```
----------------------------------------------------------------
sort_algorithms_2.h: shell sort variants, mergesort
----------------------------------------------------------------

#ifndef __SORT_2_H__
#define __SORT_2_H__

#include <algorithm>
#include <cmath>
#include <vector>

template <typename T>
void shell_1_sort(std::vector<T> &A) {
  int i, j, N=A.size(), gap;

  for (gap=N/2; 0<gap; gap/=2) {
    for (i=gap; i<N; i++) {
      T tmp = A[i];
      for (j=i; gap<=j && tmp<A[j-gap]; j-=gap)
        A[j] = A[j-gap];
      A[j] = tmp;
    }
  }
}

template <typename T>
void shell_2_sort(std::vector<T> &A) {
  int i, j, N=A.size(), k, gap;

  for (k=(int)std::log2(N+1); 0<k; k--) {
    for (gap=(1<<k)-1, i=gap; i<N; i++) {
      T tmp = A[i];
      for (j=i; gap<=j && tmp<A[j-gap]; j-=gap)
        A[j] = A[j-gap];
      A[j] = tmp;
    }
  }
}

----------------------------------------------------------------
Hint: When 1<gap, data is moved closer to where it ultimately
should be in the sorted sequence. When gap=1, shell sort becomes
regular insertion sort, at which point data is moved as needed.
----------------------------------------------------------------
```

```
template <typename T>
void merge(std::vector<T> &A, std::vector<T> &tmpA,
           int left, int middle, int right) {
  int i=left, j=middle+1, k=left;

  while (i<=middle && j<=right) {
    if (A[i] < A[j]) tmpA[k++] = A[i++];
    else             tmpA[k++] = A[j++];
  }

  while (i<=middle) tmpA[k++] = A[i++];
  while (j<=right)  tmpA[k++] = A[j++];

  std::copy(&tmpA[left], &tmpA[right+1], &A[left]);
}

template <typename T>
void mergesort(std::vector<T> &A, std::vector<T> &tmpA,
               int left, int right) {
  if (left == right)
    return;

  int middle = (left+right)/2;

  mergesort(A, tmpA, left, middle);
  mergesort(A, tmpA, middle+1, right);
  merge(A, tmpA, left, middle, right);
}

template <typename T>
void mergesort(std::vector<T> &A) {
  std::vector<T> tmpA(A.size());
  mergesort(A, tmpA, 0, A.size()-1);
}

#endif

----------------------------------------------------------------
Hint: The call to mergesort is translated into a recursion that
includes temporary storage (tmpA). The recursion continues to
split the input array until one element remains (left == right).
Subarrays are sorted as they get merged. This is merely a matter
of determining which element to pick next.
----------------------------------------------------------------
```