Kadai4-1,4-2

J2200002 青山和樹

Kadai4-1

- プログラム

  別途提出

- 実行結果

```
Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
1000 > /dev/null
データ数: 1000 シェルソートの実行時間 = 0.000044[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
5000 > /dev/null
データ数: 5000 シェルソートの実行時間 = 0.000218[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
10000 > /dev/null
データ数: 10000 シェルソートの実行時間 = 0.000469[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
50000 > /dev/null
データ数: 50000 シェルソートの実行時間 = 0.010432[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
100000 > /dev/null
データ数: 100000 シェルソートの実行時間 = 0.007147[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
500000 > /dev/null
データ数: 500000 シェルソートの実行時間 = 0.039094[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./shellB ../under10M.dat
1000000 > /dev/null
データ数: 1000000 シェルソートの実行時間 = 0.083383[秒]
```

```
Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
1000 > /dev/null
データ数: 1000 クイックソートの実行時間 = 0.000035[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
5000 > /dev/null
データ数: 5000 クイックソートの実行時間 = 0.000204[秒]
```

```
Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
10000 > /dev/null
データ数: 10000 クイックソートの実行時間 = 0.000544[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
50000 > /dev/null
データ数: 50000 クイックソートの実行時間 = 0.002412[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
100000 > /dev/null
データ数: 100000 クイックソートの実行時間 = 0.006292[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
500000 > /dev/null
データ数: 500000 クイックソートの実行時間 = 0.021945[秒]

Azuma-ya:~/Gunma-u/cs-2/lecture4/kadai4-1% ./quick ../under10M.dat
1000000 > /dev/null
データ数: 1000000 クイックソートの実行時間 = 0.046625[秒]
```

- プログラムの流れ
  - shellA
  - shellB

    プログラムの根幹である shell_sort を説明する。まず最初に、k（間隔）の設定をする。これは配列の長さを超えない最大の`3*k+1`の大きさで初期化している。なぜ`3*k+1`であるかというとこれが Knuth 版のシェルソートであるからだ。これでなくても良い。

    また、この k は while 文を使って`k /= 3`をすることによって、処理を実行するごとに間隔を小さくしている。内部の for 文では間隔 k を使って i を初期化し、最後の要素になるまでループしている。このとき、i 番目の要素の値と、i から k を引いた(= j)番目の要素の値を比較している。i 番目の要素の値の方が小さい時、j 番目の要素を j に k を足した番目に代入する。その時 j を k 引いた分の値に更新する。これを j が 0 より大きいときまで繰り返すことによって、部分的にソートしている。

    最初にあったとおり間隔 k は小さくなっていき、最終的に 0 になったときに配列は完全にソートされる

- 考察

  考察

Kadai4-1

- プログラム

  別途提出

- 実行結果

- プログラムの流れ
  - select
  - lselect

    最初に適当な数字（50）を用意し、その値よりも配列の長さが小さければ、単に配列をソートし、k 番目を返している。

    50 より配列の長さが大きいとき、配列を 5 個ずつの要素からなるグループが並んだものと考え、各 5 つの要素からなるグループをソートする。次に、5 つの要素からなるグループの中央値、つまり要素の 3 番面の値を新たな medians という配列に格納している。ここで medians に対して再帰的に l_select 関数を呼んでいる。ここで、l_select 関数の第 3 引数には配列の長さを 10 で割ったものを与えている。これは medians の配列の長さが全体の配列の長さを 5 で割ったものであることから分かるように、medians の配列の真ん中を指している。よって最終的に medians の中央値を得ることができる。

    次に、for 文を使って配列の最初から見ていき、中央値より小さいもの、等しいもの、大きいものの 3 つの配列に格納していく。またこのとき、それぞれの配列の長さを取得しておく。

    最後に if 文を使って、k とそれぞれの配列の長さを比較していく。もし k が中央値より小さい要素の配列の長さより小さい場合、再帰的に l_select 関数を呼ぶ。第 1 引数には中央値より小さい要素の配列、第 2 引数はその配列の長さ、第 3 引数は k だ。よって、最終的に k 番目の要素を偉る。

    次に k が中央値より小さい要素の配列の長さと、等しい要素の配列の長さを足したものより小さい時、これは k が中央値と等しい配列の中のインデックスを指すこととなるため、戻り地として直ちに中央値を返す。

    以上の条件に当てはまらなかった場合、つまり k が中央値より大きい要素の配列のインデックスを指す場合は、再帰的に l_select を呼ぶ。第 1 引数には中央値より大きい要素の配列、第 2 引数はその配列の長さ、第 3 引数は`k - size_S1 - size_S2`だ。これによって配列全体に対しての k 番目から、中央値より大きい要素の配列に対して同じインデックスを指すようにしている。よって、最終的に k 番目の要素を得る。

- 考察

  考察