Lecture 12: Array and Vector – Part II

Class page: https://github.com/tsung-wei-huang/cs1410-40

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

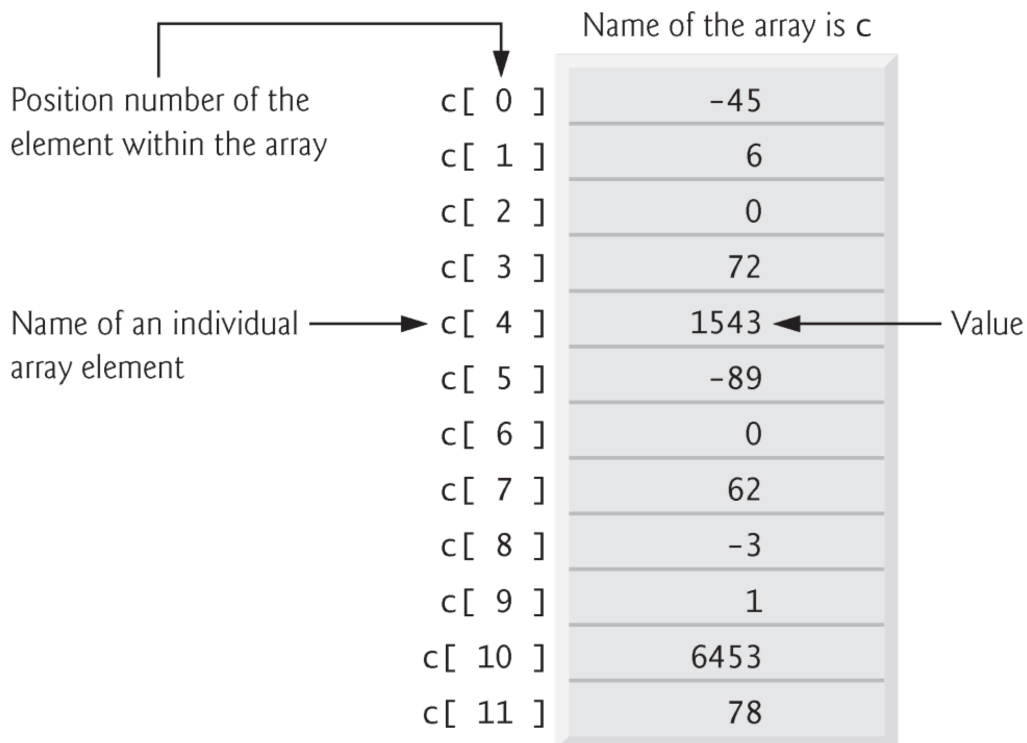
Announcement

- **First midterm on 10/12 starting at the class time**
  - Concept + Programming questions whatever we've covered
  - Due on 23:59 PM 10/18 (Sunday)
  - No lectures or Lab on the midterm week, but
    - We will have extra office hours at the scheduled class/lab time
    - We will help clarity your questions rather than give you solutions
  - Midterm will be take-home exam
    - Free to find solutions using the internet resource
    - Free to discuss solutions with your friends
  - **NEVER JUST COPY SOLUTIONS**
    - You can't violate the rules of academic integrity
    - You are already undergraduates, be mature and responsible
    - Ultimately, it is your own knowledge, not mine

Array

- **An array is a consecutive group of memory locations that all have the same type.**
- **To refer to a particular location or element in the array, specify the name of the array and the position number of that element.**
- **The position number is formally called a subscript or index.**
  - This number specifies the number of elements from the beginning of the array.
  - A subscript must be an integer or integer expression (using any integral type).
- **The first element in every array has subscript 0 (zero) and is sometimes called the zeroth element.**

Memory Layout of Array



Declaring an Array

- ❑ **Arrays occupy space in memory.**
- ❑ **To specify the type of the elements and the number of elements required by an array use a declaration of the form:**
    - *type arrayName[arraySize];*
  - Ex: `int n[5];` → 5 integer elements
- ❑ **The compiler reserves the appropriate amount of memory.**
  - ❑ All elements are put in a continuous memory space.

Array Example

```
// Initializing an array.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int n[ 10 ]; // n is an array of 10 integers

   // initialize elements of array n to 0
   for ( int i = 0; i < 10; i++ )
      n[ i ] = 0; // set element at location i to 0

   cout << "Element" << setw( 13 ) << "Value" << endl;

   // output each array element's value
   for ( int j = 0; j < 10; j++ )
      cout << setw( 7 ) << j << setw( 13 ) << n[ j ] << endl;
} // end main
```

Array Example

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

Search an Item in an Array

- **Often it may be necessary to determine whether an array contains a value that matches a certain key value.**
  - Called searching.
- **The linear search compares each element of an array with a search key.**
  - Because the array is not in any particular order, it's just as likely that the value will be found in the first element as the last.
  - On average, therefore, the program must compare the search key with half the elements of the array.
- **To determine that a value is not in the array, the program must compare the search key to every element of the array.**

Example 1

- **Input:**
  - A std::vector vec of integer numbers
  - An integer number v
- **Output**
  - “Yes”, if v can be found in vec
  - “No”, if v cannot be found in vec

Example 2: Maximum Subarray Sum Problem

- **Given a sequence of integer number, find the largest sum of contiguous array numbers**

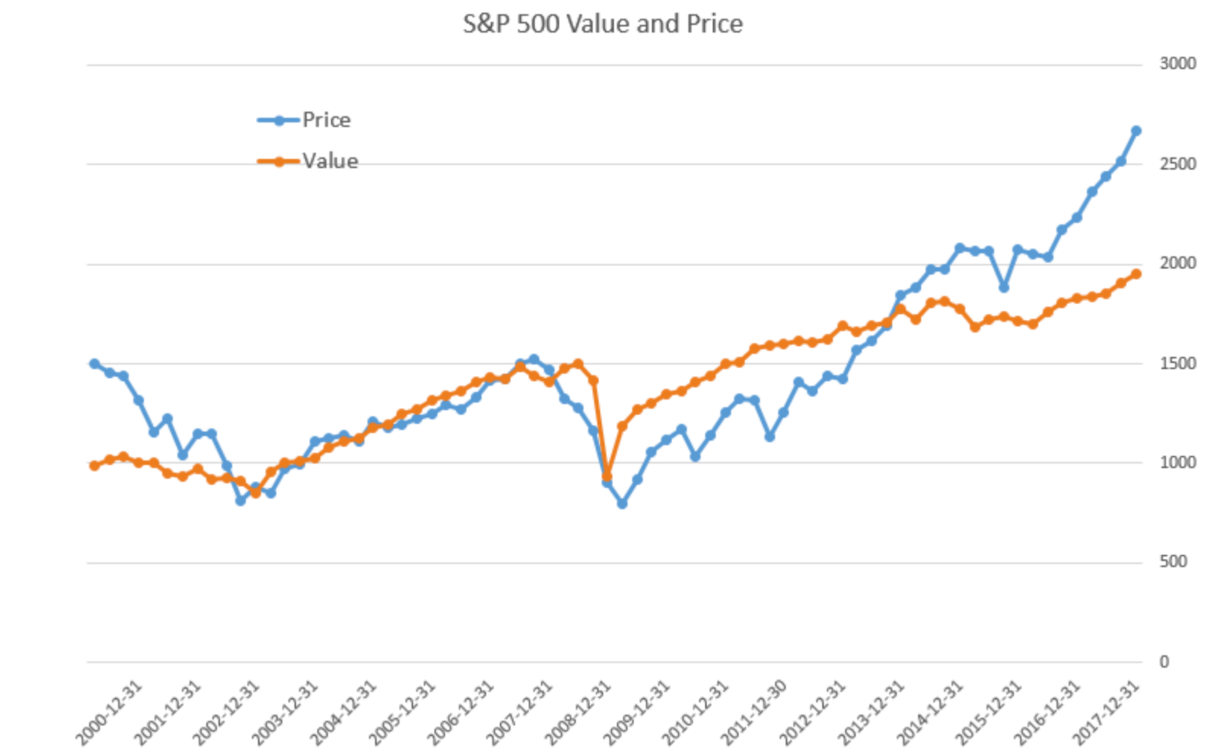
-2	-3	4	-1	-2	1	5	-3
0	1	2	3	4	5	6	7

4 + (-1) + (-2) + 1 + 5 = 7

Maximum Contiguous Array Sum is 7

Practical Application

❑ A basic routine of financial computing



Maximum subarray sum to find the optimal long-term investment

Debrief

- The solution we presented has two-level loops
  - At worst, you need N^2 iterations, where N is the size of the input array

```
int brute_force(const std::vector<int>& D, int beg, int end) {
  int max = std::numeric_limits<int>::min();
  for (int i = beg; i < end; ++i) {
    int sum = 0;
    for (int j = i; j < end; ++j) {
      sum += D[j];
      max = std::max(sum, max);
    }
  }
  return max;
}
```

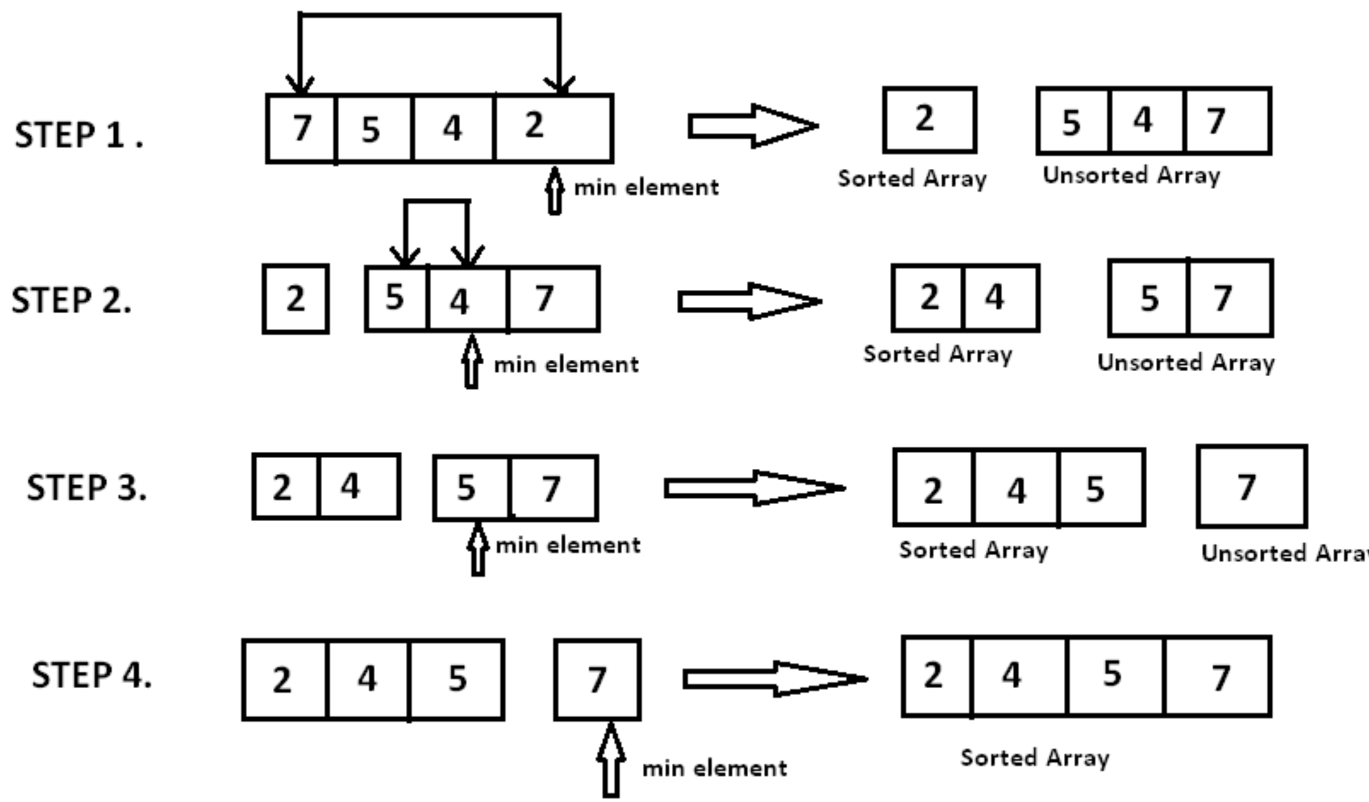
- Can we do better?

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

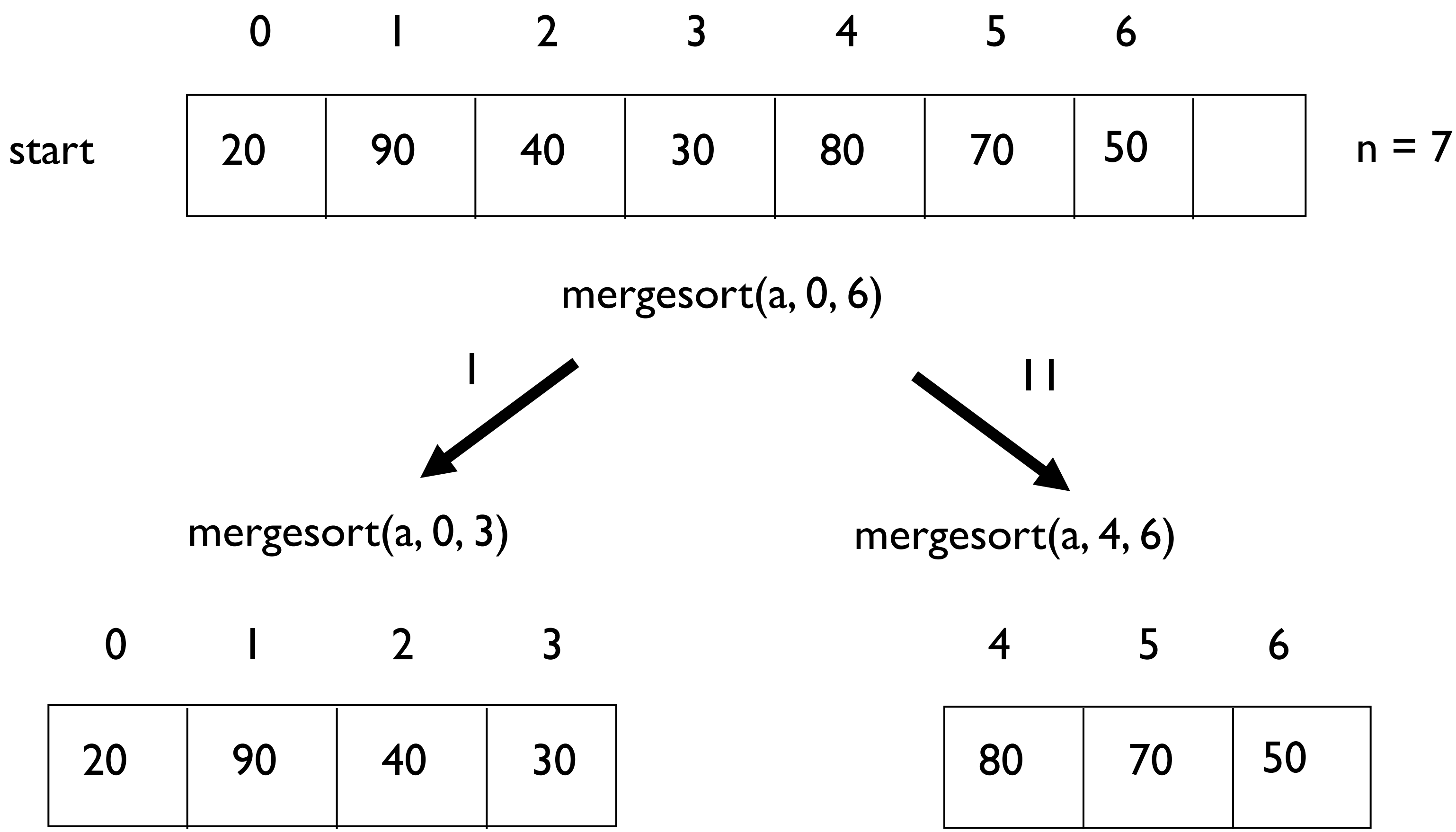
```
void brute_force(std::vector<int>& D, int beg, int end) {
  int max = std::numeric_limits<int>::min();
  for (int i = beg; i < end; ++i) {
    int min_v = D[i];
    int min_j = i;
    for (int j = i+1; j < end; ++j) {
      if(D[j] < min_v) {
        min_v = D[j];
        min_j = j;
      }
    }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

Number of iterations N^2, or $O(N^2)$

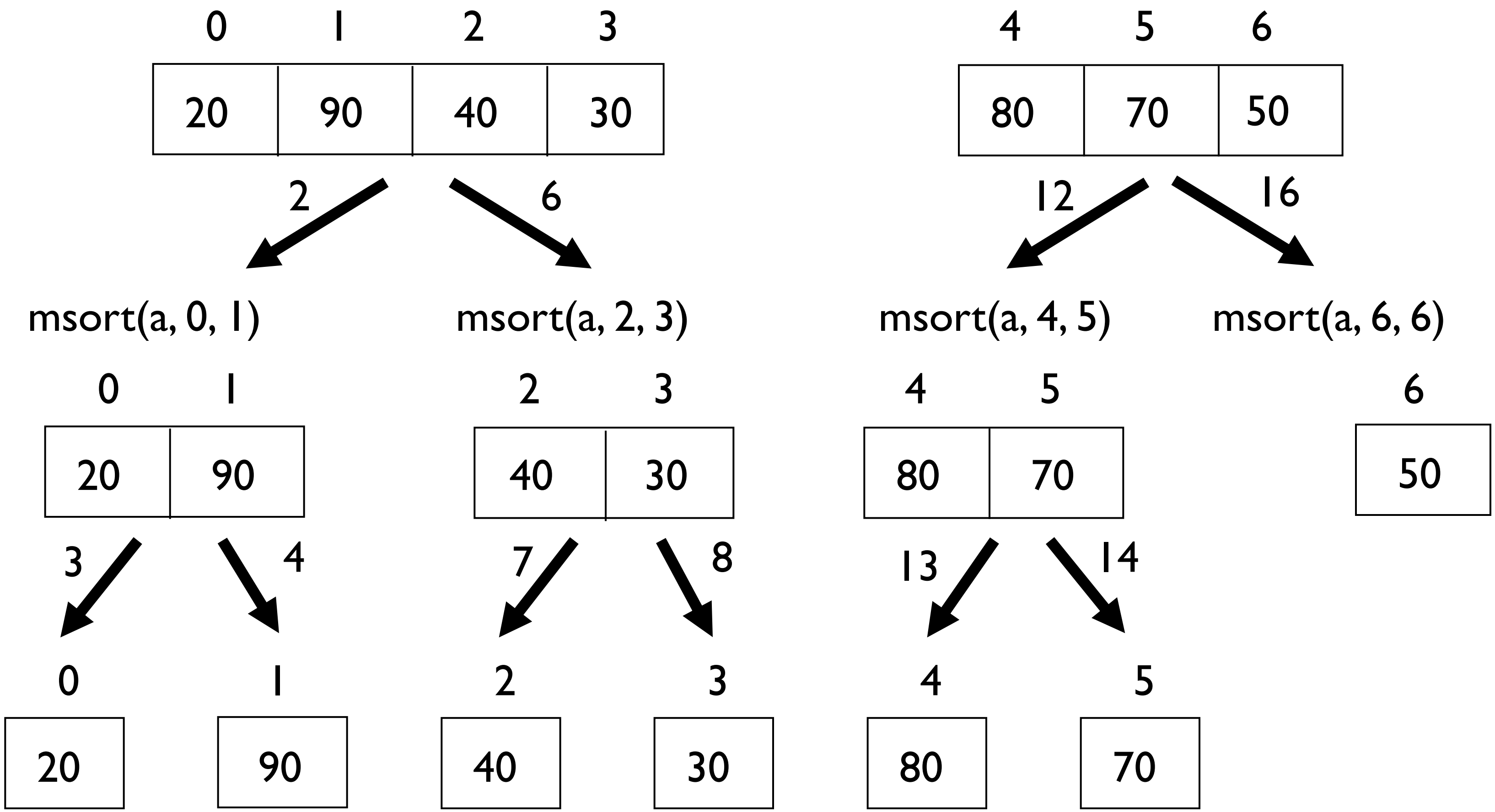
Using Divide and Conquer: Merge Sort

- **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.

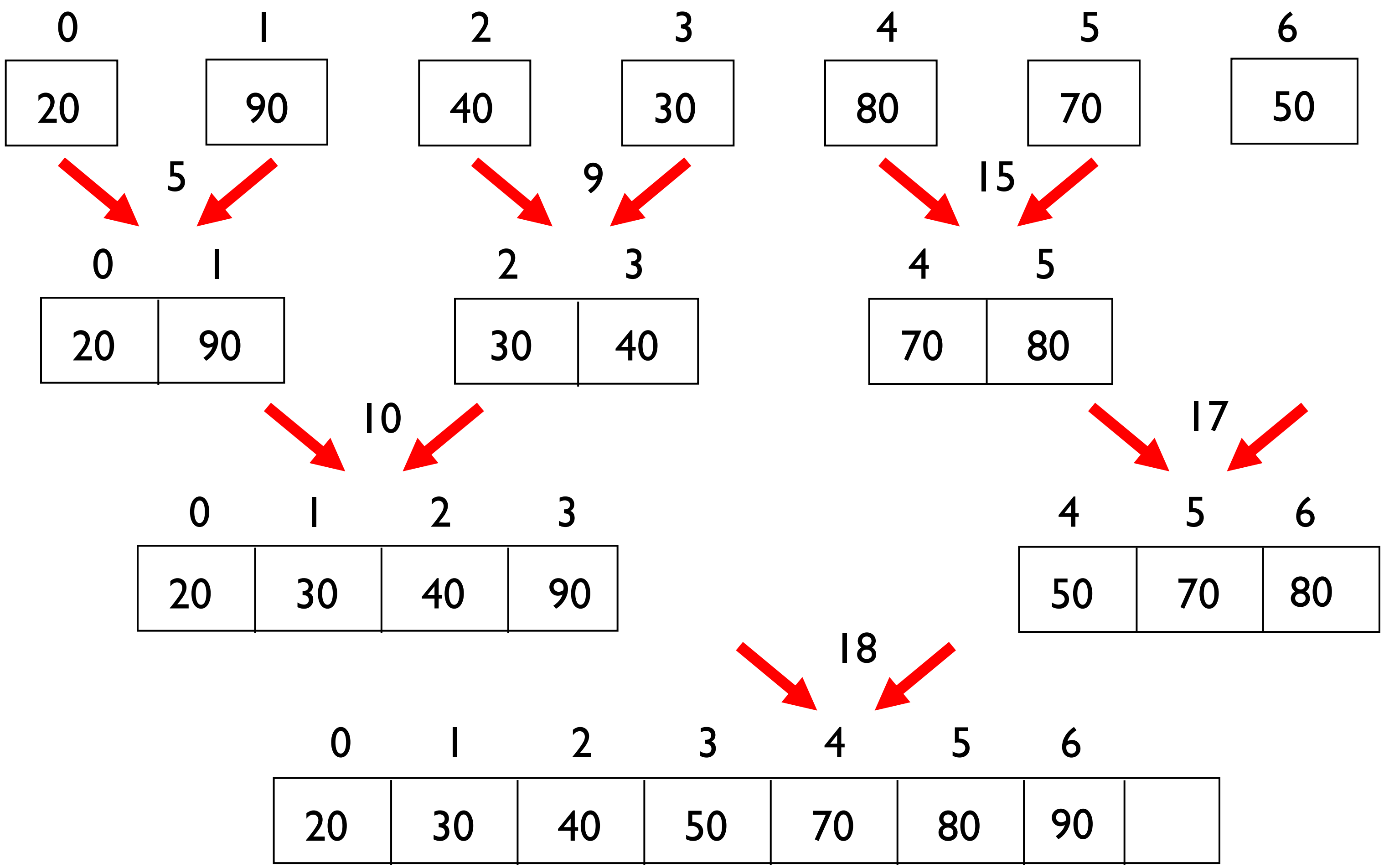
Illustration



Illustration

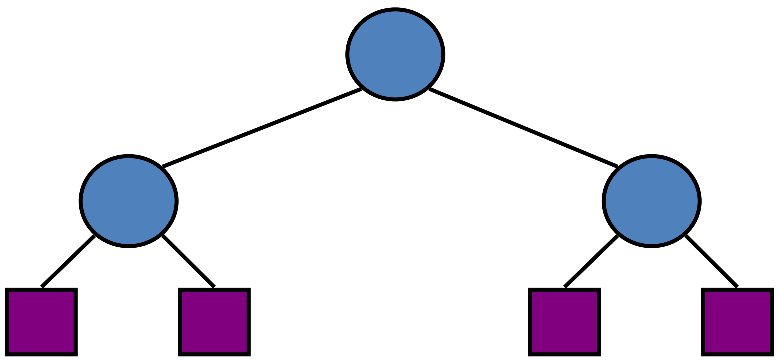


Illustration



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)**

Example 3: Merge Two Sorted Arrays

- Input: Given two sorted arrays, L and R
- Output: A merged, sorted array of L and R
- Example:
  - L = {1, 2, 5, 6, 9}
  - R = { -5, -1, 0, 4, 19, 20 }
  - Output: {-5, -1, 0, 1, 2, 4, 5, 6,, 9, 19, 20}

Summary

- **Array**
- **Maximum subarray sum**
  - Commonly used in financial computing and VLSI designs
- **Sorting**
  - Selection sort (kinda brute force: $O(N^2)$)
  - Merge sort (divide and conquer: O(nlogn))