アルゴリズム論 レポート3

各種ソートアルゴリズムの計算時間の違いを理解する

276156 根本 貴大

2017

1. ソースコード

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define NUM 10000

#define swap(type,x,y) do {type t=x; x=y; y=t;} while(0)

void bubble(int a[], int n);

void selsort(int a[], int n);

void insertion(int a[], int n);

void quick(int a[], int left, int right);

void downheap(int a[], int left, int right);

void heap(int a[], int n);

int main(void) {

int i, j, x0[NUM], x1[NUM], x2[NUM], x3[NUM], x4[NUM];

double temp, dt0 = 0.0, dt1 = 0.0, dt2 = 0.0, dt3 = 0.0, dt4 = 0.0;

time\_t start, end;

srand(time(NULL));

for (i = 0; i < NUM; i++) {

temp = (double)rand() / (double)RAND\_MAX;

x0[i] = x1[i] = x2[i] = x3[i] = x4[i] = (int)(temp\*1000.0);

}

start = clock();

bubble(x0, NUM);

end = clock();

dt0 += (double)(end - start) / CLOCKS\_PER\_SEC;

start = clock();

selsort(x1, NUM);

end = clock();

dt1 += (double)(end - start) / CLOCKS\_PER\_SEC;

start = clock();

insertion(x2, NUM);

end = clock();

dt2 += (double)(end - start) / CLOCKS\_PER\_SEC;

start = clock();

quick(x3,0,NUM-1);

end = clock();

dt3 += (double)(end - start) / CLOCKS\_PER\_SEC;

start = clock();

heap(x4, NUM);

end = clock();

dt4 += (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Running time bsort=%lf (sec), ssort=%lf (sec),isort = %lf(sec)\n", dt0, dt1, dt2);

printf("Running time qsort=%lf (sec), hsort=%lf (sec) \n", dt3, dt4);

return 0;

}void bubble(int a[], int n){

for (int i = 0; i<n - 1; i++) {

for (int j = n - 1; j>i; j--) {

if (a[j - 1]>a[j])

swap(int, a[j - 1], a[j]);

}

}

}

void selsort(int a[], int n){

int min;

for (int i = 0; i <= n - 1; i++) {

min = i;

for (int j = i + 1; j <= n - 1; j++) {

if (a[j]<a[min]) {

min = j;

}

swap(int, a[i], a[min]);

}

}

}

void insertion(int a[], int n){

int j,tmp;

for (int i = 1; i <= n - 1; i++) {

tmp = a[i];

j = i;

while ((a[j - 1]>tmp) && (j>0)){

a[j] = a[j - 1];

j--;

}

a[j] = tmp;

}

}

void quick(int a[], int left, int right){

int pl = left,pr = right,x = a[(pl + pr) / 2];

do {

while (a[pl]<x)

pl++;

while (a[pr]>x)

pr--;

if (pl <= pr) {

swap(int, a[pl], a[pr]);

pl++;

pr--;

}

} while (pl <= pr);

if (left<pr)

quick(a, left, pr);

if (pl<right)

quick(a, pl, right);

}

void downheap(int a[], int left, int right){

int temp = a[left],child,parent;

for (parent = left; parent < (right + 1) / 2; parent = child) {

int cl = parent \* 2 + 1;

int cr = cl + 1;

if (cr <= right && a[cr]>a[cl])

child = cr;

else

child = cl;

if (temp >= a[child])

break;

a[parent] = a[child];

}

a[parent] = temp;

}

void heap(int a[], int n) {

for (int i = (n - 1) / 2; i >= 0; i--)

downheap(a, i, n - 1);

for (int i = n - 1; i > 0; i--) {

swap(int, a[0], a[i]);

downheap(a, 0, i - 1);

}

1. 入力データ数を変化させて計算時間を確認

各アルゴリズムの計算時間は以下の表1に示す。

表1.各アルゴリズムの計算時間

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | データの個数 | | | | |
| 10000 | 20000 | 30000 | 40000 | 50000 |
| アルゴリズム | bsort | 0.300000 | 1.311000 | 2.930000 | 5.350000 | 8.384000 |
| ssort | 0.299000 | 1.071000 | 2.315000 | 3.942000 | 6.116000 |
| isort | 0.078000 | 0.321000 | 0.713000 | 1.328000 | 1.983000 |
| qsort | 0.001000 | 0.003000 | 0.004000 | 0.005000 | 0.006000 |
| hsort | 0.003000 | 0.005000 | 0.008000 | 0.012000 | 0.014000 |

表1の計算時間を図1に記載した。クイックソートとヒープソートの値は近似しているため重なっている。

図1.計算量グラフ

1. 各種ソートの計算量とオーダを考察

プログラムを作成し、実行した結果

バブルソート > 単純選択法 > 挿入法 > ヒープソート > クイックソート

の順に計算時間が長いことがわかった。

バブルソート、単純選択法、挿入法のどれもが、配列に格納された値を一つ一つ比較しているため処理計算量が増加し計算時間が増加してしまう。

クイックは他のソートと比べて計算時間が圧倒的に短い。理由としてはバブルソートと比べると、バブルソートでは隣同士の値を毎回比較するためといえる。しかしクイックソートの場合は中心の値を基準値とし、基準値より大きい値は右に、基準値より小さい値は左に、というようにソートしていくので計算量が圧倒的に違う。