CS 2420: Sorting Algorithms

Dr. Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Outline

- In this topic, we will introduce sorting, including:
  - Definitions
  - Assumptions
  - *In-place* sorting
  - Sorting techniques and strategies
  - Overview of run times

- Lower bound on run times
- Define inversions and use this as a measure of *unsortedness*

Definition

Sorting is the process of:

- Taking a list of objects which could be stored in a linear order

  $$(a_0, a_1, ..., a_{n-1})$$

  e.*g*., numbers, and returning an reordering

  $$(a'_0, a'_1, ..., a'_{n-1})$$

  such that

  $$a'_0 \le a'_1 \le \cdots \le a'_{n-1}$$

The conversion of an Abstract List into an Abstract Sorted List

Definition

In these topics, we will assume that:

- Arrays are to be used for both input and output,
- We will focus on sorting objects and leave the more general case of sorting records based on one or more fields as an implementation detail

In-place Sorting

Sorting algorithms may be performed *in-place*, that is, with the allocation of at most $\Theta(1)$ additional memory (*e.g.*, fixed number of local variables)

- Some definitions of *in place* as using $\mathbf{o}(n)$ memory

Other sorting algorithms require the allocation of second array of equal size

- Requires $\Theta(n)$ additional memory

We will prefer in-place sorting algorithms

Run-time

The run time of the sorting algorithms we will look at fall into one of three categories:

$$\Theta(n) \quad \Theta(n \ln(n)) \quad \mathbf{O}(n^2)$$

We will examine average- and worst-case scenarios for each algorithm

The run-time may change significantly based on the scenario

Run-time

We will review the more traditional $\mathbf{O}(n^2)$ sorting algorithms:

- Insertion sort, Bubble sort, Selection sort

Some of the faster $\Theta(n \ln(n))$ sorting algorithms:

- Heap sort, Quicksort, and Merge sort

And linear-time sorting algorithms

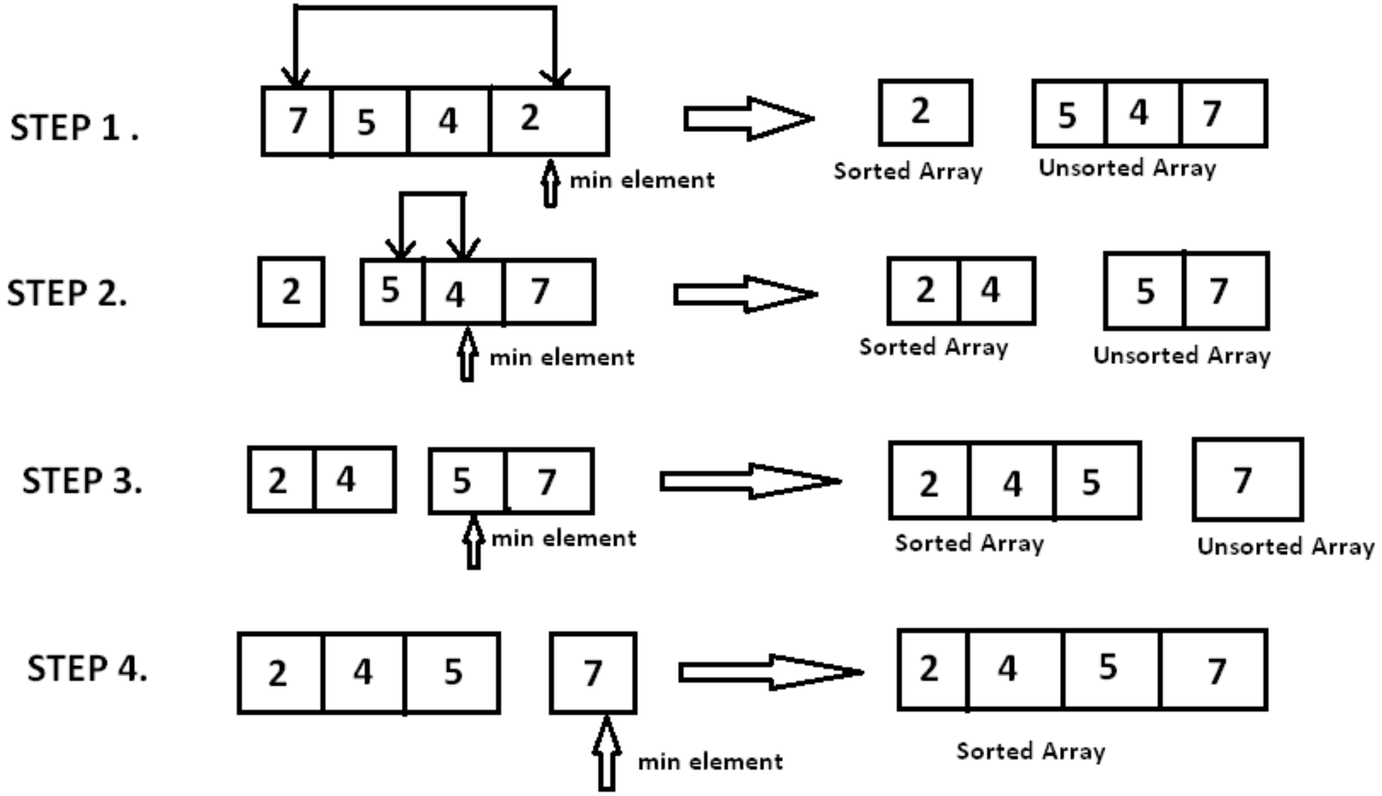
- Bucket sort and Radix sort
- We must make assumptions about the data

Sorting

- The most fundamental algorithm in all subjects …
- Goal: puts elements in a certain order
  - Increasing order: 1, 2, 5, 6, 8, 90, 123
  - Decreasing order: 123, 90, 8, 6, 5, 2, 1
- Many algorithm paradigms
  - Bubble sort
  - Selection sort
  - Merge sort
  - Qsort
  - …
- Today, new sorting algorithms are being invented

Selection Sort

- Two loops
  - Outer loop to repeat n-1 times
  - Inner loop to find the minimum element



Selection Sort Implementation

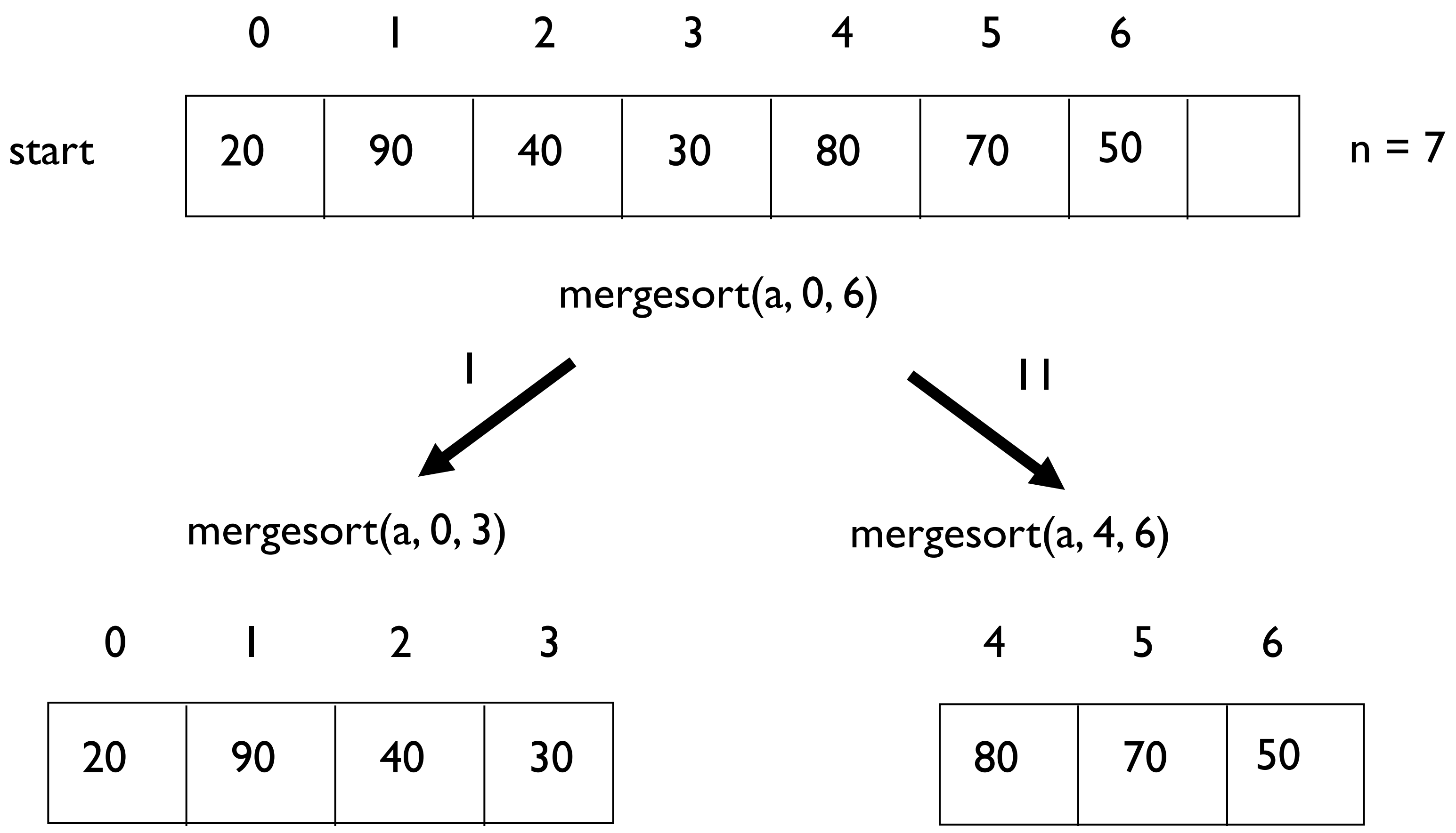
```cpp
void brute_force(std::vector<int>& D, int beg, int end) {
  int max = std::numeric_limits<int>::min();
  for (int i = beg; i < end; ++i) {
    int min_v = D[i];
    int min_j = i;
    for (int j = i+1; j < end; ++j) {
      if(D[j] < min_v) {
        min_v = D[j];
        min_j = j;
      }
    }
    std::swap(D[i], D[min_j]);
  }
}
```

Time complexity $O(N^2)$

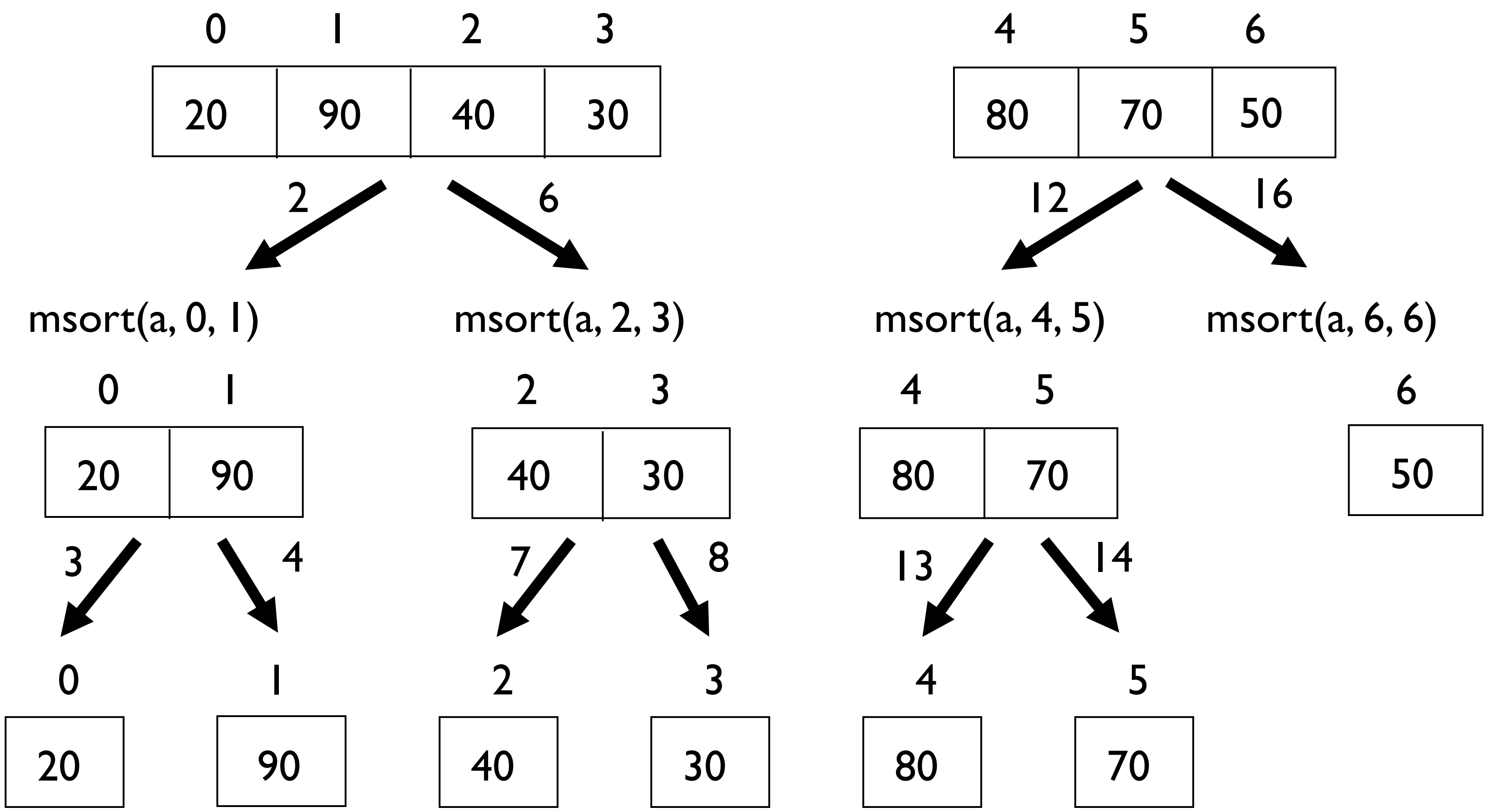
Using Divide and Conquer: Merge Sort

- **Divide**: If S has at two or more elements (nothing needs to be done if S has zero or one elements), remove all the elements from S and put them into two sequences, S_L and S_R, each containing about half of the elements of S. (i.e. S_L contains the first $\lceil n/2 \rceil$ elements and S_R contains the remaining $\lfloor n/2 \rfloor$ elements.
- **Recurse**: Recursively sort sequences S_L and S_R.
- **Conquer**: Put back the elements into S by merging the two sorted sequences S_L and S_R into a sorted sequence.

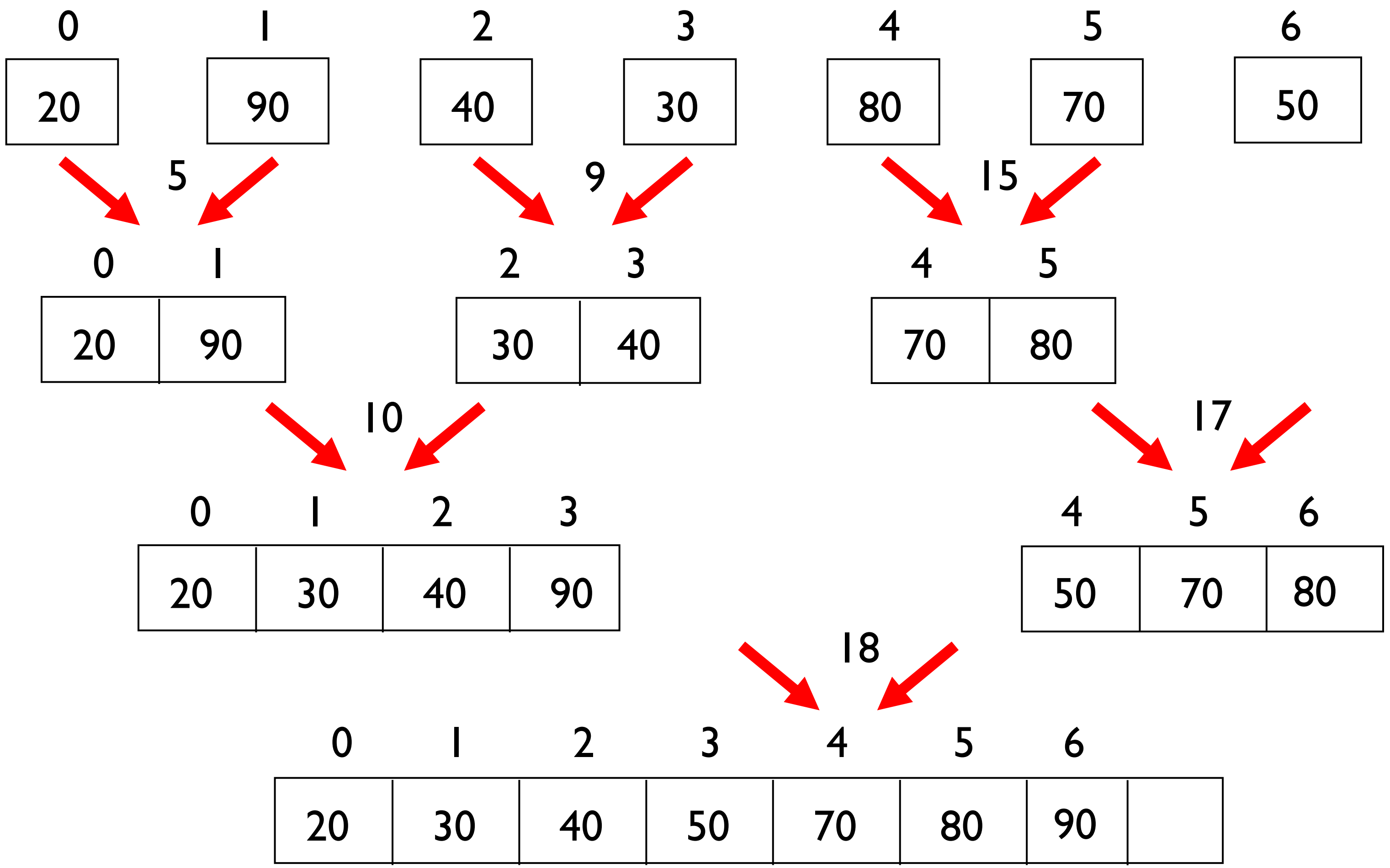
Illustration



Illustration

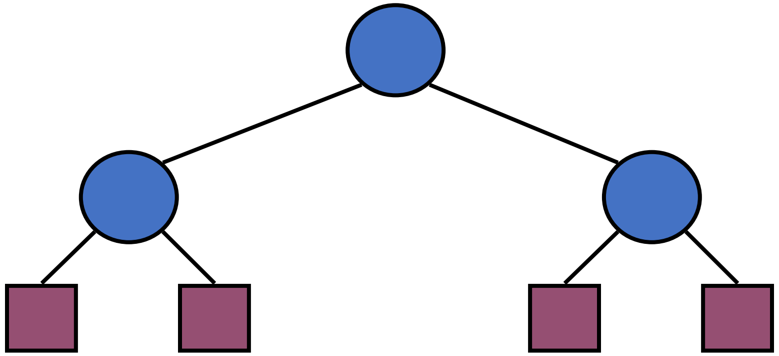


Illustration



Merge Sort Complexity

- Run Time Analysis
  - At each level in the binary tree created for Merge Sort, there are n elements, with O(1) time spent at each element
  - O(n) running time for processing one level
  - The height of the tree is O(log n)



- Therefore, the time complexity is O(nlog n)

Summary

Introduction to sorting, including:

- Assumptions
- In-place sorting ($\mathbf{O}(1)$ additional memory)
- Sorting techniques
  - insertion, exchanging, selection, merging, distribution
- Run-time classification: $\mathbf{O}(n)$ $\mathbf{O}(n \ln(n))$ $\mathbf{O}(n^2)$

Overview of proof that a general sorting algorithm must be $\boldsymbol{\Omega}(n \ln(n))$

We will have out second Midterm next Week (starting on 11/22)