COMPSCI 105

Principles of Computer Science

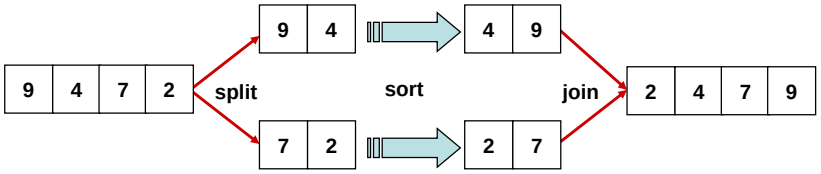
MergeSort

Merge sort

Divide and Conquer

Imagine a set of cards

- Split the deck in half
- Get two friends to sort each half into order (recursive case)
- Merge the two sorted halves into correct order



Instructions

MergeSort

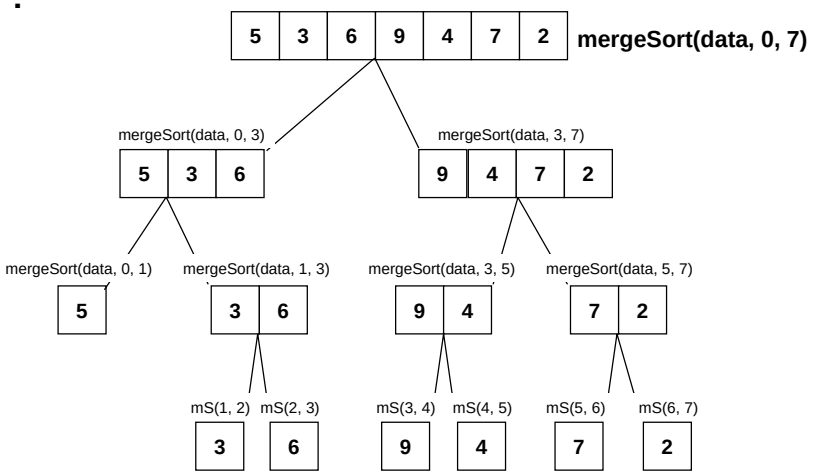
- Accept a pile of cards
- If you were given a single card, then return it, else
  - Divide pile in half
  - Give first half to neighbour and wait for it to be returned
  - Give second half to neighbour and wait for it to be returned
  - Give both halves to the Merge person and wait for it to be returned
  - Return the sorted pile to the person that gave it to you

Merge

- Accept two piles of cards
- Repeat until all elements are checked.
  - Compare top of each pile
  - Take largest and add to bottom of new pile

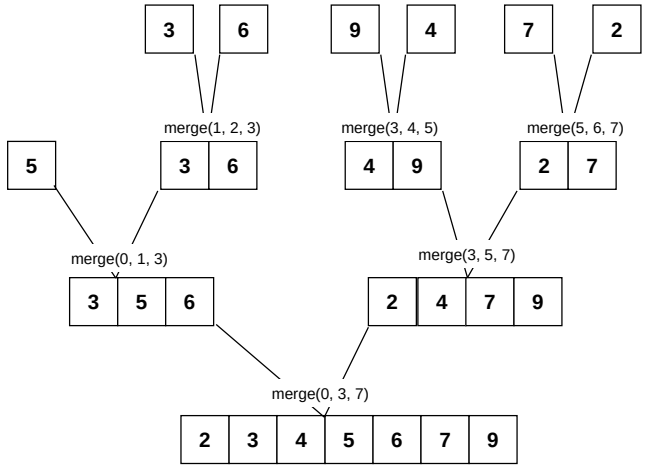
Mergesort

.



Mergesort

.



Pseudocode for Merge sort

MergeSort (start, end)

```
if number of elements to sort > 1
    split in half (find middle)
    sort left half (start, middle)
    sort right half (middle, end)
    merge both halves (start, middle, end)
```

Merge

```
create temporary array
for each element of temp
    compare elements from each sorted half
    assign smallest element to temp
```


Mergesort

```
public void mergesort(int[] theArray, int first, int last) {
    if (first < last) {
       int mid = (first + last)/2;
       mergesort(theArray, first, mid);
       mergesort(theArray, mid+1, last);
       merge(theArray, first, mid, last);
    }
} // end mergesort
```


Merge

```
private void merge(int[] theArray,  int first, int mid, int last) {
    int[] tempArray = new int[theArray.length];
    int i = first, j = mid + 1, index = i;

    while ((i <= mid) && (j <= last)) {
        if (theArray[i] < theArray[j]) {
            tempArray[index] = theArray[i];
            i++;
        } else {
        tempArray[index] = theArray[j];
            j++;
        }
        index++;
    }

    while (i <= mid || j <= last) {
        if (i <= mid ) {
            tempArray[index] = theArray[i];
        i++;
        } else {
            tempArray[index] = theArray[j];
            j++;
        }
        index++;
    }

    for (index = first; index <= last; index++) {
        theArray[index] = tempArray[index];
    }
}
```

Merge

Alternate code for merge

```
private void merge(int[] theArray,  int first, int mid, int last) {
    int[] tempArray = new int[theArray.length];
    int i = first, j = mid + 1, index;

    for (index = first; index <= last; index++) {
        if (j > last || i <= mid && (theArray[i] < theArray[j])) {
            tempArray[index] = theArray[i++];
        } else {
            tempArray[index] = theArray[j++];
        }
    }

    for (index = first; index <= last; index++) {
        theArray[index] = tempArray[index];
    }
}
```


Exercise

Perform a code trace for the code:

```
int[] a = {5,1,8,3,6};
mergeSort(a, 0, a.length-1);
```


Order Analysis

Divide and Conquer

- Split problem into 2 halves.
- Sort each half
- Merge solutions
- Better performance than $O(n^2)$?

Imagine Sorting 32 cards using selection sort.

- Time ≈ 1024 (i.e. 32^2)

Compare with sorting 16 cards twice.

- Time ≈ 512 (i.e. $16^2 + 16^2$)

Must add the cost of merging solutions

Compare with sorting 8 cards four times

- Time ≈ 256 (i.e. $8^2 + 8^2 + 8^2 + 8^2$)