Merge Sort

Brian E. Lavender

April 15, 2016

# Merge Sort Concept

Sorts an array of values

* **Divide and Conquer**:
* **Recursive Routine**
* **Complexity**. O(Nlog2N)
* **Disadvantage**: auxiliary array

# Separate video

Harvard University [Merge Sort video](https://youtu.be/EeQ8pwjQxTM)

# Merge two sorted halves Example

Two sorted arrays of size *k/2*.
**Merge** them to destination of size *k*.

min(4,8)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| **4** | 15 | 16 | 50 |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| **8** | 23 | 42 | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ |

# Merge Half Arrays To Destination (Step 1)

Move *4* from left\_half to destination

min(15,8)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| \* | **15** | 16 | 50 |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| **8** | 23 | 42 | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 |  |  |  |  |  |  |  |

# Merge Half Arrays To Destination (Step 2)

Move 8 from right\_half to destination.

min(15,23)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  | **15** | 16 | 50 |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| \* | **23** | 42 | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 |  |  |  |  |  |  |

# Merge Half Arrays To Destination (Step 3)

Move 15 from left half to destination.

min(16,23)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  | \* | **16** | 50 |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  | **23** | 42 | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 |  |  |  |  |  |

# Merge Half Arrays To Destination (Step 4)

Move 16 from left\_half to destination.

min(50,23)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  | \* | **50** |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  | **23** | 42 | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 | *16* |  |  |  |  |

# Merge Half Arrays To Destination (Step 5)

Move 23 from the right\_half to destination.

min(50,42)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  | **50** |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  | \* | **42** | 108 |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 | 16 | *23* |  |  |  |

# Merge Half Arrays To Destination (Step 6)

Move 42 from the right\_half to destination.

min(50,108)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  | **50** |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  | \* | **108** |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 | 16 | 23 | *42* |  |  |

# Merge Half Arrays To Destination (Step 7)

Move 50 from the left\_half to destination.

empty(left\_half)?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  | \* |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  | **108** |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 | 16 | 23 | 42 | *50* |  |

# Merge Half Arrays To Destination (Step 8)

Move 108 from the right\_half to destination.

(have\_items(left\_array) or have\_items(right\_array))?

Left Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  |  |

Right Half (size 4)

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
|  |  |  | \* |

Destination (size 8)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | 8 | 15 | 16 | 23 | 42 | 50 | *108* |

Destination array **sorted**

8 steps *linear time*.

# The C++ merge code

C++ Code illustrating the above merge. Consider four cases.

1. Either *left\_half* or *right\_half* have elements to move to the destination.
2. All elements from *left\_half* moved to destination. Move minimum element from *right\_half*.
3. All elements from *right\_half* moved to destination. Move minimum element from *left\_half*.
4. Move smaller element from *left\_half* or *right\_half* to destination.

const int SIZE\_HALF\_LEFT = 4;
const int SIZE\_HALF\_RIGHT = 4;
const int SIZE\_DEST = 8;

int left\_half[SIZE\_HALF\_LEFT] = {4, 15, 16, 50};
int right\_half[SIZE\_HALF\_RIGHT] = {8, 23, 42, 108};
int destination[SIZE\_DEST];

// index counters
// i - left array , j - right array, k - destination array
i = 0; j = 0; k = 0;

// Merge the two half lists together
// while we have elements in either of the two lists
while (i < SIZE\_HALF\_LEFT || j < SIZE\_HALF\_RIGHT) {
 // Completed the first half
 if ( i >= SIZE\_HALF\_LEFT ) {
 destination[k] = right\_half[j];
 j++;
 }

 // Completed the second half
 else if (j >= SIZE\_HALF\_RIGHT) {
 destination[k] = left\_half[i];
 i++;
 }

 // pick the smallest element from one
 // of the two lists
 else if (left\_half[i] < right\_half[j]) {
 destination[k] = left\_half[i];
 i++;
 } else {
 destination[k] = right\_half[j];
 j++;
 }

 // increment our counter for destination
 k++;
}

# Next Challenge

Merge assumes *left\_half* and *right\_half* **already sorted**.

If we start with the following.

int destination[] = {108, 15, 50, 4, 8, 42, 23, 16};

Halves **not sorted**!

int left\_half[] = {108, 15, 50, 4};
int right\_half[] = {8, 42, 23, 16};

Keep dividing in half?

# Get to the bottom of this

Repeatedly divide in half!

log28 = 3 levels

--- 8
^ 4 4
3 levels 2 2 2 2
v 1 1 1 1 1 1 1 1
---

# Divide array into halves

Repeatedly call merge\_sort on halves!

int destination[] = {108, 15, 50, 4, 8, 42, 23, 16};

## First Division

Sorted? No

int left\_half[] = {108, 15, 50, 4};
int right\_half[] = {8, 42, 23, 16};

## Second Division

Sorted? No

int left\_half[] = {108, 15};
int right\_half[] = {50, 4};

int left\_half[] = {8, 42};
int right\_half[] = {23, 16};

## Third Division

Sorted? **Yes**

int left\_half[] = {108};
int right\_half[] = {15};

int left\_half[] = {50};
int right\_half[] = {4};

int left\_half[] = {8};
int right\_half[] = {42};

int right\_half[] = {23};
int right\_half[] = {16};

# Merge two half arrays of size 1 to destination of size 2

Build it back up

min(23,16)?

Left Half (size 1)

|  |
| --- |
| 0 |
| **23** |

Right Half (size 1)

|  |
| --- |
| 0 |
| **16** |

Destination (size 2)

|  |  |
| --- | --- |
| 0 | 1 |
|  |  |

# Merge two half arrays of size 1 to destination of size 2 (step 1)

Move *16* from right\_half to destination

empty(right\_half)?

Left Half (size 1)

|  |
| --- |
| 0 |
| **23** |

Right Half (size 1)

|  |
| --- |
| 0 |
| \* |

Destination (size 2)

|  |  |
| --- | --- |
| 0 | 1 |
| *16* |  |

# Merge two half arrays of size 1 to destination of size 2 (step 2)

Move 23 from first\_half to destination

Left Half (size 1)

|  |
| --- |
| 0 |
| \* |

Right Half (size 1)

|  |
| --- |
| 0 |
|  |

Destination (size 2)

|  |  |
| --- | --- |
| 0 | 1 |
| 16 | 23 |

We merged the simplest two half arrays of size 1 back into a dest array of 2 items.

# Algorithm for recursion

1. Figure out our base case
2. Determine our recursive step.

# Base Case

1 item already sorted.

void merge\_sort(int destination[], int dest\_size) {
 // base case. Array size is one.
 if (dest\_size == 1)
 return;

}

# Recursive Step

1. Make new left\_half and right\_half. Copy data from destination to halves.
2. Call *merge\_sort* on each half array.
3. Merge the sorted halves together.
4. Return the destination array

# Divide the Array

Making new halves.

void merge\_sort(int destination[], int dest\_size) {
 int left\_size, right\_size;
 if (dest\_size == 1)
 return;
 else {
 // \*\*\*\*\*\*\*\*\*\*\*
 // Make two new half arrays
 // \*\*\*\*\*\*\*\*\*\*\*
 // Make two new arrays: left\_array half and right\_array half
 // Integer division for size: 5 / 2 -> 2
 size\_left = dest\_size / 2 ;
 size\_right = array\_size - size\_left;
 int left\_array[size\_left];
 int right\_array[size\_right];
 for (i=0; i< size\_left; i++)
 left\_array[i] = x[i];
 for (i=0; i< size\_right; i++)
 right\_array[i] = x[size\_left + i];

 // call merge\_sort on left\_array
 // call merge\_sort on right\_array
 // merge the left\_array and right\_array together
 }
 // return our sorted array
 return;
}

# Recursively Call Merge sort on halves

Recursively call *merge\_sort* on the two halves.

void merge\_sort(int destination[], int dest\_size) {
 int left\_size, right\_size;
 if (size == 1)
 return;
 else {
 // Make two new arrays: left\_array half and right\_array half
 // Integer division for size: 5 / 2 -> 2
 size\_left = dest\_size / 2 ;
 size\_right = dest\_size - size\_left;
 int left\_array[size\_left];
 int right\_array[size\_right];
 for (i=0; i< size\_left; i++)
 left\_array[i] = x[i];
 for (i=0; i< size\_right; i++)
 right\_array[i] = x[size\_left + i];

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*
 // Recursively Call Merge sort on halves
 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*
 merge\_sort(left\_array,size\_left);
 merge\_sort(right\_array,size\_right);

 // Merge the two halves
 // Our inductive step
 }
 // return our sorted array
 return;
}

# Side Step

Pause for a moment and make a function out our merge example previously illustrated.

void merge(int left[], int right[], int destination[],
 int size\_of\_left\_half, int size\_of\_right\_half) {
 int i = 0; // works on left half
 int j = 0; // works on right half
 // The merge
 while (i < size\_of\_left\_half || j < size\_of\_right\_half) {
 // Completed the first half
 if ( i >= size\_of\_left\_half ) {
 destination[k] = right[j];
 j++;
 }
 // Completed the second half
 else if (j >= size\_of\_right\_half) {
 destination[k] = left[i];
 i++;
 }
 // pick the smallest one
 else if (left\_half[i] < right\_half[j]) {
 destination[k] = left[i];
 i++;
 } else {
 destination[k] = right[j];
 j++;
 }
 k++;
 }
}

# Merge the two halves

Merge left\_half and right\_half back to destination.

void merge\_sort(int destination[], int dest\_size) {
 int left\_size, right\_size;
 if (dest\_size == 1)
 return;
 else {
 // Make left\_array half and right\_array half
 size\_left = dest\_size / 2 ;
 size\_right = dest\_size - size\_left;
 int left\_array[size\_left];
 int right\_array[size\_right];
 for (i=0; i< size\_left; i++)
 left\_array[i] = destination[i];
 for (i=0; i< size\_right; i++)
 right\_array[i] = destination[size\_left + i];

 // recursively call merge\_sort on the
 // left\_half and right\_half
 merge\_sort(left\_array,size\_left);
 merge\_sort(right\_array,size\_right);

 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*
 // Merge the two sorted halves
 // Our inductive step
 // \*\*\*\*\*\*\*\*\*\*\*\*\*\*
 merge(left\_array,right\_array,destination,size\_left,size\_right);
 }
 // return our sorted array
 return;
}

# Return

Who called me?

# Final Code

The following is the entire merge sort program with a main driver. It also illustrates an array of an odd number of elements.

#include <iostream>

using namespace std;

void display(int x[], int size) {
 int i;
 cout << "Size is " << size << endl;
 for (i=0; i < size; i++)
 cout << x[i] << " ";
 cout << endl;
}

void merge(int left[], int right[], int destination[],
 int size\_of\_left\_half, int size\_of\_right\_half) {
 int i = 0; // works on left half
 int j = 0; // works on right half
 int k = 0; // merged array count
 // The merge
 while (i < size\_of\_left\_half || j < size\_of\_right\_half) {
 // Completed the first half
 if ( i >= size\_of\_left\_half ) {
 destination[k] = right[j];
 j++;
 }
 // Completed the second half
 else if (j >= size\_of\_right\_half) {
 destination[k] = left[i];
 i++;
 }
 // pick the smallest one
 else if (left[i] < right[j]) {
 destination[k] = left[i];
 i++;
 } else {
 destination[k] = right[j];
 j++;
 }
 k++;
 }

}

// precondition
// size of the array is 1 or greater
void merge\_sort(int destination[], int dest\_size) {
 int size\_left = 0, size\_right = 0;
 int i;

 // Our base case.
 // An array of one element is considered sorted
 if (dest\_size == 1)
 return ;

 // recursively call merge\_sort of two halves.
 else {
 size\_left = dest\_size / 2 ;
 size\_right = dest\_size - size\_left;

 // temporary arrays for divide portion
 int left\_array[size\_left];
 int right\_array[size\_left];

 // Copy data into new temp array for the left.
 for (i=0; i< size\_left; i++)
 left\_array[i] = destination[i];

 // Copy data into new temp array for the right.
 for (i=0; i< size\_right; i++)
 right\_array[i] = destination[size\_left + i];

 // recursively call merge\_sort on the
 // left\_half and right\_half
 merge\_sort(left\_array,size\_left);
 merge\_sort(right\_array,size\_right);

 // Merge the two halves
 // Our inductive step
 merge(left\_array,right\_array,destination,size\_left,size\_right);
 }

 return ;
}

int main()
{

 const int SIZE\_DEST = 11;

 int destination[SIZE\_DEST] = {101,99,21,12,11,25,20,4,2,17,3};

 cout << "Before sort" << endl;
 display(destination, SIZE\_DEST);

 merge\_sort(destination, SIZE\_DEST);

 cout << "After sort" << endl;
 display(destination, SIZE\_DEST);

 return 0;
}

# Questions?!

# Exercise

Randomly arrange the cups and see if you can perform the merge sort.

# References

1. Dale, Nell, *C++ Plus Data Structures, 3rd Ed.*, Jones and Bartlett Publishers, 2003, pp 601-608.