Design and Analysis of Algorithm for Merging of two Sorted Sets

Nairit Banerjee
IIT2016505

Kamal Nayan Chaubey
IIT2016510

Ashutosh Vishwakarma
IIT2016017

Mandeep Chakma
IIT2016012

February 2, 2018

Abstract

This document is a final report of our study of merging of two sorted sets. We are given two sorted sets and task is to merge both sets into a final sorted set. This algorithm takes order of *O(n+m)*(where n and m are size of the given sets) time for best ,worst and average cases. We finally arrived at a conclusion that running time for the three different cases differ only by constants.

Keywords: *Merging, Time Complexity, Space Complexity, Algorithm Design, Graphical Analysis*

1 Introduction And Literature Survey

In mathematics and computer science, the process of combining elements of two sorted sets into another set in sorted order is known as merging. Merging is not a new term in this era. Merging is the most important part in the Merge Sort algorithm which is widely used in complex search engine algorithms to stock markets, sorting has an impeccable presence in this modern day era of information technology. In this document we are providing the best suited algorithm to combine two sorted sets. By sorted set we mean that all the elements in the set is arranged in a particular order either decreasing or increasing.
For example, suppose we have two sets namely A and B where,
A=[1,3,5,7,9] and B=[2,4,6,8,10]
after merging will result into set
C=[1,2,3,4,5,6,7,8,9,10]

1.1 Different Cases

There arises three different cases in this problem. Different Cases arises because of the fact that both the given sets may be in any sorted order. Algorithm for all the cases is almost the same.Following are all the three cases:

(a)Case 1:
The first case is that one of the given set is in increasing order and the other one being in decreasing order.

(b)Case 2:
The second case is that both the given sets of integers are in increasing order.

(c)Case 3:
The third and the last case is that both the given sets of integers are in decreasing order.

2 Algorithm Design

The central part of the algorithm is:

```
if a[m]<b[n] then a[m+i]:=b[n]
else b[n+1]:=a[m]
i:=1 ; j:=1
nm:=n+m
for k:=1 to nm do
  begin
    if a[i]<b[j] then
      begin
        c[k]:=a[i]
        i:=i+1
      end
    else
      begin
        c[k]:=b[j]
          j:=j+1
      end
  end
```

This algorithm is although a clean and simple implementation can in certain circumstances do considerably more comparisons than are

necessary. These unnecessary comparisons can be easily avoided by optimizing the above stated algorithm with the introduction of the previously discussed three cases.

2.1 Optimized Algorithm Description

(1) procedure merge

1. Establish the arrays a[1...m] and b[1...n]
2. If last a element less than or equal to last b element then
(a) merge all of a with b
(b) copy rest of b
else
(a') merge all of b with a
(b') copy rest of a
3. Return the merged result c[1...n+m]

(2) procedure mergecopy

1. Establish the arrays a[1...m] and b[1...n] with a[m]<=b[n]
2. If last element of a less than or equal to first element of b then
(a) copy all of a into first m elements of c
(b) copy all of b into c starting at m+1
else
(a') merge all of a with b into c
(b') copy rest of b into c starting at position just past where merge finished

(3) procedure shortmerge

1. Establish the arrays a[1...m] and b[1...n] with a[m]<=b[n]
2. While all of a array still not merged do
(a) if current a element less than or equal to current b element then
(a.1) copy current a into the current c position
(a.2) advance pointer to next position in a array
else
(a'.1) copy current b into the current c position
(a'.2) advance pointer to next position in b array
(b) advance pointer for c array by one

(4)procedure copy

1. Establish the arrays b[1...n] and c[1...n+m] and also establish where copying is to begin in b(i.e. at j) and where copying isto begin in c(i.e. at k)
2. While the end of the b array is still not reached do
(a) copy element from current position in b to current position in c
(b) advance pointer j to next position in b
(c) advance pointer k to next position in c

2.2 Pseudo Code

The pseudo code for the above stated algorithm,

Algorithm 1 Merge

```
1:  procedure COPY(b, c, j, n, k)
2:      i ← j
3:      while i ≠ n do
4:          c[k] ← b[i]
5:          k ← k + 1
6:      end while
7:  end procedure
8:  procedure SHORTMERGE(a, b, c, m, j, k)
9:      i ← index for array a
10:     while i ≤ m do
11:         if a[i] ≤ b[j] then
12:             c[k] ← a[i]
13:             i ← i + 1
14:         else
15:             c[k] ← b[j]
16:             j ← j + 1
17:         end if
18:         k ← k + 1
19:     end while
20: end procedure
21:
22: procedure MERGECOPY(a, b, c, m, n)
23:     i ← First position in array a
24:     j ← Current position in array b
25:     k ← Current position in merged array c
26:     i ← 1
27:     j ← 1
28:     k ← 1
29:     if a[m] ≤ b[i] then
30:         COPY(a,c,i,m,k)
31:         COPY(b,c,j,n,k)
32:     else
33:         SHORTMERGE(a,b,c,m,j,k)
34:         COPY(b,c,j,n,k)
35:     end if
36: end procedure
37:
38: procedure MERGE(a, b, c, m, n)
39:     if a[m] ≤ b[n] then
40:         MERGECOPY(a,b,c,m,n)
41:     else
42:         MERGECOPY(b,a,c,n,m)
43:     end if
44: end procedure
```

3 Analysis and Discussions

3.1 Time Complexity

The time complexity for the above stated algorithm can be computed by adding units of time taken by each statement multiplied by the number of times the statement is executed. Hence,

$T = \Sigma(cost(i) * no.of.times(i))$,for all statements

The running time will be dependent on the number of elements of the two sets. The running time for best, average and worst cases as a function of number of elements can be written as,

$$T_{best} \propto 4 + 6(n+m) - 6c_1 - 6c_2$$

$$T_{average} \propto 16 + 11m + 6n - 6c_3$$

$$T_{worst} \propto 4 + 11m + 6n - 6c_4$$

Hence the order of growth for best case, average case and best case are

$\Omega(n+m), \Theta(n+m), O(n+m)$ respectively.

The time taken by all the three cases will differ only in the constants depending on the number of comparisons made. For the best case, there will be only two comparisons made. For the worst case, all the elements of one of the sets will be compared to all elements of the other set. Any intermediate number of comparisons will result into average case.

For the dataset,

Table 1: Data for plotting running time graph

n	T_{best}	T_{avg}	T_{worst}
25	512	583	658
125	2412	2778	3154
225	4312	4981	5645
325	6212	7168	8149
425	8112	9381	10583
525	10012	11571	13087
625	11912	13742	15642

We will get the following time complexity graph after plotting almost 100 points for 100 different values of n,

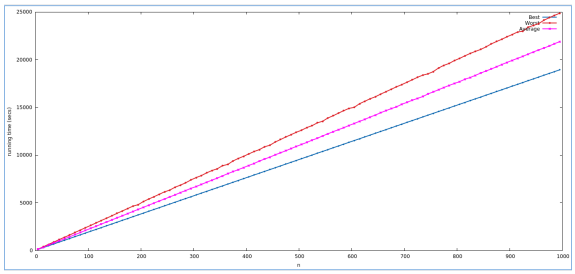

Figure 1: Running time graph.

3.2 Space Complexity

Suppose the input sets are of size n and m, then the resultant merged set will be of size n+m. That is the space occupies is directly proportional to the sum of number of elements of the input sets.

$$Space(n, m) \propto (n + m)$$

Hence order of growth of Space is linear.

4 Experimental Study

We may assume the following examples to study the various running times possible,

Best Case: **A**=[2,3,5,9,15], **B**=[18,20,25,29,35] and the resulting set will be **C**=[2,3,5,9,15,18,20,25,29,35]. This type of test cases are the best cases as it takes only one comparison to combine both these sets.

Average Case: **A**=[1,5,6,8,9], **B**=[5,11,15,16,18,20] and the resulting set will be **C**=[1,5,5,6,8,9,11,15,16,18,20]. This type of test cases are the average cases as it do not take all of (5+6=11 i.e n+m) comparisons but it only takes (5+2=7) comparisons.

Worst Case: **A**=[1,5,7,12,20], **B**=[2,6,8,16,18,20] and the resulting set will be **C**=[1,2,5,6,7,8,12,16,18,20,20]. This type of test cases are the worst cases as it takes all of (5+6=11 i.e.n+m) comparisons.

Thus we can claim that the running time depends on the size of the input sets and the number of comparisons performed to merge the two sets.

5 Conclusion

In this paper we proposed an algorithm to merge two sorted sets into one single sorted set in O(n + m) time complexity and linear space complexity.However experimental study

showed that the time complexity for best,average and worst case differs only in the constants depending upon the number of comparisons made. The above algorithm is a vital part of a merge sort.

Hence practical implications are: It is used in many e-commerce websites where sorting plays an important role as items you order are arranged in sorted fashion.

Ideally, all of this would result in successful attempts at utilizing the above algorithm for specific fields especially analytics involving large data sets.

References

[1] Introduction To Algorithms (Second Edition) by Thomas H.Cormen, Charles E.Leiserson , R.L.Rivest and Clifford Stein.

[2] How to Solve it by Computer by R.G.Dromey.

[3] C in Depth (3rd Edition) by Deepali Srivastava.

[4] The C Programming Language by Brian Kernighan and Dennis Ritchie.

[5] https://www.geeksforgeeks.org/

[6] https://www.stackoverflow.com/

[7] https://www.gnuplotting.org/