int pow\_2( int n )

{  
      if ( n==1) 1+ n>1  
               return 2;   
      if ( n > 1) 1+ n>2

     return ( 2 \* pow\_2( n-1 )  ); 2+T(n-1)

}

T(n) = T(n-1) + C

= T(n-2)+ 2C

= T(n-k)+kC

n-k=0 => n=k

T(n) = T(0) +nC

T(n) = nC+1

O(n)

assume (iEnd - iBegin + 1) is n  (that's the size of A), this algorithm will take c1.n + c0 to finish. Please represent the time complexity of TopDownSplitMerge(A[], iBegin, iEnd, B[]) as a recurrence equation.  You don't need to solve this equation.

TopDownMerge(A[], iBegin, iMiddle, iEnd, B[])

{

i = iBegin, j = iMiddle;

for (k = iBegin; k < iEnd; k++) n

{

if (i < iMiddle && (j >= iEnd || A[i] <= A[j])) +1

{

B[k] = A[i];

i = i + 1;

}

else +1

{

B[k] = A[j];

j = j + 1;

}

}

}

T(n) = 2T(n/2)+n

L(k)=2^k \*L(0)+k\*2^k => O(nlog(n))