CS062
DATA STRUCTURES AND ADVANCED PROGRAMMING

18: Mergesort

Alexandra Papoutsaki
LECTURES

Mark Kampe
LABS
Lecture 18: Mergesort

- Mergesort
MERGESORT

Basics

- Divide array into two halves.
- Recursively sort each half.
- Merge the two halves
2.2 Merging Demo
Merging

```java
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++) {
        if (i > mid) // ran out of elements in the left subarray
            a[k] = aux[j++];
        else if (j > hi) // ran out of elements in the right subarray
            a[k] = aux[i++];
        else if (less(aux[j], aux[i]))
            a[k] = aux[j++];
        else
            a[k] = aux[i++];
    }
}
```
Merging Example - copying to auxiliary array

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

lo mid hi
Merging Example - k=0

<table>
<thead>
<tr>
<th></th>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo</td>
<td>mid</td>
<td>hi</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>i</td>
</tr>
</tbody>
</table>

```
0 1 2 3 4 5 6 7 8 9
```

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
<td>L</td>
<td>O</td>
<td>R</td>
<td>H</td>
<td>I</td>
<td>M</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
case: aux[i]<aux[j]
a[0]=aux[0]
i++;
```
Merging Example - k=1

Array aux
<table>
<thead>
<tr>
<th>A</th>
<th>G</th>
<th>L</th>
<th>O</th>
<th>R</th>
<th>H</th>
<th>I</th>
<th>M</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

Array a
<table>
<thead>
<tr>
<th>A</th>
<th>G</th>
<th>L</th>
<th>O</th>
<th>R</th>
<th>H</th>
<th>I</th>
<th>M</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

\[\text{lo} \quad \text{mid} \quad \text{hi} \quad i \quad j \quad k\]

case: \(\text{aux}[i] < \text{aux}[j]\)
\(a[1] = \text{aux}[1]\)
\(i++;\)
Merging Example - \( k=2 \)

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A G L O R H I M S T</td>
<td>A G H O R H I M S T</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

lo mid hi

i j
k

case: \( aux[i] > aux[j] \)
\( j++; \)
Merging Example - k=3

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A G L O R H I M S T</td>
<td>A G H I R H I M S T</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

lo | mid | hi |
---|-----|----|
  |     |    |

i | j |
---|---|

k

case: aux[i] > aux[j]
Merging Example - k=4

Array aux

<table>
<thead>
<tr>
<th>A</th>
<th>G</th>
<th>L</th>
<th>O</th>
<th>R</th>
<th>H</th>
<th>I</th>
<th>M</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

lo | mid | hi

| i | j |

| k |

case: aux[i] < aux[j]
i++;
### Merging Example - k=5

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A G L O R H I M S T</td>
<td>A G H I L M I M S T</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>lo</td>
<td>mid</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>

**Case: aux[i] > aux[j]**

```plaintext
j++;
```
Merging Example - k=6

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A G L O R H I M S T</td>
<td>A G H I L M O M S T</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
</tbody>
</table>

lo  mid  hi

 i   j

k

case: aux[i]<aux[j]
i++;
### Merging Example - k=7

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>L</th>
<th>O</th>
<th>R</th>
<th>H</th>
<th>I</th>
<th>M</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>lo</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>mid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>L</th>
<th>M</th>
<th>O</th>
<th>R</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

**case:** aux[i] < aux[j]

i++;
Merging Example - k=8

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A G L O R H I M S T</td>
<td>A G H I L M O R S T</td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>lo mid hi</td>
<td></td>
</tr>
<tr>
<td>i j k</td>
<td></td>
</tr>
</tbody>
</table>

case: i > mid
a[8] = aux[8]
j++;
## MERGESORT

**Merging Example - k=9**

<table>
<thead>
<tr>
<th>Array aux</th>
<th>Array a</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>G</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>lo</td>
<td>mid</td>
</tr>
<tr>
<td>i</td>
<td>j</td>
</tr>
</tbody>
</table>

**case:** i > mid


j++;
Practice time

How many calls does `merge()` make to `less()` in order to merge two already sorted subarrays, each of length $n/2$ into a sorted array of length $n$?

A. $\sim 1/4n$ to $\sim 1/2n$
B. $\sim 1/2n$
C. $\sim 1/2n$ to $n$
D. $\sim n$
Answer

How many calls does \texttt{merge()} make to \texttt{less()} in order to merge two already sorted subarrays, each of length \( n/2 \) into a sorted array of length \( n \)?

C. \( \sim 1/2n \) to \( n \)

That is \( O(n) \)
Mergesort - the quintessential example of divide-and-conquer

```java
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi) {
    if (hi <= lo)
        return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}

public static void sort(Comparable[] a) {
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length - 1);
}
```
Example: Look at board for a simpler case
Practice time

Which of the following subarray lengths will occur when running merge sort on an array of length 10?

A. { 1, 2, 3, 5, 10 }  
B. { 2, 4, 6, 8, 10 }  
C. { 1, 2, 5, 10 }  
D. { 1,2,3,4,5,10}
Answer

Which of the following subarray lengths will occur when running mergesort on an array of length 10?

A. { 1, 2, 3, 5, 10 }
Good algorithms are better than supercomputers

- Your laptop executes $10^8$ comparisons per second
- A supercomputer executes $10^{12}$ comparisons per second

<table>
<thead>
<tr>
<th>Computer</th>
<th>Insertion sort</th>
<th>Mergesort</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Thousand inputs</td>
<td>Million inputs</td>
</tr>
<tr>
<td>Home</td>
<td>Instant</td>
<td>2 hours</td>
</tr>
<tr>
<td>Supercomputer</td>
<td>Instant</td>
<td>1 second</td>
</tr>
</tbody>
</table>
Analysis of comparisons

- The number of comparisons $C(n)$ to mergesort an array of length $n$ satisfies the recurrence:
  $C(n) \leq C(n/2) + C(n/2) + n - 1$ for $n > 1$, with $C(1) = 0$

- We will simplify the problem by assuming that $n$ is a power of 2 and solve this simple recurrence:
  $T(n) = 2T(n/2) + n$
Mergesort uses $\leq n \log n$ compares to sort an array of length $n$. 

https://cs.stackexchange.com/questions/64060/mergesort-recursion-tree-depth-logs
Analysis of array accesses

- As with number of comparisons, mergesort uses \( \leq n \log n \) array accesses to sort an array of size \( n \).
Any algorithm with the same structure takes $n \log n$ time

```java
public static void f(int n) {
    if (n == 0)
        return;
    f(n/2);
    f(n/2);
    linear(n);
    f(n/2);
    f(n/2);
    linear(n);
}
```
Mergesort analysis of memory

- Auxiliary memory is proportional to $n$, as $\text{aux}[]$ needs to be of length $n$ for the last merge.

- At its simplest form merge sort is not an in-place algorithm.

- There are modifications for halting the size of the auxiliary array but in-place merge is very hard.
Stability

A sorting algorithm is **stable** if it preserves the relative order of equal keys. For example:

- **Input:** $CA_1BA_2A_3$
- **Output:** $A_3A_1A_2BC$
- This sorting algorithm is not stable
Mergesort is stable

- Look into `merge()`, if equal keys it takes them from the left subarray.
- So is insertion sort, but not selection sort.
Mergesort practical improvements

- Use insertion sort for small subarrays.
  - Too much overhead for tiny subarrays.
  - Cutoff to insertion sort usually around 10 items.
  - Improvement of 10-15% in running time.
- Stop if already sorted.
  - Is largest item in first half smaller or equal the smallest item in second half? In sort method
    - if (!less(a[mid+1], a[mid])) return;
- Eliminate the copy the auxiliary array by saving time (not space).
  - sort (aux, a, lo, mid);
  - sort (aux, a, mid+1, hi);
  - merge(a, aux, lo, mid, hi);
- Java’s default sort algorithm is a mergesort with all the above tricks with cutoff at 7.
## MERGESORT

### Sorting: the story so far

<table>
<thead>
<tr>
<th>Method</th>
<th>In place</th>
<th>Stable</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>X</td>
<td></td>
<td>$1/2n^2$</td>
<td>$1/2n^2$</td>
<td>$1/2n^2$</td>
<td>$n$ exchanges</td>
</tr>
<tr>
<td>Insertion</td>
<td>X</td>
<td>X</td>
<td>$n$</td>
<td>$1/4n^2$</td>
<td>$1/2n^2$</td>
<td>Use for small arrays or partially ordered</td>
</tr>
<tr>
<td>Merge sort</td>
<td>X</td>
<td></td>
<td>$1/2n \log n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>Guaranteed performance; stable</td>
</tr>
</tbody>
</table>
Lecture 18: Mergesort

- Mergesort
Readings:

- Textbook:
  - Chapter 2.2 (pages 270-277)

- Website:
  - Mergesort: https://algs4.cs.princeton.edu/22mergesort/
  - Code: https://algs4.cs.princeton.edu/22mergesort/Merge.java.html

Practice Problems:

- 2.2.1-2.2.2, 2.2.11