アルゴリズムとデータ構造

氏名：津波古正輝
学籍番号：075739A
締め切り：7月11日
提出日：8月2日

選択法、挿入法、マージソート、クイックソートのプログラムを開発しなさい。また、実行時間を計測し、比較しなさい。

選択法、挿入法、マージソート、クイックソートは各々共通の部分のプログラムを持つので、その共通の部分は sort.c として記述しておく。

基本(sort.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAXN 1000000
int A[MAXN],n,m;
inputdata()
{
   int i;
    /*  乱数系列の初期化  */
    srand((unsigned) time(NULL));
   printf("テ゛ータ数を入力してくた゛さい¥n");
   /*データの格納*/
   scanf("%d",&n);
    printf("¥n");

    printf("ソート前¥n");
   for(i=1; i<=n; i++){
      A[i]=rand()%100000;
      printf("%d ", A[i]);
      }
   printf("¥n");
   return;
   }
```

```
/*値の入れ替え*/
 swap(int i,int j)
 {
    int temp; /*作業用の変数*/
    temp = A[i];
    A[i] = A[j];
    A[j] = temp;
  }

printdata()
 {
    int i;
    printf("¥n");
  printf("ソート済みテ゛ータ¥n");
    for(i=1; i<=n; i++)
      printf("%d ", A[i]);
  printf("¥n ");

    printf("費やした処理時間は%f 秒 。 ¥n",(double)m/CLOCKS_PER_SEC);
  return;
  }
```

解説

A[i]=rand()%100000;	･100000 以下の乱数を発生させ、A[i]に格納
swap	･第一引数と第二引数の格納場所を交換
srand((unsigned) time(NULL));	･実行時間を元にして、発生させる乱数を決定
(double)m/CLOCKS_PER_SEC)	･m は実行に費やした時間。ただし、秒単位ではないので、CLOCKS_PER_SEC で割る。(CLOCKS_PER_SEC で割ると､秒単位になる)

この sort.c をインクルード必要な部分は参照できるので、ソースコードが少なくてすむ。

選択法

```
#include <stdio.h>
#include <time.h>
#include "sort.c"

main(){
   inputdata();
   selectionsort(1,n);
   printdata();
}
 /*選択法のアルゴリズム*/
selectionsort(int p,int q){
   int i, j, cmin;
   clock_t start, end;
   start = clock();
   for(j=p; j<=q; j++){
      cmin=j;
      for(i=j+1; i<=q; i++)
         if(A[cmin]>A[i]) cmin=i;
      swap(j,cmin);
   }
   end =clock();
   m = end -start;
}
```

cmin は最小値の配列での格納場所を示している。最小値を仮に配列の先頭に格納されている値と決め、そこから配列要素をインクリメントし、その時の配列の値と比較。最小値より、インクリメントされた配列要素の格納値が小さいのなら、交換する。…(1)

```
if(A[cmin]>A[i]) cmin=i;
swap(j,cmin);
```

(1) の作業が配列の最後の格納値まで終了したのなら、最小値は配列の一番先頭に存在するはずである。
次に、最小値の格納場所(上のプログラムでは、初めは cmin=j=p=1)をインクリメントさせる。…(2)
(2) の作業が配列の最後まできたら終了である。

挿入法

```
#include <stdio.h>
#include <time.h>
#include "sort.c"

main() {
   inputdata();
   insertionsort(1,n);
   printdata();
}
/*挿入法のアルゴリズム*/
insertionsort (int p, int q){
   int i, j, c;
   clock_t start, end;
   start = clock();
   for(j=p+1; j<=q; j++){
      c=A[j]; i=j;
      while(i>p && A[i-1]>c){
         A[i]=A[i-1]; i=i-1;
      }
      A[i]=c;
}
   end =clock();
   m = end -start;
}
```

ソートが終わっているものとみなし、終わっている要素と新しい要素を比較して、新しい要素が小さいならば、交換する。

```
while(i>p && A[i-1]>c){
A[i]=A[i-1]; i=i-1;
}
```

あたらしい値の方が大きいならば、そのままソートされている部分+1 の配列番号に格納される。

```
A[i]=c;
```

この作業を for 文で条件をインクリメントさせ、配列の最後まで実行する。

```
for(j=p+1; j<=q; j++)
```

マージソート

```
#include <stdio.h>
#include <time.h>
#include "sort.c"

int B[MAXN],n;

main(){
   inputdata();
   clock_t start, end;
   start = clock();
   Mergesort(1,n);
   end =clock();
   m = end -start;
   printdata();
}
/*マージソートのアルゴリズム*/
Mergesort(int p, int r) {
   int q;
   if(p<r){
/*二分割する為に中央の値を決定*/
      q=(p+r)/2;
/*列(前半)の要素が 2 以上ならば、更に分割する*/
      Mergesort(p,q);
/*列(後半)の要素が 2 以上ならば、更に分割する*/
      Mergesort(q+1,r);
/*全て分割し終わったならば合わせていく*/
      merge(p,q,r);
   }
}
/*統合のアルゴリズム*/
merge(int p, int q, int r) {
   int i, j, k;
   i=p; j=q+1;
   /*ソート部分*/
   for(k=p;k<=r;k++){
      if((j>r) || ((i<=q)&&(A[i]<=A[j]))){
         B[k]=A[i]; i=i+1;
      }
      else{
```

```
        B[k]=A[j];j=j+1;
      }
    }
    for(k=p; k<=r; k++) A[k]=B[k];
}
```

関数：Mergesort

生成された配列を 2 分割していく。最終的に一つの数字ずつに分割されるまで実行される。最後まで分割されたら、関数 merge を呼び出し、統合を行う。

関数：merge

分割された数列の先頭に格納されている値同士を比較。小さい方を B[k]の配列に先頭から格納していく。格納された値の数列の方は、配列要素をインクリメントさせる。

クイックソート

```
#include <stdio.h>
#include <time.h>
#include "sort.c"

main() {
   inputdata();
   A[0] = -999999999;
   clock_t start, end;
   start = clock();
   quicksort(1,n);
   end =clock();
   m = end -start;
   printdata();
}

int find_median3(int left, int right) {
   int center, p;
/*基準値となる値の決定*/
   center = (left+right)/2;
   if(A[left]>A[center]) swap(left,center);
   if(A[left]>A[right]) swap(left,right);
   if(A[center]>A[right]) swap(center,right);
   p=A[center];
/*値の交換*/
   swap(center,right);
   return(p);
}

int partition(int left, int right, int q){
   int i, j;
   i=left-1; j=right;
   do{
      do i=i+1; while(A[i]<q);
      do j=j-1; while(A[j]>q);
      if(i<j) swap(i,j);
   }while(i<j);
   swap(i,right);
   return(i);
}
```

```
/*クイックソートのアルゴリズム*/
quicksort(left,right)
int left, right;
{
   int cutoff,pivot,i,j;
   cutoff = 10;
   if ( (right-left) < cutoff) insertionsort(left, right);
   else{
      pivot = find_median3(left,right); /*基準値を算出*/
      i = partition(left,right,pivot); /*基準値を中心に数値の並ひ゛替え*/
      quicksort(left,i-1); /*基準値の右側の配列にクイックソートを適用*/
      quicksort(i+1,right); /*基準値の左側の配列にクイックソートを適用*/
   }
}
/*挿入ソート*/
insertionsort (int p, int q){
   int i, j, c;
   for(j=p+1; j<=q; j++){
      c=A[j]; i=j;
      while(i>p && A[i-1]>c){
         A[i]=A[i-1]; i=i-1;
      }
      A[i]=c;
   }
}
```

ソートすべき数値の集合の中から一つの基準値を取りだし，その数よりも小さい集 合と大きい集合との２つに分け, 基準値よりも左の列と右の列に対して同じ操作を繰り返す。一般に最も早いソートである。

計測時間比較

　選択法と挿入法、マージソートとクイックソートに分けて比較する。理由は、後者の方が断然に実行時間が早すぎて、同じデータ数では計測できなかったからである。

選択法と挿入法

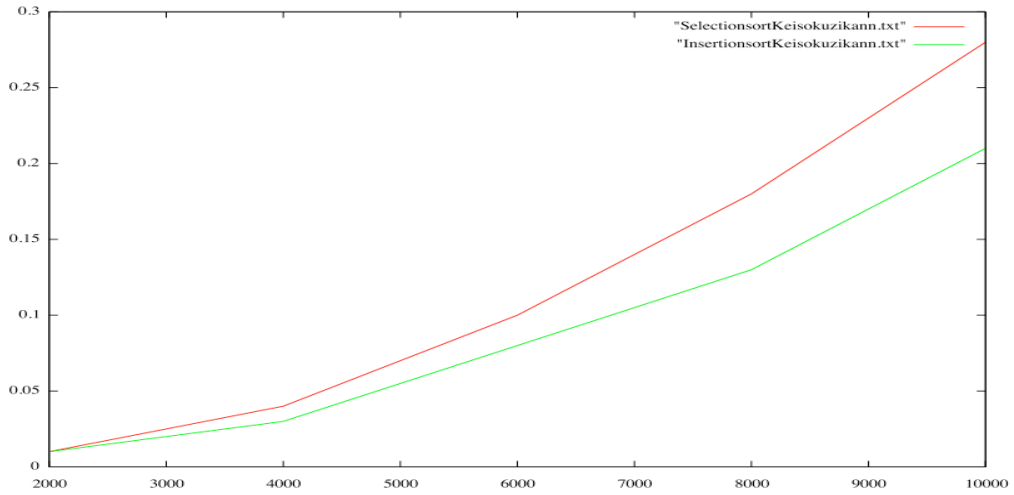


図１：選択法と挿入法の比較
赤：選択法　　緑：挿入法

マージソートとクイックソート

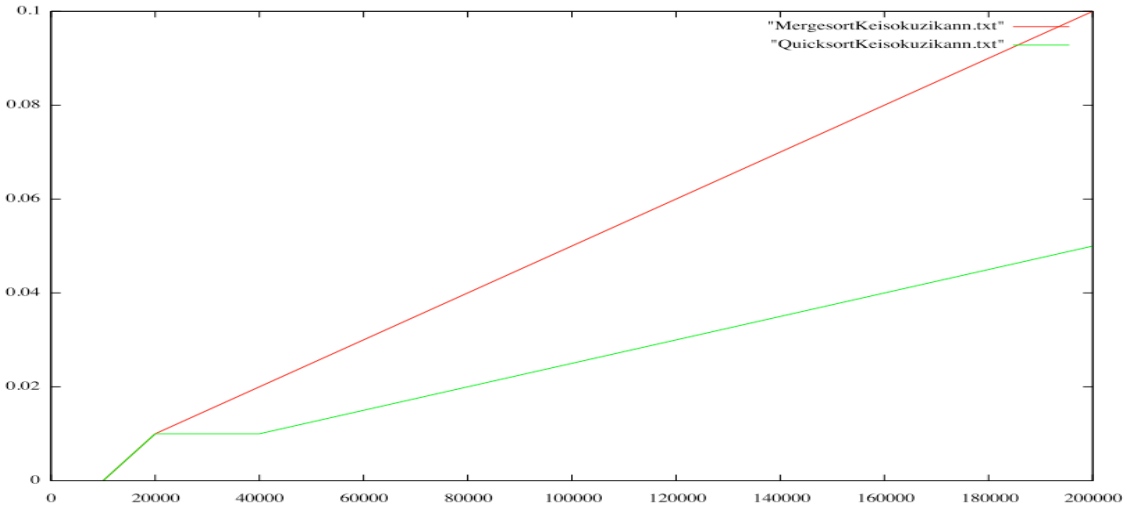
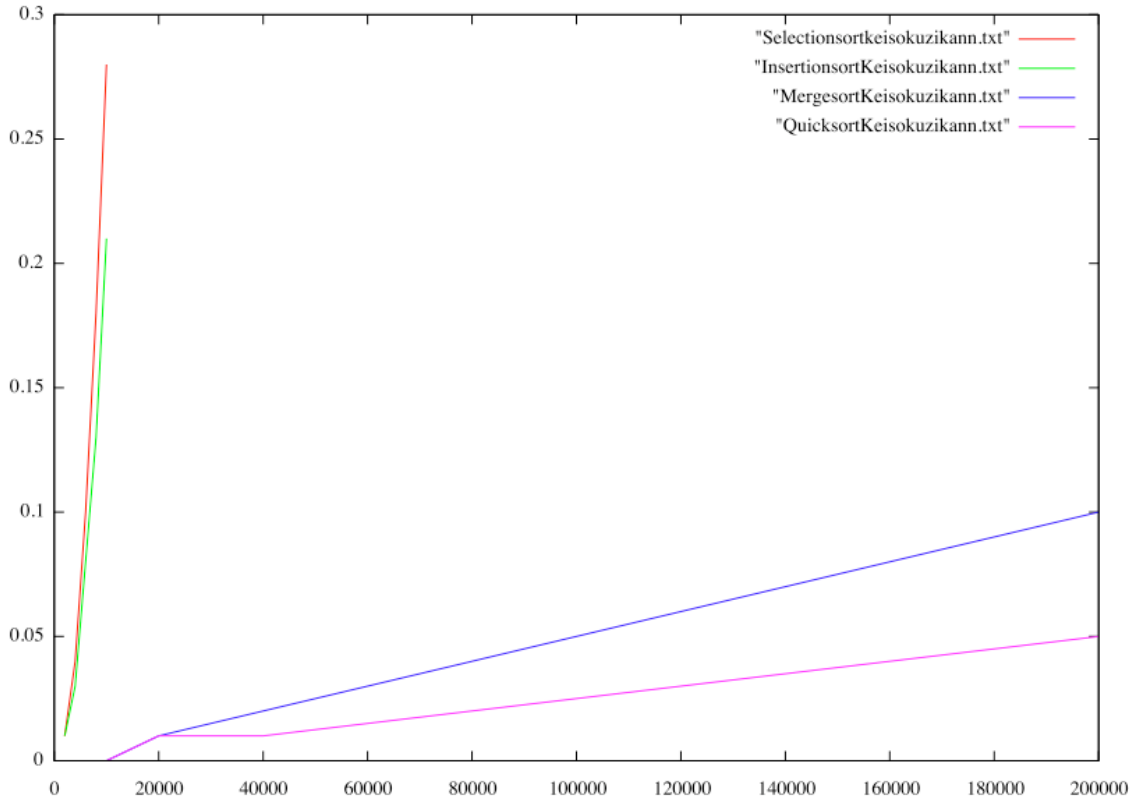


図２：マージソートとクイックソートの比較
赤：マージソート　　緑：がクイックソート

図 1,2 を見ると、マージソートとクイックソートが選択法、挿入法よりも格段に早いことが分かる。また、クイックソートが 4 つのソート法の中で一番早いことが分かる。

まとめ



赤：選択法　　緑：挿入法　　青：マージソート　　紫：クイックソート

参考文献

クイックソートの理論
http://www.klic.org/software/klic/lang/node84.html