A New Sorting Algorithm

Kiran Kumar Sundararajan             Soubhik Chakraborty*
Population System Analysis-ESD             Department of Statistics
Infotech Enterprises Ltd             Marwari College
Infocity                                T. M. Bhagalpur University
Hyderabad-500033                         Bhagalpur – 812007
India                                   India
Email:kiranks@infotechsw.com            email:soubhikc@yahoo.co.in
                                           *communicating author

Abstract: We are introducing a new sorting algorithm.

Key words: sorting, temporary array, pivot, interchange

Section I is the introduction in which the new sorting algorithm is proposed. Section II provides a pseudo code which can be programmed easily. Section III gives some empirical results. Section IV is the conclusion and suggestions for future work.

Section I

Introduction:

We introduce the new sorting algorithm as follows:

step 1: Initialize the first element of the array as a pivot(key) element

step 2: Starting from the second element, compare it to the pivot element.
step 2.1: if pivot < element then place the element in the last unfilled position of a temporary array (of same size as the original one)

step 2.2: if pivot > element then place the element in the first unfilled position of the temporary array

Step 3: repeat step 2 till last element of the array.

step 4: finally place the pivot element in the blank position of the temporary array

(remark: the blank position is created because one element of the original array was taken out as pivot)

Step 5: split the array into two, based on the pivot element's position.

step 6: repeat steps 1-5 till the array is sorted fully.

Section II

Here is the pseudocode:

My_sort(int numbers[], int b[], int l, int r)
{
// pivot=first element of array, low=l, up=r , numbers[] is the original
array, b[] is temp array.

// l is left index and r is right index

// My logic starts here.

for (i=l+1; i<=r; i++)
{
    if (numbers[i] <= pivot)
    {
        b[low] = numbers[i];
        low++;
    }
    else
    {
        b[up]=numbers[i];
        up--;
    }
}

b[low]=pivot;
for (i = l; i <= r; i++)
    numbers[i]=b[i];

if (l < up-1)
    My_sort(b,numbers, l, up-1);

if(up+1 < r)
    My_sort(b,numbers, up+1,r);

} 

Section III

We are producing below some interesting empirical results. For n<=1,5,00,000 our new sorting algorithm performed even better than heapsort!(see table 1). However, it slows down for higher n(not shown in the table). The codes we used are given in the appendix.

Table 1

<table>
<thead>
<tr>
<th>n</th>
<th>heapsort time</th>
<th>new sorting algorithm time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>100000</td>
<td>0.18</td>
<td>0.15</td>
</tr>
<tr>
<td>500000</td>
<td>1.231</td>
<td>1.041</td>
</tr>
<tr>
<td>1000000</td>
<td>2.834</td>
<td>2.65</td>
</tr>
<tr>
<td>1500000</td>
<td>4.747</td>
<td>4.616</td>
</tr>
</tbody>
</table>
Conclusion and suggestions for future work:

We conclude that for n<=15 lakhs, it is worth recommending our algorithm to sort n items but perhaps not for higher n unless this algorithm is improved for the purpose. Accordingly, our investigations are on for the higher n case. It may be of interest to find out how this algorithm behaves if the pivot is selected randomly (use of the concept of “randomized algorithm” is proposed; the book in ref[3] is an excellent source of reference on the topic). Future work also includes doing a complexity analysis of (the improved version of) the algorithm and studying how this new sorting algorithm behaves for inputs coming from specific probability distributions.

UPSHOT: Our algorithm works on the “divide and conquer” strategy similar to Quicksort but the use of an auxiliary array results in avoiding interchanges of elements, thereby sacrificing space.

Acknowledgement: We thank the editor Dr. R. G. Graf for helping us bring the paper to the present form.

References:


C PROGRAM FOR HEAP SORT

#include <stdlib.h>
#include <stdio.h>
#include <sys/timeb.h>
#include <time.h>

#define n 100000

void heap(int a[], int array_size);
void siftDown(int a[], int root, int bottom);

int a[n];

int main()
{
    int i;
    clock_t start, finish;
    double duration;

    for (i = 0; i < n; i++)
        a[i] = rand();

    start = clock();

    heap(a, n);

    finish = clock();

    duration = (double) (finish - start) / CLOCKS_PER_SEC;

    printf("Elapsed time duration is %7.4f \n\n", duration);
}

void heap(int a[], int array_size)
{
    int i, temp;

    for (i = (array_size / 2)-1; i >= 0; i--)
        siftDown(a, i, array_size);
for (i = array_size-1; i >= 1; i--)
{
    temp = a[0];
    a[0] = a[i];
    a[i] = temp;
    siftDown(a, 0, i-1);
}

void siftDown(int a[], int root, int bottom)
{
    int done, maxchild, temp;

done = 0;
while ((root*2 <= bottom) && (!done))
{
    if (root*2 == bottom)
        maxchild = root * 2;
    else if (a[root * 2] > a[root * 2 + 1])
        maxchild = root * 2;
    else
        maxchild = root * 2 + 1;

    if (a[root] < a[maxchild])
    {
        temp = a[root];
        a[root] = a[maxchild];
        a[maxchild] = temp;
        root = maxchild;
    }
    else
        done = 1;
}
}
C PROGRAM FOR NEW SORT

#include <stdlib.h>
#include <stdio.h>
#include <sys/timeb.h>
#include <time.h>

#define n 100000

void MySort(int a[], int array_size);
void My_sort(int a[], int left, int right);

int a[n], b[n];

int main()
{
    int i;
    int

    for (i = 0; i < n; i++)
        a[i] = rand();

    MySort(a, n);

}

void MySort(int a[], int array_size)
{
    clock_t start, finish;
    double duration;

    start = clock();

    My_sort(a, 0, array_size - 1);

    finish = clock();
    duration = (double) (finish-start) / CLOCKS_PER_SEC;
    printf("Elapsed time duration is %10.6f\n", duration);
}

void My_sort(int a[], int l, int r)
{
    
}
System Specifications:

128 MB RAM, Pentium III processor with 550 MHz, 12 GB HDD