6 Sorting Algorithms

Topics

- Sorting
- Insertion Sort
- Selection Sort
- Merge Sort
  - Divide-and-Conquer Paradigm
- Quicksort
  - Divide-and-Conquer Paradigm

Sorting

- Arranging elements in a particular order
- Used in a wide range of applications
- Several sorting algorithms have been invented because the task is so fundamental and also frequently used

Insertion Sort

- One of the simplest algorithms
- Quite intuitive and is analagous to a way of arranging a collection of cards
  - Goal: Arrange a standard deck of cards from lowest to highest rank
  - Given: cards, table 1, table 2
  - Initially: Unsorted cards are neatly placed on table 1
  - Technique: Sorted cards will be positioned on table 2
  - Pick first card from table 1, compare this with those on table 2 and place this card in its proper position on table 2
  - Repeat this step until all cards are placed on table 2

Insertion Sort: Algorithm

- Divides data elements to be sorted into two groups
  - Unsorted section
  - Sorted section

- Repeated the following steps until no elements are left in the unsorted part of the array
  - First available element is selected from the unsorted section of the array
  - Place selected element in its proper position in the sorted section of the array

Insertion Sort: Algorithm

```
void insertionSort(Object array[], int startIdx,
                           int endIdx) {
    for (int i = startIdx; i < endIdx; i++) {
        int k = i;
        for (int j = i + 1; j < endIdx; j++) {
            if (((Comparable) array[k]).compareTo(
                                        array[j])>0) {
                k = j;
            }
        }
        swap(array[i], array[k]);
    }
}
```

Insertion Sort: Example

Given	1^{st} Pass	2^{nd} Pass	3^{rd} Pass	4^{th} Pass
Mango	Mango	Apple	Apple	Apple
Apple	Apple	Mango	Mango	Banana
Peach	Peach	Peach	Orange	Mango
Orange	Orange	Orange	Peach	Orange
Banana	Banana	Banana	Banana	Peach

Selection Sort

- Another simple sorting algorithm
  - Intuitive and easy to implement

- Also analogous to another way of arranging cards
  - Goal: Arrange cards in ascending order
  - Given: cards, table
  - Initially: Cards neatly arranged are on the table
  - Check the rank of each card and select card with the lowest rank
  - Exchange position of this card with that of the first card on the table
  - Find the card with the lowest rank among the remaining cards
  - Swap the newly selected card with the card in the second position
  - Continue in this manner until card in the second to the last position on the table is challenged and possibly swapped with last card

Selection Sort: Algorithm

- Select the element with the lowest value
- Swap chosen element with the element in the *i*th position
  - i starts from 1 to n
  - where n is the total number of elements minus 1

Selection Sort: Algorithm

```
void selectionSort(Object array[], int startIdx,
                        int endIdx) {
    int min;
    for (int i = startIdx; i < endIdx; i++) {
        min = i;
        for (int j = i + 1; j < endIdx; j++) {
            if (((Comparable) array[min]).compareTo(
                                            array[j])>0) {
                min = j;
            }
        }
        swap(array[min], array[i]);
    }
}
```

Selection Sort: Example

Given	***1st Pass***	***2nd Pass***	***3rd Pass***	***4th Pass***
Maricar	Hannah	Hannah	Hannah	Hannah
Vanessa	*Vanessa*	Margaux	Margaux	Margaux
Margaux	*Margaux*	*Vanessa*	Maricar	Maricar
Hannah	Maricar	*Maricar*	*Vanessa*	Rowena
Rowena	Rowena	Rowena	*Rowena*	Vanessa

Merge Sort: Divide-and-Conquer Paradigm

- Use recursion to solve a given problem
  - Original problem is split into subproblems
  - Solutions to subproblems lead to the solution of the main problem

- Three steps:
  - Divide
    - Divide the main problem into subproblems
  - Conquer
    - Conquer the subproblems by recursively solving them
    - If suproblems are simple and small enough, solve in a straightforward manner
  - Combine
    - Combine the solutions to suproblems, leading to solution of main problem

Merge Sort: Algorithm

- Uses the divide-and-conquer approach
  - Divide
    - Divide the sequence of data elements into two halves
  - Conquer
    - Conquer each half by recursively calling the *mergeSort* method
  - Combine
    - Combine or merge the two halves recursively to come up with the sorted sequence

- Recursion stops when the half to be sorted has exactly one element
  - Already sorted

Merge Sort: Algorithm

```
void mergeSort(Object array[], int startIdx,
               int endIdx) {
   if (array.length != 1) {
      Divide the array into two halves,
                                leftArr and rightArr
      mergeSort(leftArr, startIdx, midIdx);
      mergeSort(rightArr, midIdx+1, endIdx);
      combine(leftArr, rightArr);
   }
}
```

Merge Sort: Example

Given:

7	2	5	6

Divide given array into two:

LeftArr *RightArr*

7	2		5	6

Divide *LeftArr* of given into two:

LeftArr *RightArr*

7		2

Combine

2	7

Divide *RightArr* of given into two:

LeftArr *RightArr*

5		6

Combine

5	6

Combine *LeftArr* and *RightArr* of the given.

2	5	6	7

Quicksort: Algorithm

- Invented by C.A.R. Hoare
- Also based on the divide-and-conquer paradigm
  - Divide
    - Partition array into two subarrays A[p...q-1] and A[q+1...r] such that each element in A[p...q-1] is less than or equal to A[q] and each element in A[q+1...r] is greater than or equal to A[q]
    - A[q] is called the pivot
    - Computation of q is part of the partitioning procedure
  - Conquer
    - Sort the subarrays by recursively calling the *quickSort* method
  - No more "Combine" phase
    - Subarrays are sorted in place

Quicksort: Algorithm

```
void quickSort(Object array[], int leftIdx,
                    int rightIdx) {
    int pivotIdx;
    /* Termination condition! */
    if (rightIdx > leftIdx) {
        pivotIdx = partition(array, leftIdx, rightIdx);
        quickSort(array, leftIdx, pivotIdx-1);
        quickSort(array, pivotIdx+1, rightIdx);
    }
}
```

Quicksort: Example

Given array:

3	1	4	1	5	9	2	6	5	3	5	8

Choose the first element to be the pivot = 3.

3	1	4	1	5	9	2	6	5	3	5	8

Initialize left to point to the second element and right to point to the last element.

	left										right
3	1	4	1	5	9	2	6	5	3	5	8

Move the left pointer to the right direction until we find a value larger than the pivot. Move the right pointer to the left direction until we fina value not larger than the pivot.

		left							right		
3	1	<u>4</u>	1	5	9	2	6	5	<u>3</u>	5	8

Swap elements referred to by the left and right pointers.

		left							right		
3	1	3	1	5	9	2	6	5	4	5	8

Quicksort: Example

left right

3	1	3	1	5	9	2	6	5	4	5	8

Move the left and right pointers again.

left right

3	1	3	1	5	9	2	6	5	4	5	8

Swap elements.

left right

3	1	3	1	2	9	5	6	5	4	5	8

Move the left and right pointers again.

right left

3	1	3	1	2	9	5	6	5	4	5	8

Observe that the left and right pointers have crossed such that right < left. In this case, swap pivot with right.

pivot

2	1	3	1	**3**	9	5	6	5	4	5	8

Pivoting is now complete. Recursively sort subarrays on each side of the pivot.

Summary

- Simple Sorting Techniques
  - Insertion Sort
  - Selection Sort

- Divide-and-Conquer Paradigm
  - Merge Sort
  - Quicksort