Quicksort Performance Comparison Report

Yu Sheng (Julia)

2018/2/14

TABLE 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data size(million) | Bentley’s Quicksort | Hybrid Quicksort | Library sort() | Dual-Pivot Quicksort |
| 1 | 161.226167 | 114.888819 | 89.990812 | 89.948203 |
| 5 | 774.942859 | 486.068734 | 372.675319 | 399.60713 |
| 10 | 1523.753995 | 971.179426 | 736.247023 | 811.618149 |
| 15 | 2297.428825 | 1496.80694 | 1133.30197 | 1250.672085 |
| 20 | 3127.175161 | 2032.43479 | 1527.70198 | 1675.102585 |
| 25 | 3947.953358 | 2570.157847 | 1951.18456 | 2130.106419 |
| 30 | 4767.121874 | 3118.549618 | 2334.77568 | 2579.668775 |
| 35 | 5624.399606 | 3727.982055 | 2741.71509 | 3055.770454 |
| 40 | 6416.887437 | 4496.762704 | 3191.39751 | 3509.865708 |
| 45 | 7218.298498 | 4763.729007 | 3590.90579 | 3929.155111 |
| 50 | 8133.660851 | 5411.312281 | 4018.25206 | 4424.769619 |

Average Case Performance Plot

Fig 1

I use Java for implementation the four different quicksort algorithms and the unit for recording time is milliseconds. The library sort() functions I used in Java is Array.sort().

In average case where we shuffle the array randomly and then sort the array, we find four curves are in a shape similar to nlogn curve as shown in Fig 1. The sorting time increases as the size of dataset increases. Among four, the time duration curve of Bentley’s Quicksort rises most rapidly. It costs Bentley’s Quicksort more than 8 seconds to sort a 50 million dataset, which is almost twice the time taken by library sort() as stated in TABLE 1. Thus Bentley’s Quicksort is the slowest sort. In average case, the time complexity of quicksort is O(nlogn) while insertion sort is O(n^2). However, in the best case where array is already sorted, the time complexity of quicksort is still O(nlogn) but insertion sort is O(n)! Therefore, Hybrid Quicksort combines the advantage of quicksort in unsorted dataset and the advantage of insertion sort in near sorted dataset thus costs less time to sort compare to a pure quicksort. And in the advance of hybrid quicksort, Dual-pivot quicksort adds divisions to the array and do several sorting at the same time thus becomes more efficient. Therefore, we can see in Fig 1, time duration curve of Bentley’s Quicksort is greatly above all other three sort algorithms; Hybrid Quicksort curve is above Library sort and Dual-Pivot Quicksort curve. In Java, the library sorting algorithm Array.sort() is a Dual-Pivot Quicksort. Thus we can see the green line represents the dual pivot sort and yellow line represents the library sort are very close to each other. The subtle difference between them may cause by the different shuffle of array each time.

In conclusion, Bentley’s Quicksort is the slowest sorting algorithm among the four; Hybrid Quicksort is in the middle; Library sort and Dual-Pivot Quicksort are the fastest.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CODE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Random;

import java.util.Arrays;

public class sort {

public static void main(String[] args) {

double d1,d2,d3,d4;

long start;

int size=1000000;

int [] A= new int[size];

Random rand=new Random();

for(int i=0;i<size;i++){

A[i]=i;

}//initialize array

for(int i=0;i<size;i++){

int j=rand.nextInt(size-i)+i;

int swap = A[i];

A[i]=A[j];

A[j]=swap;

} //shuffle

start =System.nanoTime();

quicksort(A,0,size-1);

d1= (System.nanoTime()-start)/1000000.0;

start =System.nanoTime();

hybridsort(A,0,size-1);

d2= (System.nanoTime()-start)/1000000.0;

start =System.nanoTime();

librarysort(A,0,size-1);

d3= (System.nanoTime()-start)/1000000.0;

start =System.nanoTime();

dualPivotQuicksort(A,0,size-1,3);

d4= (System.nanoTime()-start)/1000000.0;

System.out.println(d4);

}//main

public static void swap(int[] A, int i, int j) {

int temp = A[i];

A[i] = A[j];

A[j] = temp;

}

public static void quicksort(int[] A,int left, int right){

Random rand=new Random();

if(right-left<=0)

return;

int swap=A[left];

int random =rand.nextInt(right-left)+left;

A[left]=A[random];

A[random]=swap;

int m=left;

for(int i=left+1;i<=right;i++){

if(A[i]<A[left]){

m++;

swap(A,i,m);

}//if

}//for

swap(A,left,m);

quicksort(A,left,m-1);

quicksort(A,m+1,right);

}//quicksort

public static void mquicksort(int[] A,int left, int right){

Random rand=new Random();

int cutoff=25;

if(right-left<=cutoff)

return;

int swap=A[left];

int random =rand.nextInt(right-left)+left;

A[left]=A[random];

A[random]=swap;

int m=left;

for(int i=left+1;i<=right;i++){

if(A[i]<A[left]){

m++;

swap(A,i,m);

}//if

}//for

swap(A,left,m);

mquicksort(A,left,m-1);

mquicksort(A,m+1,right);

}//modified quicksort

public static void insertionsort(int[] A,int left,int right){

for(int i=left+1;i<=right;i++){

int key =A[i];

int j=i-1;

while(j>=0&&A[j]>key){

A[j+1]=A[j];

j--;

}

A[j+1]=key;

}

return;

}//insertionsort

public static void hybridsort(int[] A,int left,int right){

mquicksort(A,left,right);

insertionsort(A,left,right);

return;

}//hybridsort

public static void librarysort(int[] A,int left,int right){

Arrays.sort(A);

}//librarysort

public static void dualPivotQuicksort(int[] A,int left,int right,int div){

if(left>=right)

return;

int len = right - left;

if(len<25)

insertionsort(A,left,right);

int part = len / div;

// "medians"

int m1 = left + part;

int m2 = right - part;

if (m1 <= left) {

m1 = left + 1;

}

if (m2 >= right) {

m2 = right - 1;

}

if (A[m1] < A[m2]) {

swap(A, m1, left);

swap(A, m2, right);

}

else {

swap(A, m1, right);

swap(A, m2, left);

}

// pivots

int pivot1 = A[left];

int pivot2 = A[right];

// pointers

int less = left + 1;

int great = right - 1;

// sorting

for (int k = less; k <= great; k++) {

if (A[k] < pivot1) {

swap(A, k, less++);

}

else if (A[k] > pivot2) {

while (k < great && A[great] > pivot2) {

great--;

}

swap(A, k, great--);

if (A[k] < pivot1) {

swap(A, k, less++);

}

}

}

// swaps

int dist = great - less;

if (dist < 13) {

div++;

}

swap(A, less - 1, left);

swap(A, great + 1, right);

// subarrays

dualPivotQuicksort(A, left, less - 2,div);

dualPivotQuicksort(A, great + 2, right,div);

// equal elements

if (dist > len - 13 && pivot1 != pivot2) {

for (int k = less; k <= great; k++) {

if (A[k] == pivot1) {

swap(A, k, less++);

}

else if (A[k] == pivot2) {

swap(A, k, great--);

if (A[k] == pivot1) {

swap(A, k, less++);

}

}

}

}

// subarray

if (pivot1 < pivot2) {

dualPivotQuicksort(A, less, great, div);

}

}

}//class sort