Merge Sort

By Slava Shaerman
(shaermv7765@student.laccd.edu)

History of Merge Sort

- Merge sort was one of the first sorting algorithms proposed for computing, brought forward by John von Neumann in 1945.
- Merge sort becomes one of the first divide-and-conquer sorting algorithms
- The algorithm has roots in card-sorting machines of the late 19th century

Von Neumann

The Merge Sort Algorithm

4	3	6	2	7	1	8	5

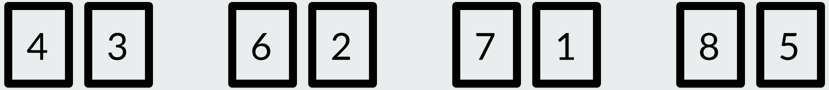
```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
if (length of array ≤ 1), then
          return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
          if (first element of left ≤ first element of right), then
                    append first element of left to final
                    remove first element of left
          else if (first element of right < first element of left), then
                    append first element of right to final
                    remove first element of right

if (left is still not empty), then
          append rest of left to final
else if (right is still not empty), then
          append rest of right to final
return final
```

The Merge Sort Algorithm

4	3	6	2		7	1	8	5

```
procedure: mergesort ($a_1, a_2, \ldots, a_n$: array of integers)
if (length of array ≤ 1), then
    return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
    if (first element of left ≤ first element of right), then
        append first element of left to final
        remove first element of left
    else if (first element of right < first element of left), then
        append first element of right to final
        remove first element of right

if (left is still not empty), then
    append rest of left to final
else if (right is still not empty), then
    append rest of right to final
return final
```

The Merge Sort Algorithm


4 3 6 2 7 1 8 5

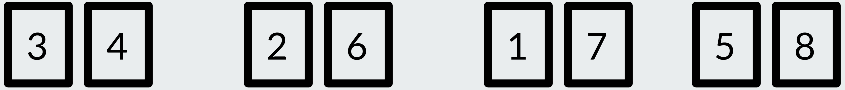
```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
if (length of array ≤ 1), then
    return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
    if (first element of left ≤ first element of right), then
        append first element of left to final
        remove first element of left
    else if (first element of right < first element of left), then
        append first element of right to final
        remove first element of right

if (left is still not empty), then
    append rest of left to final
else if (right is still not empty), then
    append rest of right to final
return final
```

The Merge Sort Algorithm

4	3	6	2	7	1	8	5

```
procedure: mergesort ($a_1, a_2, \ldots, a_n$: array of integers)
if (length of array ≤ 1), then
          return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
          if (first element of left ≤ first element of right), then
                    append first element of left to final
                    remove first element of left
          else if (first element of right < first element of left), then
                    append first element of right to final
                    remove first element of right

if (left is still not empty), then
          append rest of left to final
else if (right is still not empty), then
          append rest of right to final
return final
```

The Merge Sort Algorithm



3 4 2 6 1 7 5 8

```
procedure: mergesort ($a_1, a_2, ..., a_n$: array of integers)
if (length of array ≤ 1), then
    return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
    if (first element of left ≤ first element of right), then
        append first element of left to final
        remove first element of left
    else if (first element of right < first element of left), then
        append first element of right to final
        remove first element of right

if (left is still not empty), then
    append rest of left to final
else if (right is still not empty), then
    append rest of right to final
return final
```

The Merge Sort Algorithm

2	3	4	6		1	5	7	8

```
procedure: mergesort ($a_1, a_2, …, a_n$: array of integers)
if (length of array $\le$ 1), then
    return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
    if (first element of left $\le$ first element of right), then
        append first element of left to final
        remove first element of left
    else if (first element of right < first element of left), then
        append first element of right to final
        remove first element of right

if (left is still not empty), then
    append rest of left to final
else if (right is still not empty), then
    append rest of right to final
return final
```

The Merge Sort Algorithm

1	2	3	4	5	6	7	8

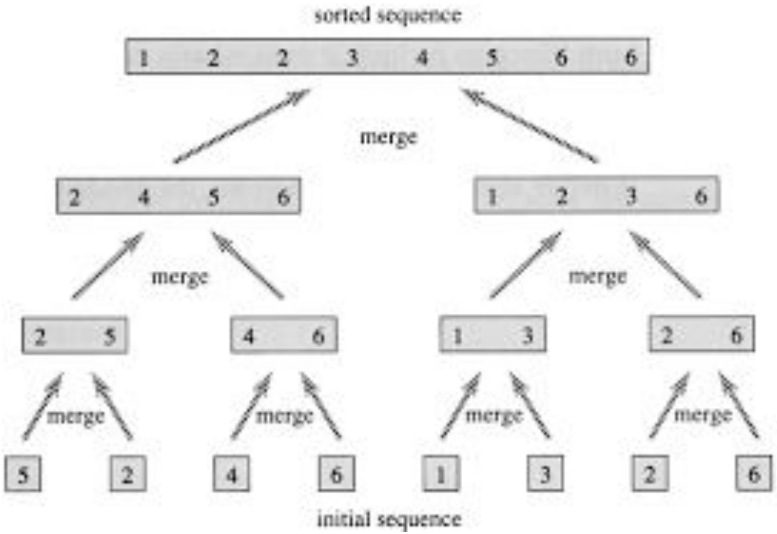
```
procedure: mergesort ($a_1, a_2, ..., a_n$: array of integers)
if (length of array ≤ 1), then
    return array
left := left half of input array
right := right half of input array
left := mergesort(left)
right := mergesort(right)

final := empty array
while (left is not empty and right is not empty) then
    if (first element of left ≤ first element of right), then
        append first element of left to final
        remove first element of left
    else if (first element of right < first element of left), then
        append first element of right to final
        remove first element of right

if (left is still not empty), then
    append rest of left to final
else if (right is still not empty), then
    append rest of right to final
return final
```

Complexity of Merge sort

Algorithm	Merge Sort	Insertion Sort (a simple sorting algorithm)
Best-case time complexity	$\Omega(nlogn)$	$O(n^2)$
Worst-case time complexity	$O(nlogn)$	$O(n)$ comparisons, $O(1)$ swaps
Space complexity	$O(n)$	$O(n)$

Alternative Merge Sort: Iterative Version

- We just examined a **top-down** implementation of Merge Sort, which uses recursion to sort the entire list
- However, we can implement a **bottom-up** Merge Sort, **without using recursion**
- To do this, we immediately start merging and sorting instead of first dividing the sequence into groups

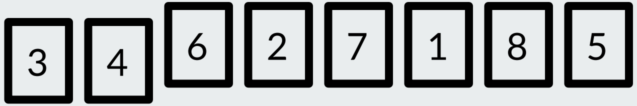


Bottom-up Merge Sort

4	3	6	2	7	1	8	5

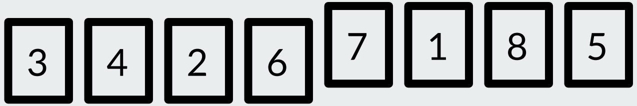
```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
        for (j=0, j<n, j+=2^(k+1)), do
                merge(list, final, j, 2^k)
```

Bottom-up Merge Sort



```
procedure: mergesort ($a_1$, $a_2$, …, $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
        for (j=0, j<n, j+=2^(k+1)), do
                merge(list, final, j, 2^k)
```

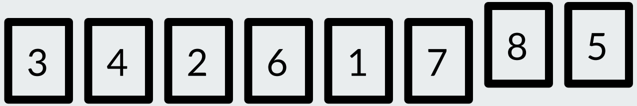
Bottom-up Merge Sort



3 4 2 6 7 1 8 5

```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
          for (j=0, j<n, j+=2^(k+1)), do
                    merge(list, final, j, 2^k)
```

Bottom-up Merge Sort



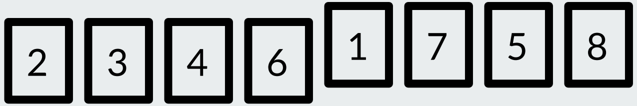
```
procedure: mergesort ($a_1, a_2, …, a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
          for (j=0, j<n, j+=2^(k+1)), do
                    merge(list, final, j, 2^k)
```

Bottom-up Merge Sort

3	4	2	6	1	7	5	8

```
procedure: mergesort ($a_1$, $a_2$, …, $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
        for (j=0, j<n, j+=2^(k+1)), do
                merge(list, final, j, 2^k)
```

Bottom-up Merge Sort



```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
    for (j=0, j<n, j+=2^(k+1)), do
        merge(list, final, j, 2^k)
```

Bottom-up Merge Sort

2	3	4	6	1	5	7	8

```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
        for (j=0, j<n, j+=2^(k+1)), do
                merge(list, final, j, 2^k)
```

Bottom-up Merge Sort

1	2	3	4	5	6	7	8

```
procedure: mergesort ($a_1$, $a_2$, ..., $a_n$: array of integers)
list := input array
final := empty array
for (k=0, k<log2(length of list), k++), do
        for (j=0, j<n, j+=2^(k+1)), do
                merge(list, final, j, 2^k)
```