A NEW APPROACH TO BUCKET SORT

BURAK TAKMAZ – MURAT AKIN
Institute of Sciences
Galatasaray University
Istanbul
TURKEY

Abstract: - The Bucket Sort has been one of the simplest distribution sorts and the Shell Sort (presented in 1959 by Donald L. Shell) has been one of the oldest sort methods until now. Although we cannot change their complexities, we determined that some improvements on the bucket sort can be made by different implementation mechanisms. In this paper, we mixed the bucket sort and shell sort. This new approach has worked with greater performance than the standard bucket sort algorithm.

Key-Words: - Bucket Sort, Shell Sort, Sorting

1 Introduction
The studies on sorting algorithms can be grouped into four categories: choosing a mathematical model and applying it to the sorting process, parallel processing, algorithm analysis and hybrid algorithm studies. A hybrid mechanism can be applied to bucket sort thanks to its modularity. This modularity can be used for different implementation mechanisms. A similar hybrid approach to the bubble and shell sort was presented in Klerlein & Fullbright[2] paper.

One of the best algorithms which work with great performance on medium size lists is the shell sort. The shell sort average time complexity is \( N^2 \) according to Knuth[3]. On the other hand, the average time complexity can be reduced to \( N^{1.5} \) considering Weiss[4] with a different increment size decision strategy.

This paper does not focus on the on the complexity of the shell sort or bucket sort. Instead, one can develop different implementation methods. In this way, one can obtain faster time results regarding to the classical bucket sort.

What if one uses the bucket sort and sort together? Dividing the shell sort into four steps and combining with bucket sort can lead to better results.

2 Bucket Sort
The term bucket sort is used for different algorithms. Aho, Hopcroft and Ullman used the term bucket sort [5] in their book. The bucket sort that they mention is also known as bin sort. It consists of indexing and counting the number of elements via the indexes.

The bucket sort used in this paper relies on the bucket sort mentioned in Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein[6]. The main idea behind this algorithm is; first to collect the numbers of a list into buckets with same intervals and secondly, sorting the buckets individually. Than the numbers in the buckets are written in the initial list.

In more details, one can define three main steps for bucket sort as below:

Consider an array of numbers in an interval belonging to \([1..n]\).

First step: At the first step, the numbers are grouped into equal sized intervals: \([0, 1/n), [1/n, 2/n), [2/n, 3/n)\) and so on.

Second Step: At the second step, the numbers are sorted using an insertion sort or a bin sort.

Third Step: At the final step, the numbers which have been sorted in the buckets are rewritten into the initial array. Thus, we obtain the initial list with sorted elements.
3 New Approach

Insertion and bin sort have been applied widely to bucket sort at the second step. In this paper, we are going to use the shell sort. The shell sort works with great performance on medium size lists. Additionally, we know that insertion sort works fine in the buckets. Since, the shell sort is a kind of insertion sort. We can foresee that it is going to work efficiently on the medium size buckets.

In [1] Edward Corwin and Antonette Logar mentioned that reordering the steps of the bucket sort can cause to better results. If we call first step as "scatter", second step as "sort" and third step as "gathering". Reordering the steps order from "scatter, sort and gather" to "scatter, gather and sort" can reduce the sorting time significantly. This is due to that, gathering operation puts the values into consecutive locations in the memory. Forasmuch as the elements of an array are in the sequential locations. We are going to see the details of the new approach.

In order to sort the array at the final step, we modified the shell sort, in such a way that the modified version works only for a limited number of elements of the array.

3.1 Classes

Let's consider, that we have an array A[1..m] with n numbers.

We have implemented the algorithm in C++. We have used two classes:

- Class Element : represents an elements of a linked list
- Class Bucket: represents a bucket containing "Element" objects.

3.2 Steps of New Approach

We can divide the new approach into 5 steps.

First step: We create the Bucket and the Element objects.

The number of buckets is found with the following operation:

Number of Buckets = (max of the list – min of the list) / interval (1).

We used the numbers between 0 and 100. And we decided to take the interval as 10. So each time, we have 10 buckets.

Second step: Subsequently, we create n Elements at one single command:

Element* elements = new Element[n];
Element objects hold the value of array elements.

Third step: We fill the buckets with Element objects.

We distribute the numbers into buckets regarding the Element object’s value. We determine bucketIndex, which is the index of bucket where the Element object will be put. The bucket index is found in this way:

bucketIndex = value of Element object / interval (2)

The buckets are composed of Element objects. These Element objects in the buckets are organized as linked lists.

Fourth step: We write back the value each Element objects in the buckets, into the initial array.

Consequently, the elements are in consecutive locations in the memory. Now the initial array is composed of sections where each section elements belong to an interval such as [0, 1/n), [1/n, 2/n), [2/n, 3/n) and so on.

Fifth step: We sort each bucket with the shell sort modified.

For this process we modified the shell sort. The modified shell sort takes two parameters: startIndex and stopIndex. Each time, while sorting the array, the shell sort work only on a determined piece of list regarding to the number of elements in the buckets filled at the second step.

Let’s call A, initial array to be sorted. Pseudo code for modified shell sort is as follows:

```
SHELLSORTMODIFIED
(parameters: int startindex, int stop index);
int temp;
int size = stopIndex – startIndex + 1;
int increment = size / 2;
WHILE increment >0
    FOR i=startIndex+increment TO stopIndex DO
        j = i;
        temp = A[j];
        WHILE (j >=increment) AND (mArray[j-increment]>temp) DO
            j = j- increment;
            A[j] = temp;
        increment /= 2;
```

4 Conclusion

The results listed in Table 1, were generated with a machine which has a Pentium III processor and 256 Mb RAM. The machine’s operating system is Linux(Fedora 4). The interval of the array is between 0 and 100. The numbers are generated randomly. The results obtained are in microseconds.

Bucket sort is an efficient sort for small and medium size lists. We reduced the sort process time by mixing the sort with shell sort and implementing...
some tricks. We have seen that, still some improvements can be made to bucket sort. The overall performance obtained is better than the Regular Bucket Sort(where each bucket is sorted with the Insertion Sort) and of course Bubble Sort. When comparing with shell sort, we can determine also that the performance of the new approach is better when the size of list is greater than 2000 elements.

Implementation is the key in sorting algorithms. In the future, same kind of improvements can be made on other sort algorithms by changing implementation details. Secondly, new “mixed” sort algorithms can be invented and applied for increasing the performance of the systems.

<table>
<thead>
<tr>
<th>List Size</th>
<th>Algorithm Used</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bubble Sort</td>
</tr>
<tr>
<td>500</td>
<td>2325</td>
</tr>
<tr>
<td>1000</td>
<td>9332</td>
</tr>
<tr>
<td>2000</td>
<td>43551</td>
</tr>
<tr>
<td>5000</td>
<td>248617</td>
</tr>
<tr>
<td>7500</td>
<td>562665</td>
</tr>
<tr>
<td>10000</td>
<td>994870</td>
</tr>
<tr>
<td>15000</td>
<td>2 seconds +180220</td>
</tr>
<tr>
<td>50000</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 1. Time comparison between different sort algorithms and the new approach(Results are in microseconds)

References: