**M**erge sort

**M**erge sort is based on the **divide-and-conquer** paradigm. It is a non-tail recursion algorithm.

To sort A[p .. r]:

1. **Divide Step**

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split A[p .. r] into two subarrays A[p .. q] and A[q + 1 .. r], each containing about half of the elements of A[p .. r]. That is, q is the halfway point of A[p .. r].

2. **Conquer Step**

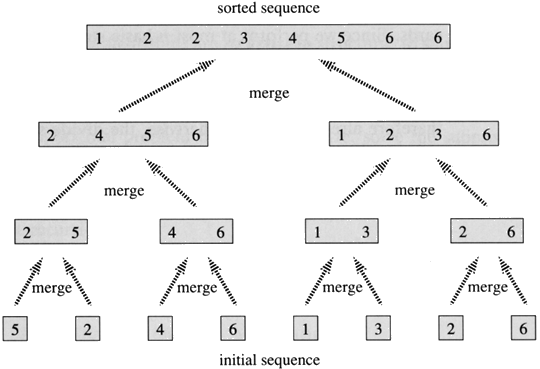
Conquer by recursively sorting the two subarrays A[p .. q] and A[q + 1 .. r].

3. **Combine Step**

Combine the elements back in A[p .. r] by merging the two sorted subarrays A[p .. q] and A[q + 1 .. r] into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Note that the recursion bottoms out when the subarray has just one element, so that it is trivially sorted.

Example: Bottom-up view of the above procedure for n = 8.



**Idea Behind Linear Time Merging**

Think of two piles of cards, Each pile is sorted and placed face-up on a table with the smallest cards on top. We will merge these into a single sorted pile, face-down on the table.

A basic step:

* Choose the smaller of the two top cards.
* Remove it from its pile, thereby exposing a new top card.
* Place the chosen card face-down onto the output pile.
* Repeatedly perform basic steps until one input pile is empty.
* Once one input pile empties, just take the remaining input pile and place it face-down onto the output pile.

**Running Time**

Each basic step should take constant time, since we check just the two top cards. There are at most *n* basic steps, since each basic step removes one card from the input piles, and we started with *n* cards in the input piles. Therefore, this procedure should take Θ(*n*) time.

int numbers[]={1,5,8,3,6,9,11};

int helper[6];

int number=7;

void mergesort(int low, int high) {

if (low < high) {

int middle = low + (high - low) / 2; // Get the index of middle element

mergesort(low, middle); // Sort the left side of the array

mergesort(middle + 1, high); // Sort the right side of the array

merge(low, middle, high); // Combine them both

}

}

void merge(int low, int middle, int high) {

int i = low;

int j = middle + 1;

int k = low;

// Copy both parts into the helper array. The key is this part which allows for the

//last item to be resolved in the right array.

for (int i = low; i <= high; i++)

helper[i] = numbers[i];

// Copy the smallest values from either left or right side back to the original array

while (i <= middle && j <= high) {

if (helper[i] <= helper[j]) {

numbers[k] = helper[i];

i++;

} else {

numbers[k] = helper[j];

j++;

}

k++;

}

// Copy the rest of the left side of the array into the target array

while (i <= middle) {

numbers[k] = helper[i];

k++;

i++;

}

/\***Don’t need to copy the right side into the target array because the last elements**

**will already be in the right place.** \*/

}

int main() {

mergesort(0, number - 1);

}

1,5,8,3,6,9,11

Numbers array

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | | 3 | 4 | | 5 | 6 | | Array subscript |
| 1 | 5 | 8 | | 3 | 6 | | 9 | 11 | |  |
| 1 | 5 |  | | | | | | | | Merge(0,1,1) |
|  | | 3 | | 8 |  | | | | | Merge(2,2,3) |
| 1 | 3 | 5 | | 8 |  | | | | | Merge(0,1,3) |
|  | | | | | 6 | | 9 |  | | Merge(4,4,5) |
|  | | | | | 6 | | 9 | 11 | | Merge(4,5,6) |
| 1 | 3 | 5 | | 6 | 8 | | 9 | 11 | | Merge(0,3,6) |
| **Mergesort (0, 6)**  Middle=3  Mergesort(0,3)  Mergesort(4,6)  Merge(0,3,6) | | | **Mergesort(0,3)**  Middle =1  Mergesort(0,1)  Mergesort(2,3)  Merge(0,1,3) | | | **Mergesort(0,1)**  Middle=0  Mergesort (0,0)  Mergesort(1,1)  Merge(0,1,1) | | | **Mergesort(0,0)**  DONE | |
| **MergeSort(1,1)**  DONE | |
| **Merge(0,1,1)**  1,5 | |
| **Mergesort(2,3)**  Middle=2  Mergesort(2,2)  Mergesort(3,3)  Merge(2,2,3) | | | **Mergesort(2,2)**  DONE | |
| **Mergesort(3,3)**  DONE | |
| **Merge(2,2,3)**  3,8 | |
| **Merge(0,1,3)**  1,3,5,8 | | | xxxxxxxxx | |
| **Mergesort(4,6)**  Middle =5  Mergesort(4,5)  Mergesort(6,6)  Merge(4,5,6) | | | **Mergesort(4,5)**  Middle =4  Mergesort(4,4)  Mergesort(5,5)  Merge(4,4,5) | | | **MergeSort(4,4)**  DONE | |
| **MergeSort(5,5)**  DONE | |
| **Merge(4,4,5)**  6,9 | |
| **MergeSort(6,6)**  DONE | | | Xxxxxxxxxxxxxx | |
| **Merge(4,5,6)**  6,9,11 | | |
|  | | | **Merge(0,3,6)**  1,3,4,6,8,9,11 | | |  | | |  | |