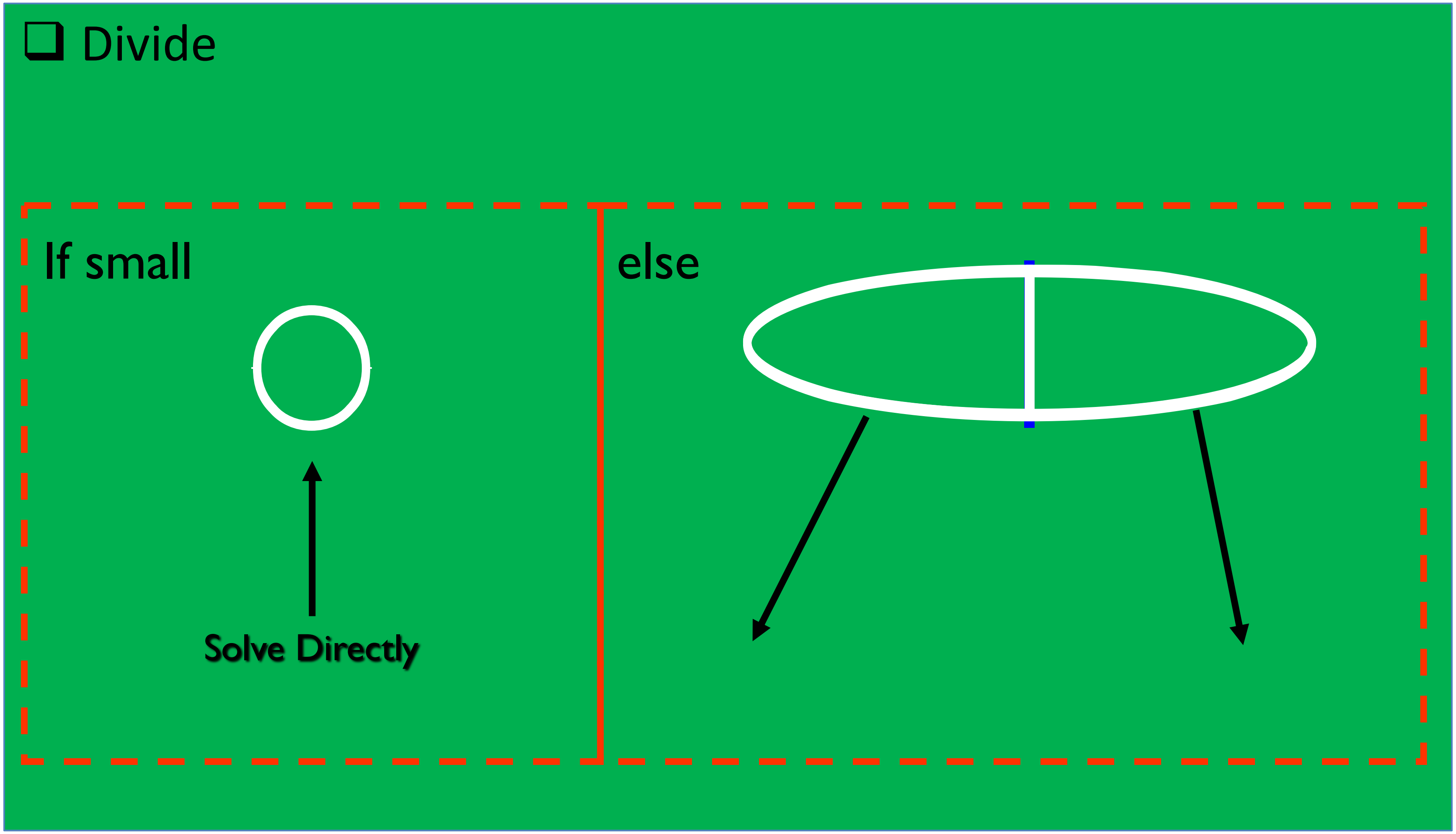
Lecture 3: Divide and Conquer Algorithms (II)

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

Divide and Conquer Recap

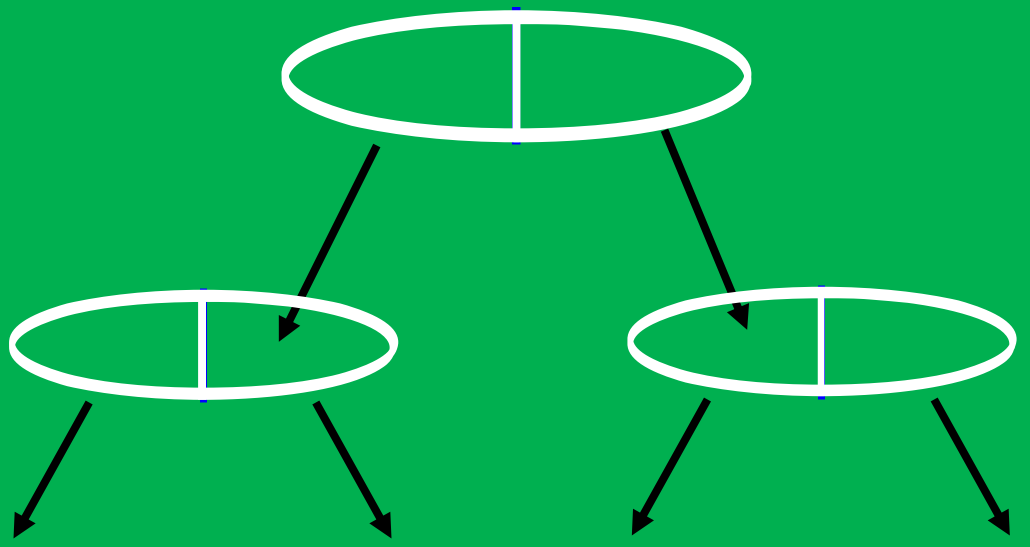
- **Divide and Conquer is a recursive algorithm**
- **Three essential steps:**
  - **Divide**: If the input size is too large to handle efficiently, divide the data into two or more disjoint subsets
  - **Recurse**: Keeps partitioning the data until a small size reasonable to solve
  - **Conquer**: Take the solutions from the subproblems and "merge" them to a solution from the division point

Visualization of Divide and Conquer



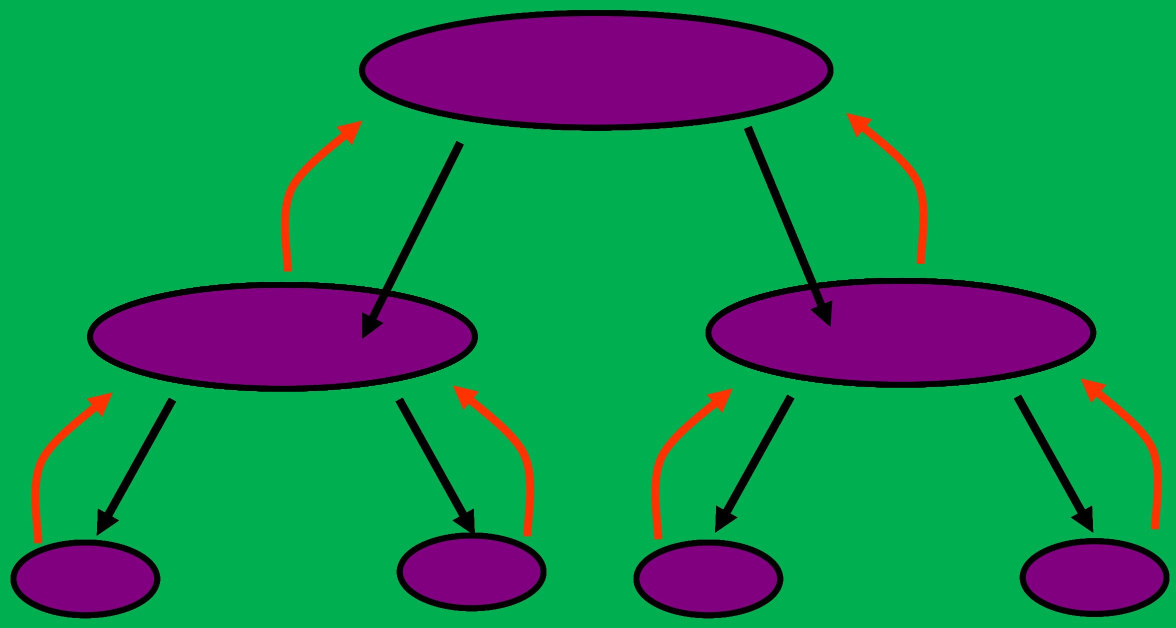
Visualization of Divide and Conquer

- Recursive



Visualization of Divide and Conquer

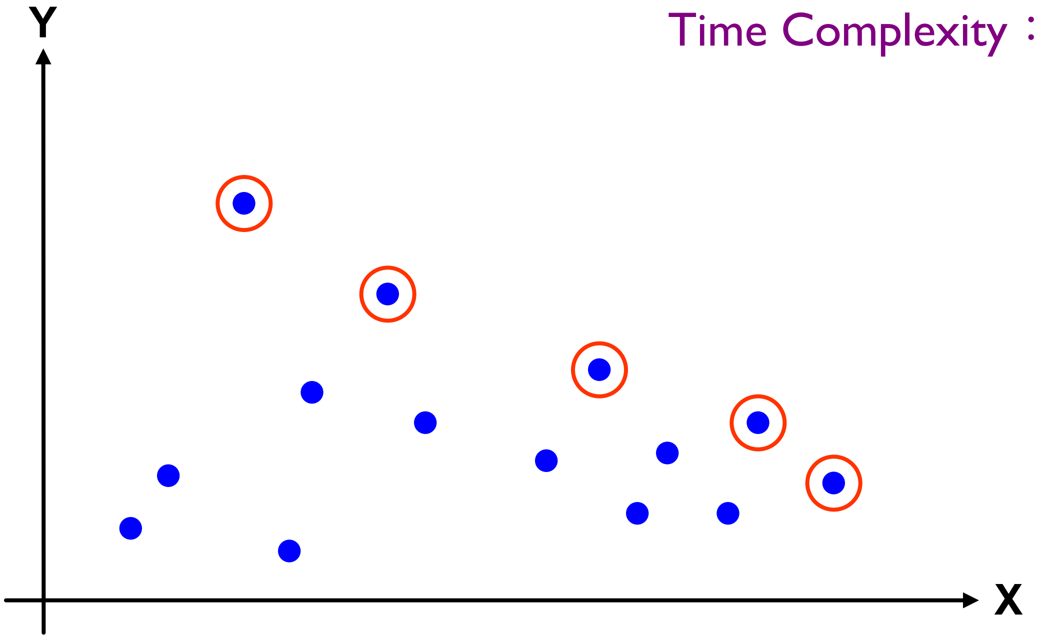
- Solve and Conquer



Example: Find the 2D Maximal Points

- Brute-force method
  - Write a two-level for-loop to compare all pair of points

Compare all pairs of points
Time Complexity：$O(n^2)$



Find 2D Maximal Points using Divide and Conquer

Input: A set S of n planar points (sorted along x).

Output: The maximal points of S.

Step 1: If S contains only one point, return it as the maxima. Otherwise, find a line L perpendicular to the X-axis which separates S into S_L and S_R, with equal sizes.

Step 2: Recursively find the maximal points of S_L and S_R .

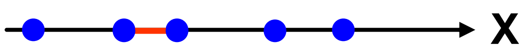
Step 3: Find the largest y-value in S_R, denoted as y_R. Discard each of the maximal points in S_L if its y-value is less than y_R.

```cpp
std::vector<Point> dc(const std::vector<Point>& S, int beg, int end) {

  if(!(beg < end)) return {};

  // base case
  if(end - beg == 1) return {S[beg]};

  // recursion
  int m = (beg + end + 1) / 2;
  auto SL = dc(S, beg, m);
  auto SR = dc(S, m, end);

  // find the highest y in SR
  int ymax = std::numeric_limits<int>::min();
  for(const auto& p : SR) {
    ymax = std::max(ymax, p.y);
  }

  // delete all points with y less than ymax from SL
  for(const auto& p : SL) {
    if(ymax > p.y) {
      continue;
    }
    SR.push_back(p);
  }

  return SR;
}
```

The 2D Closest Pair Problem

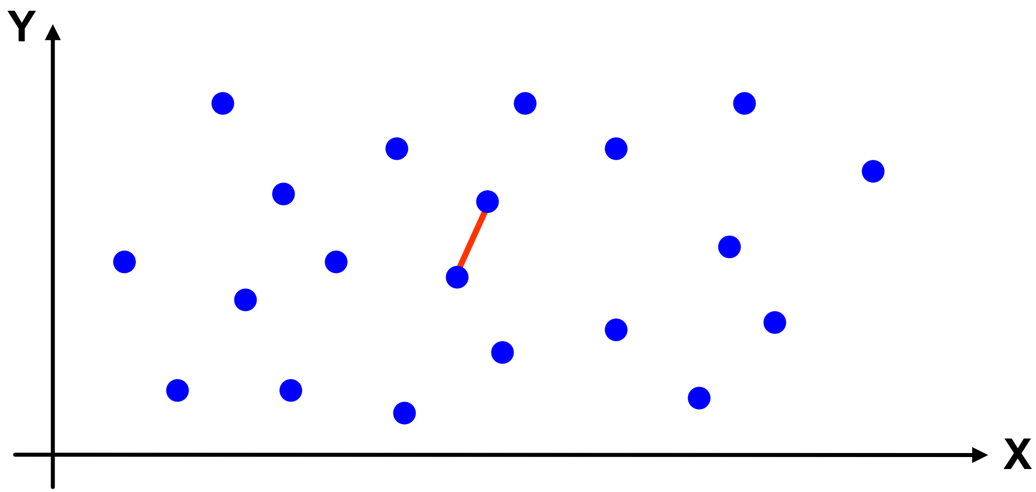
- Given a point set at a 2D plane, find a pair of points with the *minimum* distance

1-D version:



Sort and scan

2-D version:



Divide and Conquer Algorithm

Input: A set S of n planar points.

Output: The distance between two closest points.

Step 1: Sort S in increasing order of x values

Step 2: If S contains only one point, return infinity as its distance.

Step 3: Find a median line L perpendicular to the x-axis and divide S into S_L and S_R with equal sizes.

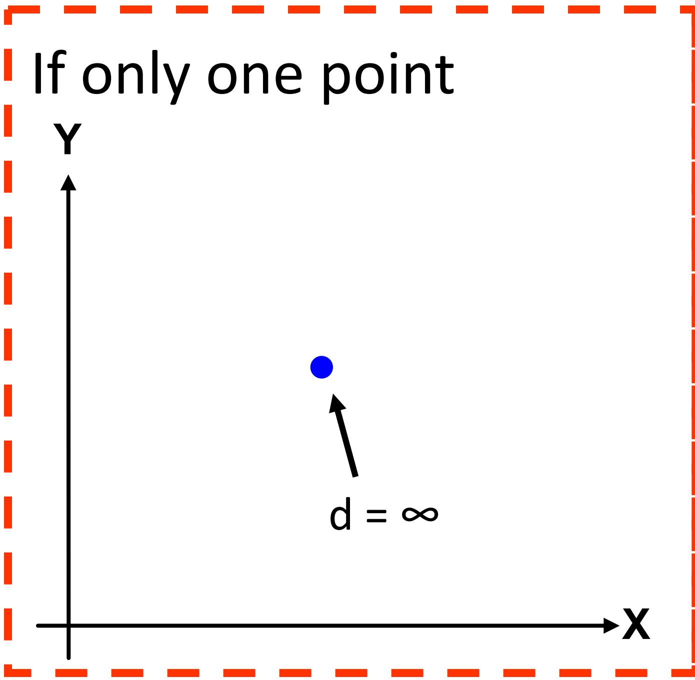
Step 4: Recursively apply Steps 2 and 3 to solve the closest pair problems of S_L and S_R. Let $d_L(d_R)$ denote the distance between the closest pair in S_L (S_R). Let d = $\min(d_L, d_R)$; Find a stripe of +d/-d from the mid point and search the minimum distance within the stripe; return the minimum

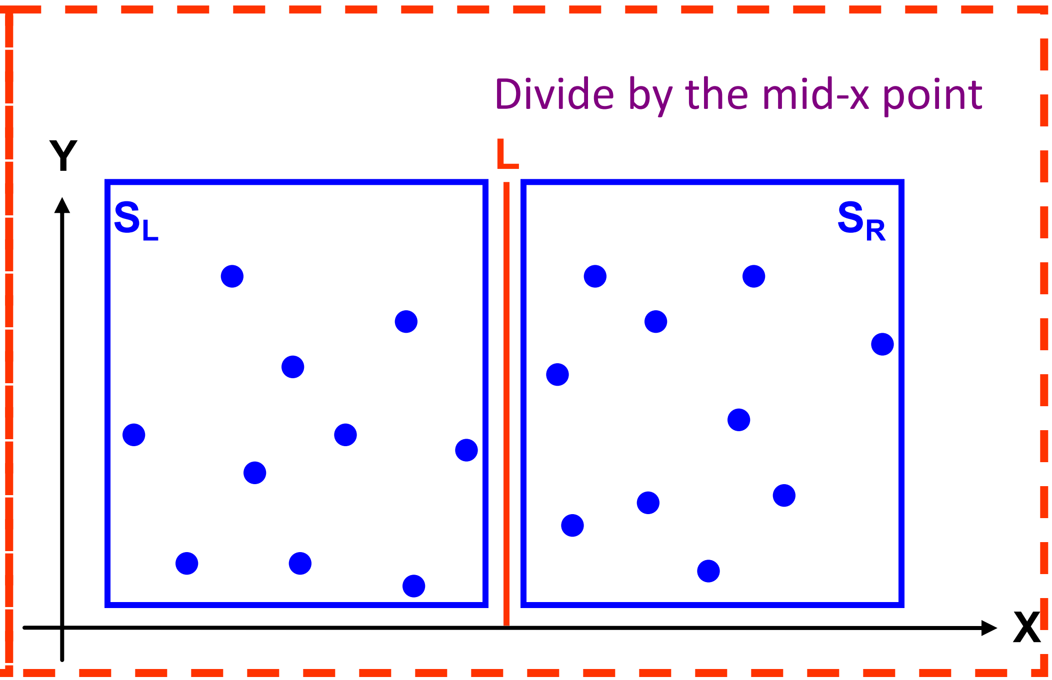
Illustration

Step 1: sort by the x and y coordinate

Step 2

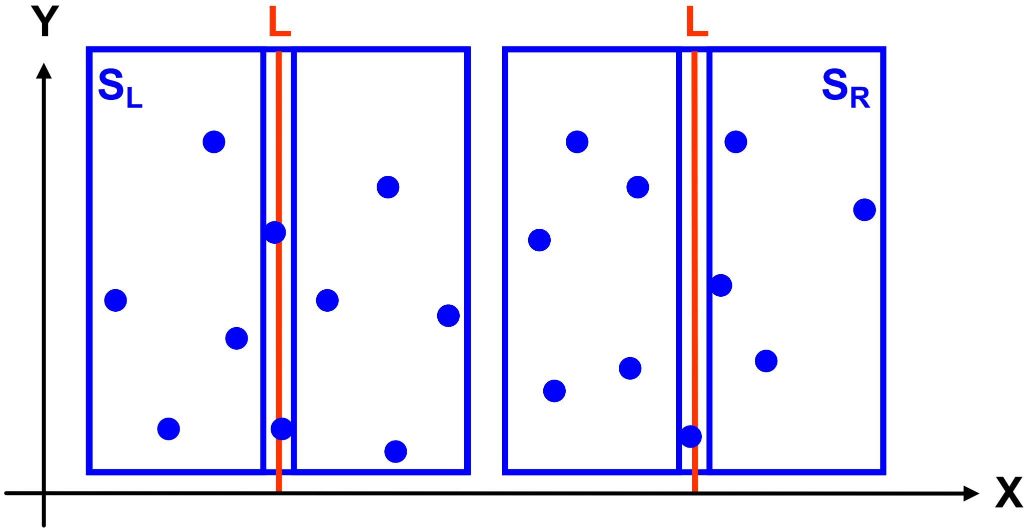
Step 3





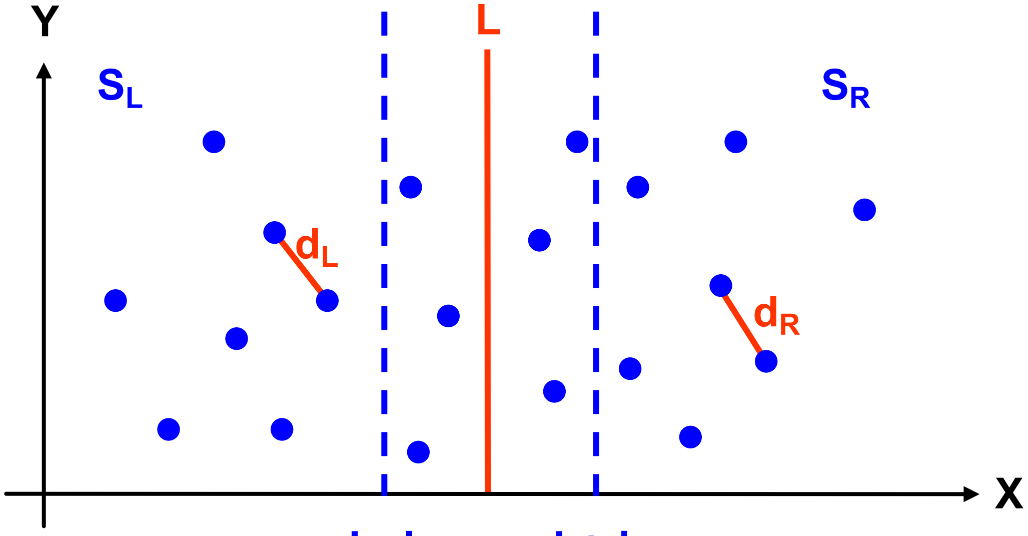
Illustration

Step 4



Illustration

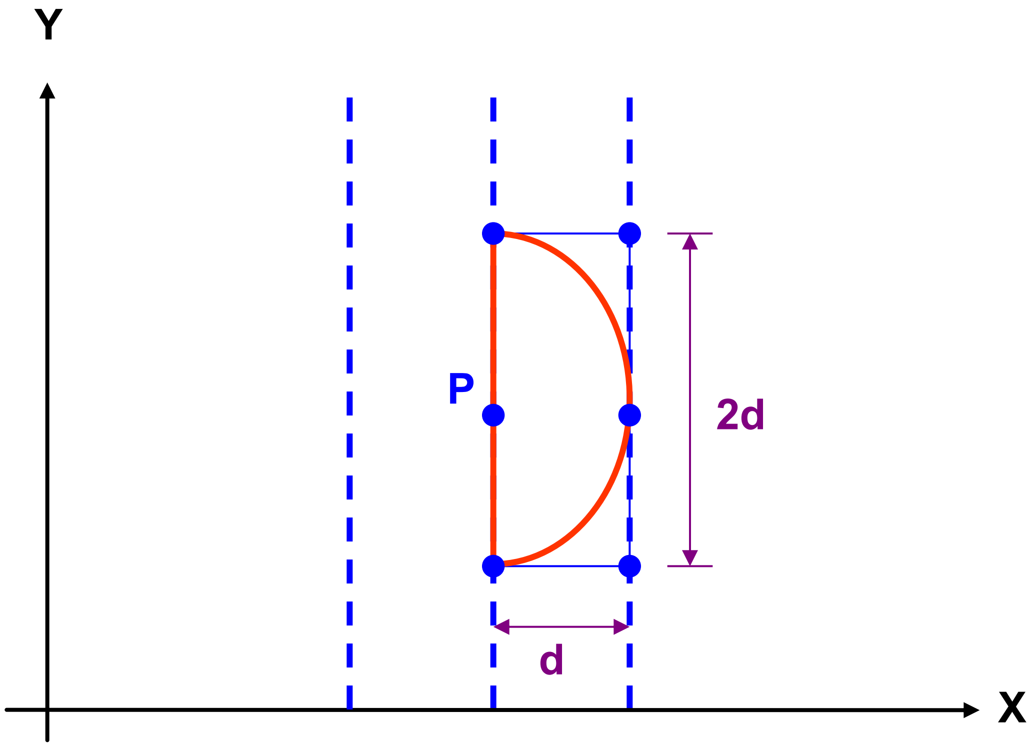
Step 5



$d=\min(d_L,d_R)$

Illustration

- You will not search too many points in the strip



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

Brute Force?

- Run two loops

```
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;
}
```

Divide and Conquer

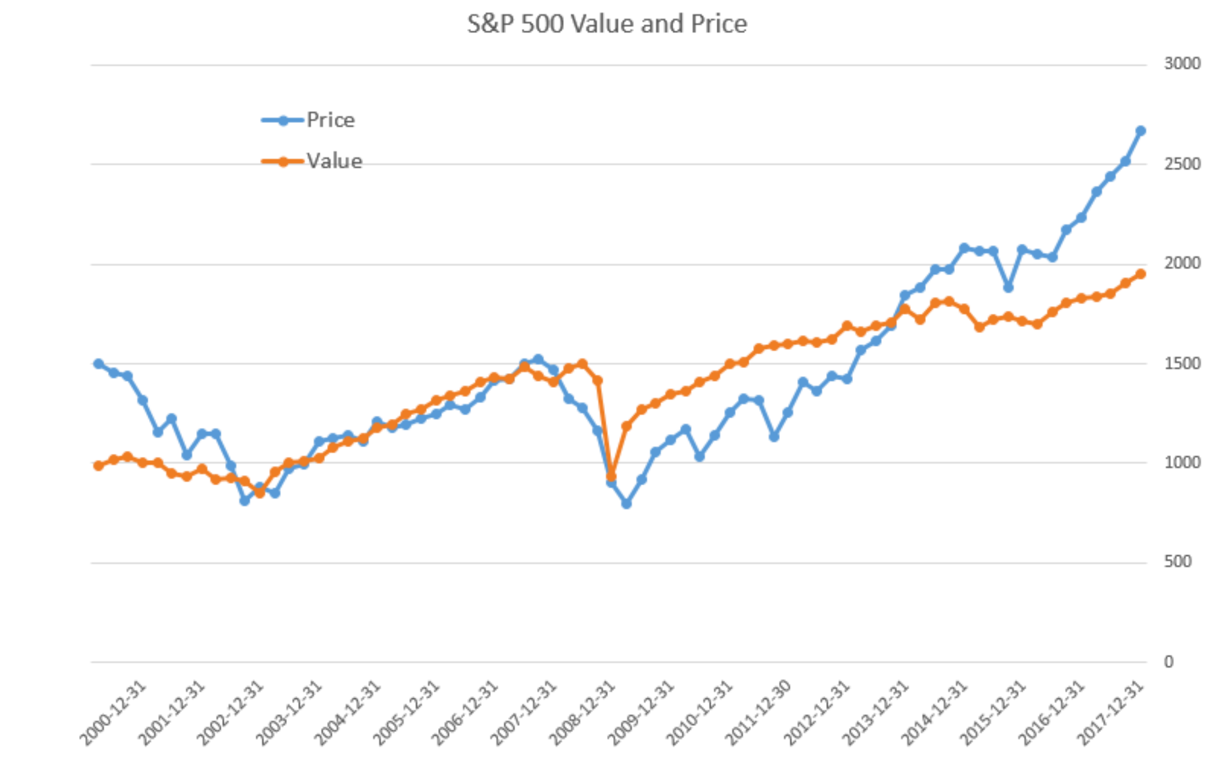
Step 1: If the array has fewer than 3 elements, go brute force

Step 2: Partition the array into two halves of equal size. Recursively find the maximum subarray sum of S_L and S_R .

Step 3: Merge two sums from S_L and S_R. Find the maximum subarray sum across the mid point. Return the overall minimum

Practical Application I

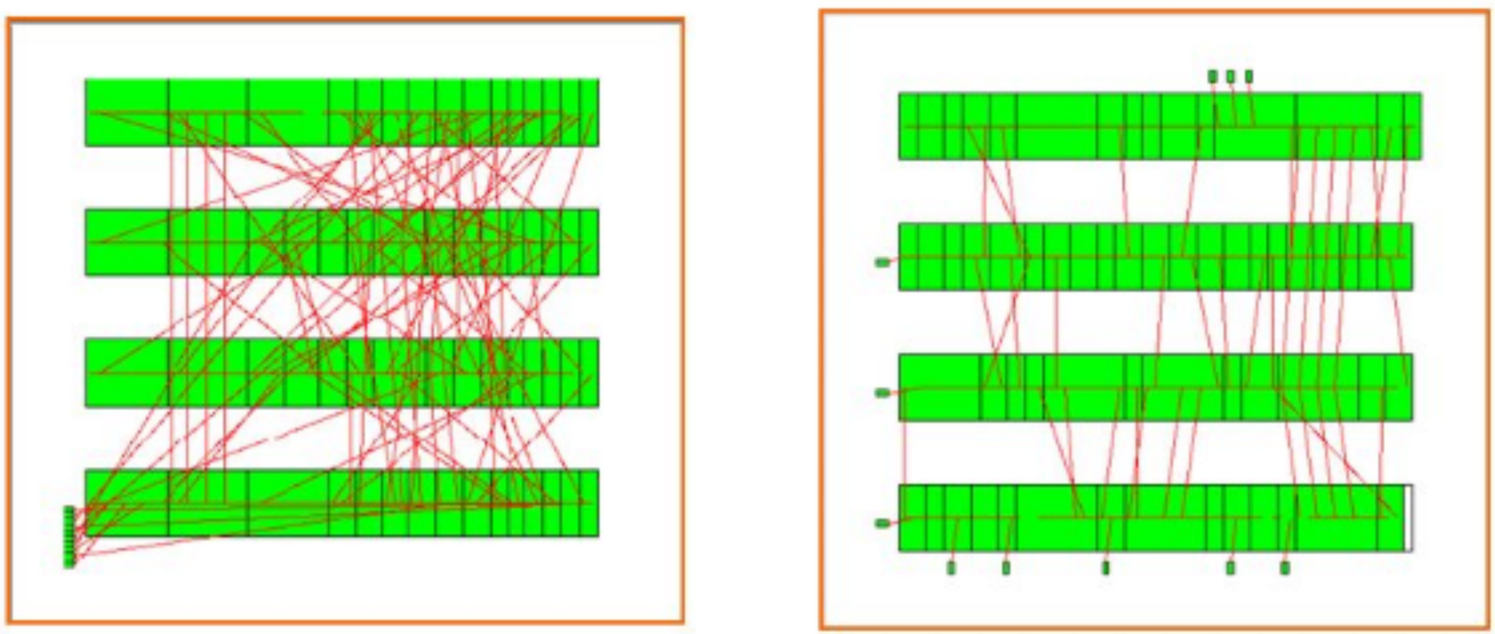
- ❑ **A basic routine of financial computing**



Maximum subarray sum to find the optimal long-term investment

Practical Application II

- ❑ **Row-based VLSI detailed placement**



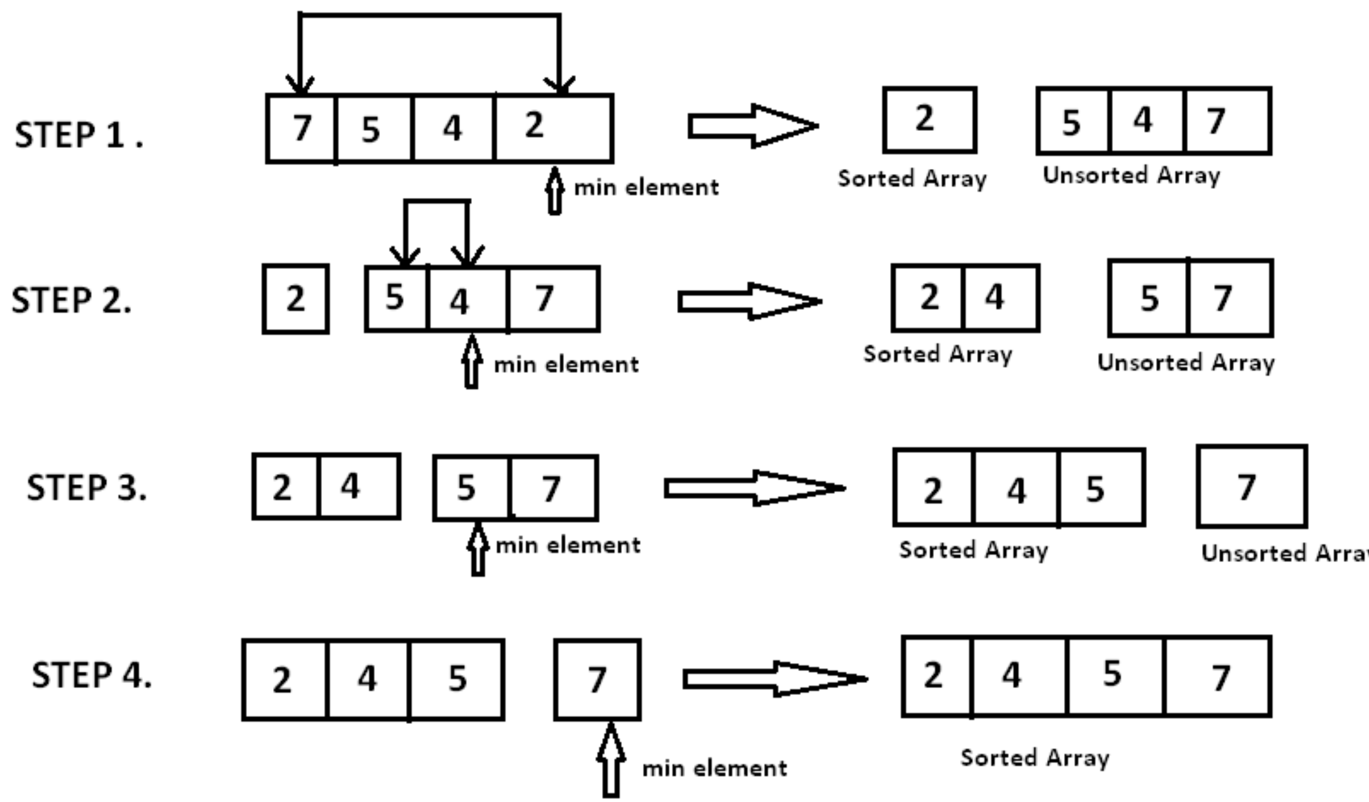
Maximum/minimum subarray sum to find per-row wirelength

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

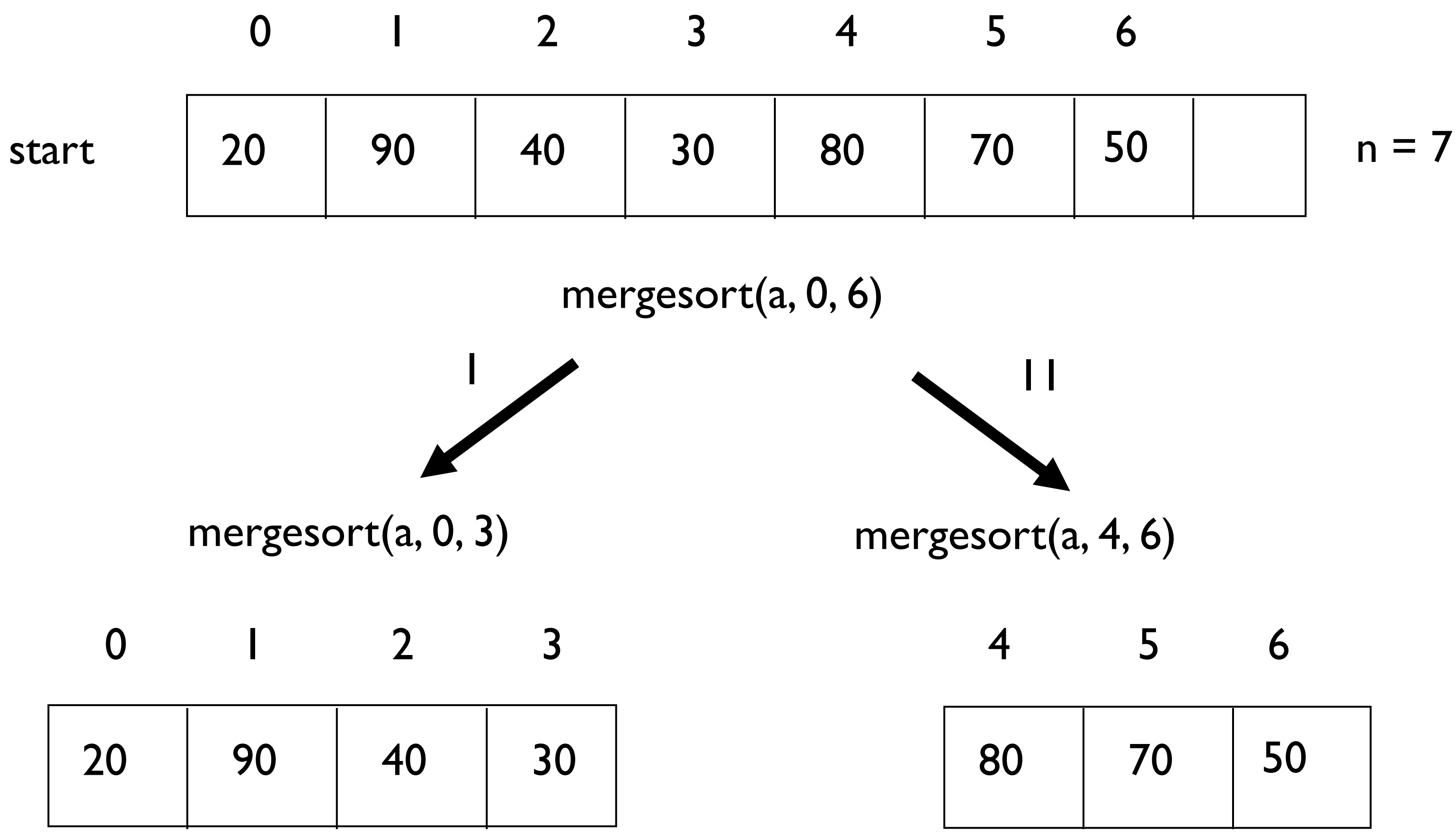
```
void brute_force(std::vector<int>& D, int beg, int end) {
  int max = std::numeric_limits<int>::min();
  for (int i = beg; i < end; ++i) {
    int min_v = D[i];
    int min_j = i;
    for (int j = i+1; j < end; ++j) {
      if(D[j] < min_v) {
        min_v = D[j];
        min_j = j;
      }
    }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

Time complexity $O(N^2)$

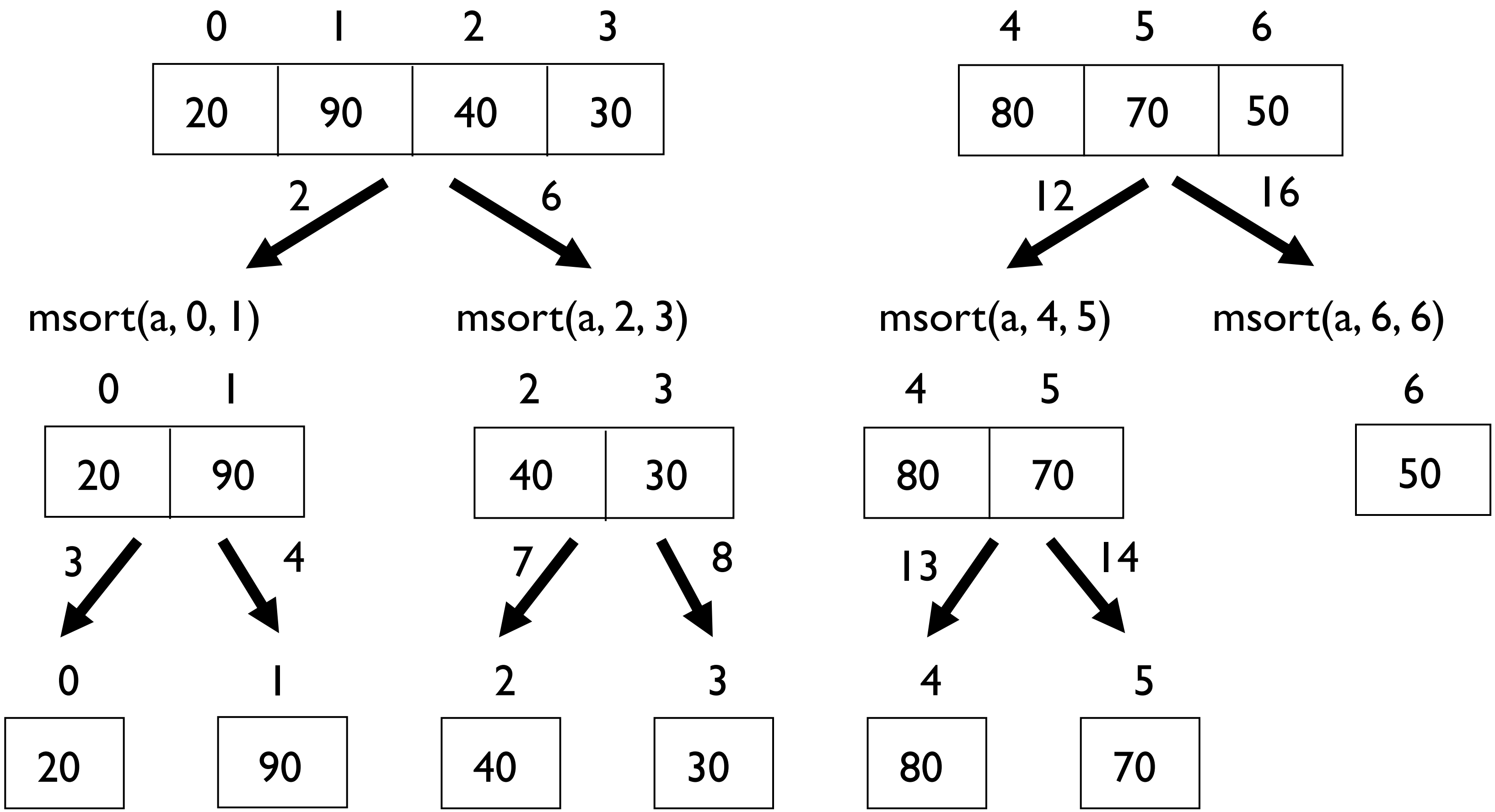
Using Divide and Conquer: Merge Sort

- **Divide**: If S has at leas two 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_1 and S_2, each containing about half of the elements of S. (i.e. S_1 contains the first $\lceil n/2 \rceil$ elements and S_2 contains the remaining $\lfloor n/2 \rfloor$ elements.
- **Recurse**: Recursive sort sequences S_1 and S_2.
- **Conquer**: Put back the elements into S by merging the sorted sequences S_1 and S_2 into a unique sorted sequence.

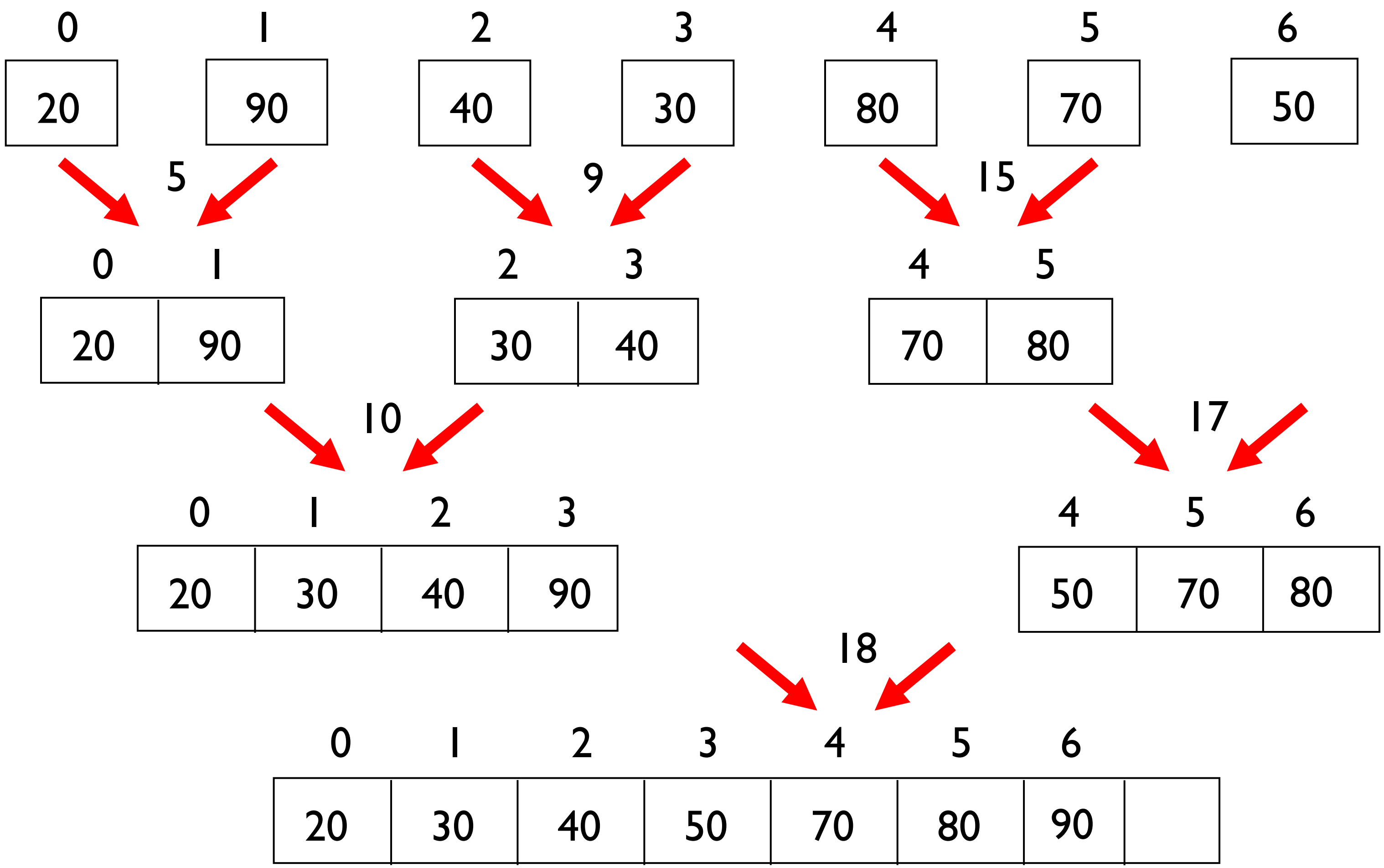
Illustration



Illustration

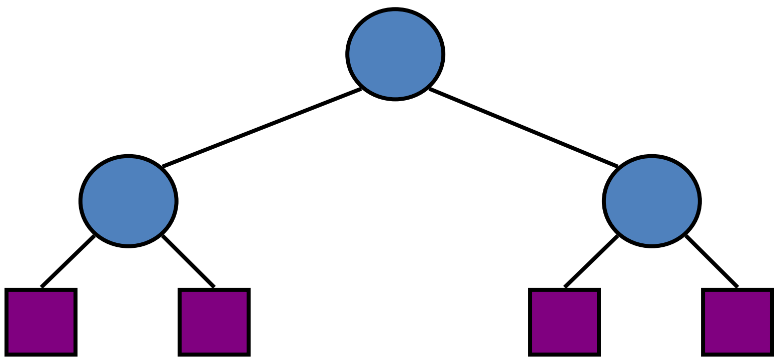


Illustration



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

- **2D closest pair of points finding**
  - Commonly used in computational geometry
- **Maximum subarray sum**
  - Commonly used in financial computing and VLSI designs
- **Sorting**
  - Insertion sort (kinda brute force: $O(N^2)$)
  - Merge sort (divide and conquer: $O(n\log n)$)

Note

- **To compile a “test.cpp” program to a binary “test”**
  - g++ test.cpp -O2 -o test
- **To feed a program with a test file from the standard output:**
  - ./simple < test.txt
- **To measure the runtime of the above program:**
  - time –p ./simple < test.txt