データ構造と
基本アルゴリズム

Report3

提出日：12 月 17 日（月）

所属：工学部情報工学科

学籍番号：135713B

氏名：天願 寛之

選択法、挿入法、マージソート、クイックソートをプログラム化し処理時間を計測せよ。また計測結果を比較検討せよ。比較の際にはグラフ等を活用し効果的に行うこと。

1.選択法(データ数 任意)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define MAX 100000
int main(){
  int arrey[MAX],t = MAX;
  inputdata(t,arrey);
  struct timeval start_timeval, end_timeval;
  double sec_timeofday;
  gettimeofday( &start_timeval, NULL );
  SelectionSort(t,arrey);
  gettimeofday( &end_timeval, NULL );
  sec_timeofday = (end_timeval.tv_sec - start_timeval.tv_sec) + (end_timeval.tv_usec - start_timeval.tv_usec)
/ 1000000.0;
  printdata(t,arrey);
  printf("処理時間 O(%f)",sec_timeofday);
}

/*データの入力 ランダム関数を用いてデータの入力*/
int inputdata(int t,int arrey[]){
  int i;
  srand((unsigned)time(NULL));
  printf("適当に並べられた数列------------¥n");
  for(i=0; i<=t; i++){
    arrey[i] = (rand()%100+1);
    printf("arrey[%d]=%d¥t",i,arrey[i]);
  }
  return(0);
}

/*選択法*/
int SelectionSort(int t,int arrey[]){
  int p,q,x,y,min;
  for(p=0; p<=t-1; p++){
    min=p; /*比較する要素*/
    x=arrey[min]; /*比較する要素の値を覚えておく*/
    for(q=p+1; q<=t; q++){
      if(arrey[min]>arrey[q]){ /*条件:比較する要素より小さい値を持つ要素が存在する*/
            arrey[min]=arrey[q]; /*ループの結果、比較した中で最も小さい値を発見する*/
            y=q; /*比較した中で最も小さい値を持つ要素を覚えておく*/
      }
    }
      /*比較する要素と比較した中で最も小さな値を持つ要素を交換*/
    arrey[p]=arrey[min];
    arrey[y]=x;
  }
  return(0);
}

/*データの出力*/
int printdata(int t,int arrey[]){
  int g;
  printf("¥n 選択法の結果--------------------¥n");
  for(g=0; g<=t; g++)
    printf("arrey[%d]=%d¥t",g,arrey[g]);
  return(0);
}
```

2.挿入法(データ数任意)

```
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>
#define MAX 100000

int main(){
  int arrey[MAX],t = MAX;
  struct timeval start_timeval, end_timeval;
  double sec_timeofday;
  inputdata(t,arrey);
  gettimeofday( &start_timeval, NULL );
  InsertionSort(t,arrey);
  gettimeofday( &end_timeval, NULL );
  sec_timeofday = (end_timeval.tv_sec - start_timeval.tv_sec) + (end_timeval.tv_usec - start_timeval.tv_usec)
/ 1000000.0;
  printdata(t,arrey);
  printf("計算時間量 O(%f)",sec_timeofday);
}

/*データの入力 ランダム関数を用いてデータの入力*/
int inputdata(int t,int arrey[]){
  int i;
  srand((unsigned)time(NULL));
  printf("適当に並べられた数列------------¥n");
  for(i=0; i<=t; i++){
    arrey[i] = (rand()%100+1);
    printf("arrey[%d]=%d¥t",i,arrey[i]);
  }
  return(0);
}

/*挿入法*/
int InsertionSort(int t,int arrey[]){
  int p,q,min;
  for(p=1; p<=t; p++){
    min=p; /*比較する要素*/
    q=arrey[min]; /*比較する要素の値を覚えておく*/
    while(min>=1 && arrey[min-1]>q){ /*条件:比較する要素が 1 以上かつ、1 つ前の要素の値が比較する要素の値より大きい
*/
            arrey[min]=arrey[min-1]; min=min-1; /*条件を満たす間、比較された要素の値を 1 つ繰り上げていく*/
      }
    arrey[min]=q; /*ループを抜けたら繰り下げられた要素に比較する要素の値を代入する*/
  }
  return(0);
}

/*データの出力*/
int printdata(int t,int arrey[]){
  int g;
  printf("¥n 挿入法の結果--------------------¥n");
  for(g=0; g<=t; g++)
    printf("arrey[%d]=%d¥t",g,arrey[g]);
  return(0);
}
```

3.マージソート(データ数任意)

```
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>
#define MAX 100000

int main(){
  int arrey[MAX],t = MAX;
  struct timeval start_timeval, end_timeval;
  double sec_timeofday;
  inputdata(t,arrey);
  gettimeofday( &start_timeval, NULL );
  MergeSort(0,t,arrey);
  gettimeofday( &end_timeval, NULL );
  sec_timeofday = (end_timeval.tv_sec - start_timeval.tv_sec) + (end_timeval.tv_usec - start_timeval.tv_usec)
/ 1000000.0;
  printdata(t,arrey);
  printf("計算時間量 O(%f)¥n",sec_timeofday);
}

/*データの入力 ランダム関数を用いてデータの入力*/
int inputdata(int t,int arrey[]){
  int i;
  srand((unsigned)time(NULL));
  printf("適当に並べられた数列------------¥n");
  for(i=0; i<=t; i++){
    arrey[i] = (rand()%100+1);
    printf("arrey[%d]=%d¥t",i,arrey[i]);
  }
  return(0);
}

/*マージソート*/
int MergeSort(int left, int right, int arrey[]){
  int mid, x, y, z, a[MAX];

  if(left<right){
    mid = (left + right) / 2; /*2 分割する位置*/
    MergeSort(left,mid,arrey); /*分割した左部分を再帰呼び出し*/
    MergeSort(mid+1,right,arrey); /*分割した右部分を再帰呼び出し*/

  x = left; y = mid+1; /*右部分と左部分、各々の左端の要素*/
  for(z=left; z<=right; z++){
      /*右部分と左部分各々を左から比較していき小さい方を仮の配列に代入*/
    if((y>right) || ((x<=mid)&&(arrey[x]<=arrey[y]))){
      a[z]=arrey[x]; x=x+1;
    }else{
      a[z]=arrey[y]; y=y+1;
    }
  }
  for(z=left; z<=right; z++) arrey[z]=a[z]; /*整えた配列をコピー*/
  }
  return(0);
}

/*データの出力*/
int printdata(int t,int arrey[]){
  int g;
  printf("¥n マージソートの結果--------------------¥n");
  for(g=0; g<=t; g++)
    printf("arrey[%d]=%d¥t",g,arrey[g]);
  return(0);
}
```

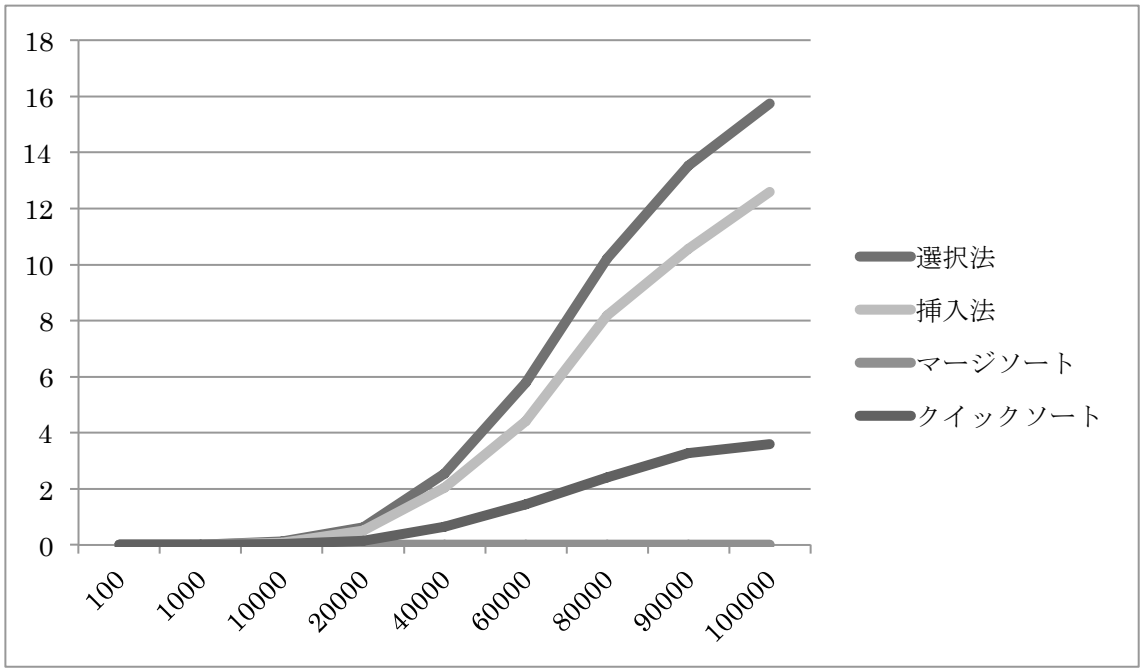
4.クイックソート(データ数任意)

```
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>
#define MAX 100000

int arrey[MAX],t = MAX;
int main(){
  struct timeval start_timeval, end_timeval;
  double sec_timeofday;
  inputdata();
  arrey[0]=-99999999;
  gettimeofday( &start_timeval, NULL );
  QuickSort(0,t);
  gettimeofday( &end_timeval, NULL );
  sec_timeofday = (end_timeval.tv_sec - start_timeval.tv_sec) + (end_timeval.tv_usec - start_timeval.tv_usec)
/ 1000000.0;
  printdata();
  printf("計算時間量 O(%f)",sec_timeofday);
}

/*データの入力 ランダム関数を用いてデータの入力*/
int inputdata(){
  int i;
  srand((unsigned)time(NULL));
  printf("適当に並べられた数列------------¥n");
  for(i=0; i<=t; i++){
    arrey[i] = (rand()%100+1);
    printf("arrey[%d]=%d¥t",i,arrey[i]);
  }
  return(0);
}

/*左端と右端、センターの中からピボットとなる要素を選出*/
int find_med(int left, int right){
  int center,p;
  center = (left+right)/2;
    /*左端と右端、センターの要素の値を比較し、小さい順に左から並べ直す*/
  if(arrey[left]>arrey[center]) swap(left,center);
  if(arrey[left]>arrey[right]) swap(left,right);
  if(arrey[center]>arrey[right]) swap(center,right);
    /*左端と右端、センターの中でピボットの値を p に代入*/
  p=arrey[center];
    /*ピボットの値と右端の値を交換*/
  swap(center,right);
  return(p);
}

/*ピボットを基準として 2 つの系列に分割*/
int partition(int left, int right, int q){
  int i,j;
  i=left-1; j=right;
    /*do 関数*/
  do{
    do i=i+1; while(arrey[i]<q); /*左の系列の中で左から右へ移動しながらピボットより小さくなる最初の位置で止まる*/
    do j=j-1; while(arrey[j]>q); /*右の系列の中で右から左へ移動しながらピボットより大きくなる最初の位置で止まる*/
    if(i<j)swap(i,j); /*ピボットより小さい値と大きい値を交換*/
  }while(i<j); /*ピボットより小さい値、大きい値からなる 2 つの系列を生成*/
  swap(i,right); /*-99999999 とピボットを交換*/
  return(i);
}

int QuickSort(int left, int right){
  int cutoff,pivot,i,j;
```

```
  cutoff = 10;
    /*要素数が 10 未満のときは挿入法*/
  if((right-left)<cutoff) InsertionSort(left,right);
    /*要素数が 10 以上のときはクイックソート*/
  else{
      /*ピボットとなる値を見つけて pivot に代入*/
    pivot = find_med(left,right);
      /*ピボットの値を i に代入*/
    i=partition(left,right,pivot);
    QuickSort(left,i-1); /*系列の左部分を再帰呼び出し*/
    QuickSort(i+1,right); /*系列の右部分を再帰呼び出し*/
  }
  return(0);
}

/*挿入法*/
int InsertionSort(int left, int right){
  int p,q,min;
  for(p=1; p<=right; p++){
    min=p;
    q=arrey[min];
    while(min>=1 && arrey[min-1]>q){
      arrey[min]=arrey[min-1]; min=min-1;
    }
    arrey[min]=q;
  }
  return(0);
}

/*スワップ関数を定義*/
int swap(int x, int y){
  int z;
  z=arrey[x];
  arrey[x]=arrey[y];
  arrey[y]=z;
  return(0);
}

/*データの出力*/
int printdata(){
  int g;
  printf("¥n クイックソートの結果----------¥n");
  for(g=0; g<=t; g++)
    printf("arrey[%d]=%d¥t",g,arrey[g]);
  return(0);
}
```

5.処理時間の計測、比較、検討

データ数は各々100,1000,10000,100000 で実行



5-a. 選択法について

4 つのソート法の中で平均時間は一番多い。小さい順に選出して並べていくので比較時間が多いのである。グラフ上ではデータ数が多くなると最もあがり幅が大きいようだ。

5-b. 挿入法について

選択法に並ぶ処理時間だが、部分的に整列していれば比較時間は選択法よりもかなり減少ことがわかる。一般的に選択法よりも速い。

5-c. マージソートについて

全体的に最も速く、安定的な処理速度を見る事が出来る。データ数が大きくなっても処理時間が比較的小さいところが魅力的である。

5-d. クイックソート

ピボットが最適かそうじゃないかで処理時間が少し変動するが、比較的速い処理速度である。その名に恥じぬ速さでナス。