```
    //AmicablePair sum computing method
  private int    factorsSum(int n) {
        long startTime = System.nanoTime();//Time count
        int j, sum = 0, num = 0;
        int[] sums = new int [n+1];
        for (int i = 1; i <=n/2; i++) {
              j = i*2;
              while (j <= n) {
              sums[j] = sums[j]+i; // add factor i to every sums in the list
              j = j+i;
              }
        }
        for (int i = 2; i <= n; i++) {
              sum = sums[i];
              if (sum > n || sum <= i)// avoid sum out of n and delete repeating such as
"284-220" from"220-284 "
                    continue;
              else {
                    if (sums[sum] == i) {// Judge Amicable Pair
                          System.out.println(num+": "+i+" and "+sum); //output Amicable Pair
                          num++;
                          }
                    }
              }
      long endTime = System.nanoTime();// Time count
      double d2 = u.timeInSec(endTime,startTime) ;// Time count
      System.out.println("AmicablePair "    + " CPU time = " + d2 + " seconds"); // Time count
      return num;
  }
```

By regular methods, we need try every number from 2 to sqrt(n) to compute remainder (%) to judge whether it is one of the factors of n. And the final Time Complexity would be O(nsqrt(n)), that will cost so long time.

So I used this optimized algorithm as followed:

	n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
			1	1	1	1	1	1	1	1	1	1	1	1	1	1	…
i=2	j=4				2		2		2		2		2		2		…
i=3	j=6						3			3			3			3	…
i=4	j=8								4				4				…
i=5	j=10										5						…
i=6	j=12												6				…

……

Sums[ ]: Sums[1] sums[2]……

We created a Sums[ ] Array to add up the potential factors except themselves as the table above. In all ,we only need to count: n/2+n/3+n/4+n/5+…+1/n+n =nlogn times. So that we could get Time Complexity of O(nlogn), which is about 300 times faster than the O(nsqrt(n)) when n equals 100 million. Finally I got the result within 20 seconds(17 s).