アルゴリズム論第三回課題

|  |  |
| --- | --- |
| 提出日 | ７月２日 |
| 学生番号 | ２５６１２９ |
| 氏　名 | 杉浦　圭 |

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define NUM 50000 /\* ソートするデータの個数\*/

#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 heapsort(int a[], int n);

int main(void){

int i;

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

double temp;

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

time\_t start, end;

srand((unsigned)time(NULL));

/\* 乱数発生　\*/

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);

}

/\* ソート時間の計測　\*/

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();

heapsort(x4, NUM);

end = clock();

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

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

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

return(0);

}

void bubble(int a[], int n)

{

int i, j;

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

for (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 i, j, min;

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

min = i;

for (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 i, j, tmp;

for (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;

int pr = right;

int 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];

int child;

int parent;

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

int cl = parent \* 2 + 1; /\* left child \*/

int cr = cl + 1; /\* right child \*/

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 heapsort(int a[], int n){

int i;

for (i = (n - 1) / 2; i >= 0; i--)　downheap(a, i, n - 1);

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

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

downheap(a, 0, i - 1);

}

}

Visual Studio 2013 での計算結果

※heapとquickの値が近いため、グラフが重なってしっている

**考察**

プログラムを作成し、Visual Studio 2013で実行したところ、

quick<heap<insert<sel<bubble

(クイック<ヒープ<挿入法<選択法<交換法)

の順に処理計算時間が短いことがわかった。

挿入法、選択法、交換法のどれもが、各配列に格納された値を一つ一つ比較しているために、処理計算量が増加し、時間を要してしまう。

クイックがほかのソートと比べて計算量は短いのは、例えばbubbleと比べると、bubbleでは隣同士の値を毎回比較するが、クイックだと真ん中の値を基準値として、大きい値は右に。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。

また、上図ではすべてのオーダーが、平均的な場合のオーダーに近くなっている。