に処理時間を見るとはっきりわかる。

(6) 感想

A. アルゴリズムの効率性とアルゴリズムを変えることで起こる時間的な効率性をわかるようになった。また、この分野はまだまだ発展可能な要素があると思うのでより効率的なアルゴリズムがこれからも出ると思う。いつかはquickSortより処理時間が早い革新的なアルゴリズムが出ると思う。また、そのようなアルゴリズムがもたらす社会の変化もまた期待される。