```
#include "Sort.h"

int getSumOfLetterCaseInsensitive(short index_letter, unsigned int index, Histog
ram **histogram);
Histogram** merge(Histogram **left, unsigned int *size_left, Histogram **right,
unsigned int *size_right, Histogram** data);

Histogram** sort(Histogram **data, unsigned int *size) {
  //printf("call sort(), size = %d\n", *size);
  //printHistogramArray(data, *size);

  if (*size == 1) { // Abbruchbedingung der Rekursion!!
    // Wenn nur ein Element vorhanden ist gebe dieses zurÃ¼ck.
    return data;
  }

  // Elemente in zwei HÃ¤lften teilen
  unsigned int size_left = *size / 2;
  unsigned int size_right = size_left;

  if (*size % 2 != 0) {
    size_right++;
  }

  // Hier wird das Array von Adressen in zwei HÃ¤lften aufgeteilt.
  Histogram **left = (Histogram**) malloc (sizeof(Histogram*) * size_left);
  Histogram **right = (Histogram**) malloc (sizeof(Histogram*) * size_right);

  // Elemente in den neuen Speicher kopieren.
  //memcpy (left, data, size_left*sizeof(Histogram)); // kopiere die erste HÃ¤lf
te
  //memcpy (right, (data+size_left), size_right*sizeof(Histogram)); // kopiert d
ie zweite HÃ¤lfte der Elemente
    // JETZT MÃM-^\SSEN NICHT MEHR DIE ELEMENTE KOPIERT WERDEN SONDERN NUR NOCH
DIE ADRESSEN MITTELS EINER FOR SCHLEIFE

    unsigned int i;
    for (i = 0; i < size_left; i++) {
        left[i] = data[i];
    } // linke hÃ¤lfte

    for (i = size_left; i < (size_right+size_left); i++) {
        right[i-size_left] = data[i];
    } // rechte HÃ¤lfte

  // Speicher Freigeben
  //free(data);

  // nun gehts weiter mit Rekursion
  left = sort(left, &size_left);
  right = sort(right, &size_right);

  // mergen und das Ergebnis zurÃ¼ckgeben
  Histogram **result = merge(left, &size_left, right, &size_right, data);

  free(left);
  free(right);

  return result;
}

Histogram** merge(Histogram **left, unsigned int *size_left, Histogram **right,
```



```
unsigned int *size_right, Histogram **merged) {
  //printf("call merge()\n");
  // Allokiere Speicher, so das beide HÃ¤lften (sortiert) hier gespeichert werde
n kÃ¶nnen.
  //Histogram *merged = (Histogram*) malloc (sizeof(Histogram) * (*size_left + *
size_right));

    // reset merged Array
    unsigned int n;
    for (n = 0; n < (*size_left + *size_right); n++) {
        merged[n] = NULL;
    }


  unsigned int index_merged = 0;

  unsigned int index_left = 0;
  unsigned int index_right = 0;

  bool didMerge = FALSE;

    // 08.08.2011 um 12:14 Uhr.... hier gehts weiter!

  while (index_left < (*size_left) || index_right < (*size_right)) { // Solange
der Index der jeweiligen HÃ¤lfte nicht grÃ¶Ã-^_er / gleich ist wie die size
    if (index_left < (*size_left) && index_right < (*size_right)) { // ...
      // Wir gehen die HÃ¤lften Elementweise durch

      short i;
      for (i = 0 ; i < 26 ; i++) {
        // Vergleiche die Anzahl der Buchstaben (GroÃ-^_ und Klein).
        short sumLeftHistogram = getSumOfLetterCaseInsensitive(i, index_left, le
ft);
        short sumRightHistogram = getSumOfLetterCaseInsensitive(i, index_right,
right);

        if (sumLeftHistogram < sumRightHistogram) { // Rechts hat mehr gleiche B
uchstaben
          // ok rechtes Histogram kommt zuertst
          // memcpy((merged+index_merged), (right+index_right), sizeof(Histogram
)); // kopiere Histogram.
                    // kopiere Histogram Adresse der rechten HÃ¤lfte am aktuelle
n Index
                    merged[index_merged] = right[index_right];
          index_merged++;
          index_right++;
          didMerge = TRUE;
          break;
        }

        if (sumLeftHistogram > sumRightHistogram) {
          // linkes Histogram kommt zuerst
          //memcpy((merged+index_merged), (left+index_left), sizeof(Histogram));
          merged[index_merged] = left[index_left];
                    index_merged++;
          index_left++;
          didMerge = TRUE;
          break;
        }

        if (sumLeftHistogram == sumRightHistogram) {
          // Wenn die gleiche Anzahl an Buchstaben, dann kommt das Histogramm mi
```

```
t den meisten kleineren Buchstaben
          short lowerCaseLetterLeftHistogram = (*left[index_left]).letter[i+26];
          short lowerCaseLetterRightHistogram = (*right[index_right]).letter[i+2
6];

          if (lowerCaseLetterRightHistogram > lowerCaseLetterLeftHistogram) {
            // ok rechtes Histogram kommt zuertst
            //memcpy((merged+index_merged), (right+index_right), sizeof(Histogra
m));
            merged[index_merged] = right[index_right];
                        index_merged++;
            index_right++;
            didMerge = TRUE;
            break;
          }

          if (lowerCaseLetterRightHistogram < lowerCaseLetterLeftHistogram) {
            // linkes Histogram kommt zuerst;
            //memcpy((merged+index_merged), (left+index_left), sizeof(Histogram)
);
                        merged[index_merged] = left[index_left];
            index_merged++;
            index_left++;
            didMerge = TRUE;
            break;
          }

          if (lowerCaseLetterLeftHistogram == lowerCaseLetterRightHistogram) {
            continue;
          }
        }
      }

      if (didMerge == FALSE) {
         // linkes Histogram kommt zuerst;
         //memcpy((merged+index_merged), (left+index_left), sizeof(Histogram));
            // wenn beide identisch sind, dann nehme das linke Histogram, hÃ¤tte
 auch das rechte sein kÃ¶nnen, ist ja egal da identisch!
                    merged[index_merged] = left[index_left];
                    index_merged++;
         index_left++;
      }

      didMerge = FALSE;

    } else {
      // Nur linke HÃ¤lfte enthÃ¤lt Elemente
      if (index_left < (*size_left)) {
        // memcpy((merged+index_merged), (left+index_left), sizeof(Histogram));
                merged[index_merged] = left[index_left];
        index_merged++;
        index_left++;
      }

            // Nur die rechte HÃ¤lfte enthÃ¤lt noch Elemente
      if (index_right < (*size_right)) {
        // memcpy((merged+index_merged), (right+index_right), sizeof(Histogram))
;
                merged[index_merged] = right[index_right];
        index_merged++;
        index_right++;
      }
```



```
    }
  }

  return merged;
}

int getSumOfLetterCaseInsensitive(short index_letter, unsigned int index, Histog
ram **histogram) {
    if (index_letter < 26) {
        int upperCaseLetter = (*histogram[index]).letter[index_letter];
            int lowerCaseLetter = (*histogram[index]).letter[index_letter+26];

        return (upperCaseLetter + lowerCaseLetter);
    }

    return 0;
}
```