```
Algorithm Mergesort ( low, high)
// a[low:high] is a global array to be sorted
// Small(P) is true if only one element to sort.
   In this case list is already sorted.
{
   if (low < high) then
      {
         mid = [low + high] /2
         Mergesort (low, mid)
         Mergesort (mid+1, high)
         Merge ( low, mid, high).
      }
}
```

```
Algorithm Merge (low, mid, high)
// a[low:high] temporary array.
{ i=low ; j = mid+1 ; k = low.

  while((i≤mid) and (j ≤ high))
  {
    if ( a[i] ≤ a[j]) then.
      {
        b[k] = a[i];
        i = i+1;
      }
    else
      { b[k] = a[j] ;
        j = j+1;
      }
    k = k+1;
  }

  if ( i > mid) then
    { for( while( j ≤ high)
          { b[k] = a[j]
            k++; j++;
          }
    }
  else
      { while ( i ≤ mid)
          { b[k] = a[i];
            k++; i++;
          }
      }
  for  i = low to high
       a[i] = b[i].
}
```

Time Complexity Analysis

$$T(n) = \begin{cases} a & n = 1, \ a \text{ constant} \\ 2T(n/2) + cn & n > 1 \end{cases}$$

$T(n) = 2T(n/2) + cn$ let $n = 2^k$

$= 2(2T(n/4) + cn/2) + cn$ $k = \log n$.

$= 2^2 T(n/2^2) + 2cn$

$= 2^2(2T(n/2^3) + cn/2^2) + 2cn$

$= 2^3 T(n/2^3) + 3cn$

$= 2^k T(n/2^k) + kcn$

$= n \cdot a + cn \log n$

$= O(n \log n)$