考察

　バブルソード、選択ソートと挿入ソートの三つのソートのステップについて考察する。

　まず二つのランダムの数値列を用意して、それぞれの三つのアルゴリズムのステップ数を出力してみた。三つのアルゴリズムの短い数値列と長い数値列に対するステップ数を考察するために、二つの数値列は自然数10個と自然数70　個と設定した。

　ステップ数を出力するために、それぞれのプログラミングのswapの後に”step++”を入れ、入れ替えを一回行ったらステップ数に1を足す。

　ステップ数を出力する結果を表1にまとめる。

表１．各アルゴリズムのステップ数

|  |  |  |  |
| --- | --- | --- | --- |
| 自然数個数 | バブルソード | 選択ソート | 挿入ソート |
| 10 | 32 | 8 | 32 |
| 70 | 988 | 64 | 988 |

　結果から分かるように、バブルソートと挿入ソートのステップ数が同じであるが、選択ソートのステップ数がそれらより小さくなっている。その原因について考察する。

　ソートには、「比較」と「入れ替え」の二つの過程がある。バブルソートは、a[0]とa[1]を比較して、a[0]<a[1]であれば交換し、それからa[1]とa[2]、a[2]とa[3]…a[n-2]とa[n-1]について同様に比較して、必要があったら交換を行う。結果として、一番小さい数値がa[n-1]の位置に移動される。そのあとa[0]～a[n-2]について同様なアルゴリズムを繰り返し、最小値をどんどん最後尾に移動することで、大きい順で数値列を並び替える。

　挿入ソートは、先頭の要素から整列済みとするという考え方で行っている。まずa[0]のみ整列済みとし、その次のa[1]と比較して、a[0]が小さければ交換する。そうするとa[0],a[1]が整列済みとなって、そのあとa[2]を整列済みの部分と最後尾から比較し、適当な位置に挿入する。

　二つの方法を見ると、バブルソートは挿入ソートと比べて、比較の回数が多いが、交換を行う回数が同じである。つまりステップが同じである。

　一方、選択ソートは、a[1]~a[n-1]から最大値を選択して、それをa[0]と比較して、a[0]が小さければ交換することで、数値列の最大値を先頭に持つ。そのあと同じアルゴリズムを繰り返し、大きい順で並び替える。比較の回数は他の二つの方法と比べて、それほど小さくないが、交換を行う回数が少ない。

　三つの方法を比べると、数値列が短いとき、どちらでも効率的に並び替えることができるが、数値列が長くなると、バブルソートと挿入ソートは比較回数と交換回数が極めて大きくなるので、効率が下がる。優位性として、データ量が小さいとき三つの方法はだいたい同じであるが、データ量が大きくなると、選択ソート＞挿入ソート＞バブルソートで選択ソートは一番効率的である。

\*用意した数値列

10個:34 0 564 12 67 80 15 65 209 1256

70個: 202 381 734 498 982 727 580 401 345 917 633 229 179 767 911 277 703 67 813 385 777 462 243 886 32 964 149 591 691 785 560 747 408 889 237 351 649 200 742 427 675 464 144 936 159 95 515 41 899 363 189 216 148 72 760 40 310 530 411 498 852 15 657 69 686 146 957 221 232 8