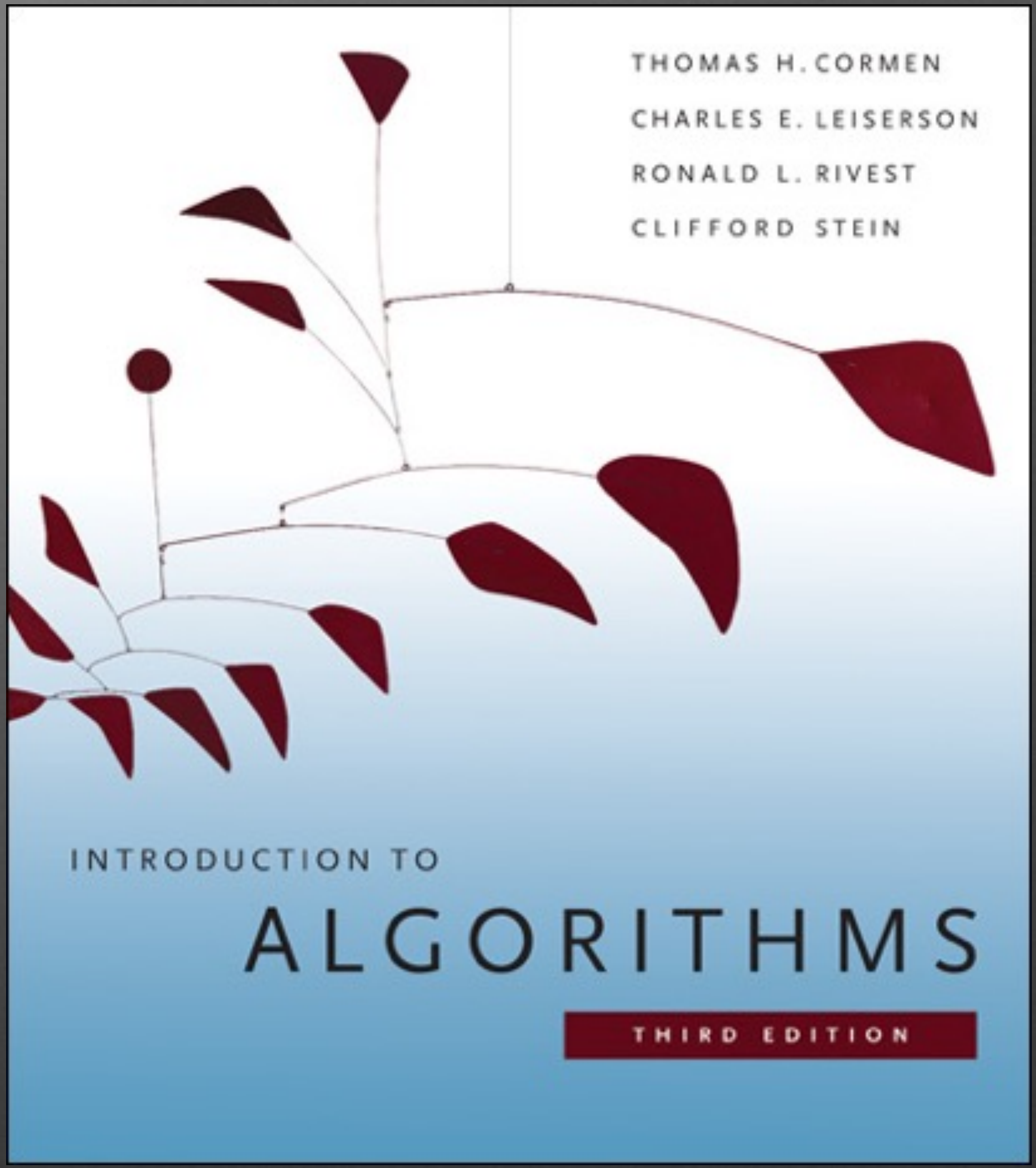
Algorithms

王士哲

Chapter 0

Reference

聖經



Language

- C/C++
- Pseudo code

Plan

- Artificial Intelligence
- Machine Learning

Chapter 1

The Role of Algorithms in Computing

What are algorithms?

- Computational problem
- Input/Output
- A procedure is designed for achieving that input/output relation

What kinds of problems are solved by algorithms?

Search engine

- Input : keyword
- Output : Website list

Traveling salesman problem

- Input : road map, target city, cost
- Output : a path cost is minimum

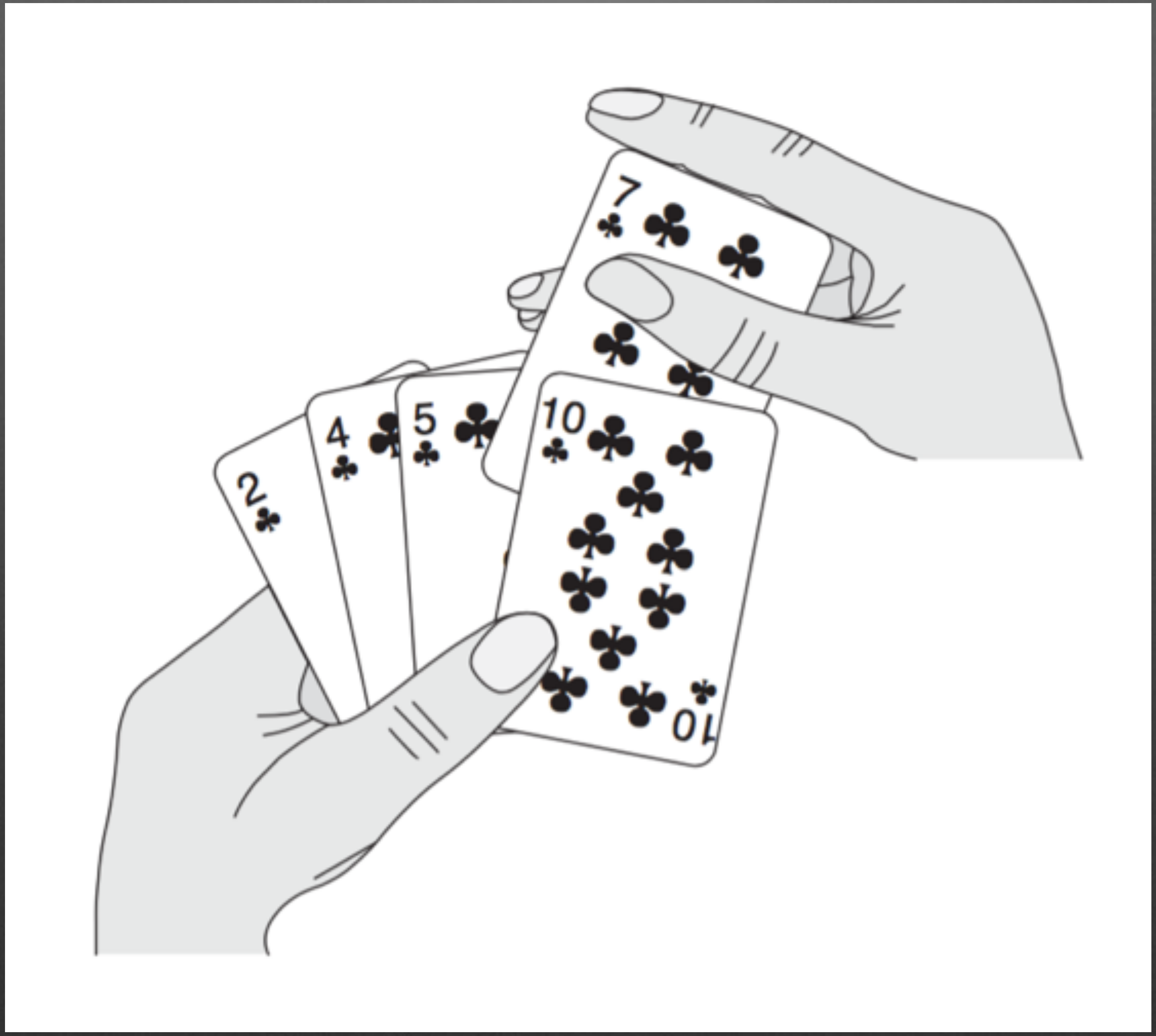
Sorting problem

- Input : A sequence of n numbers a[1 … n]
- Output : A permutation a[1 … n] of the input sequence such a[1] <= a[2] <= ….. <= a[n]

Chapter 2

getting started

Insertion sort



Insertion sort

- Insertion-Sort(A)

```
for j = 2 to A.length
key = A[ j ]
i = j-1
while i  > 0 and A[ i ] > key
    A [ i+1 ] = A[ i ]
    i=i-1
A[ i+1 ] = key
```

Loop invariants

- Initialization: It is true prior to the first iteration of the loop
- Maintenance: If it is true before an iteration of the loop, it remains true before the next iteration
- Termination: When the loop terminates, the invariant gives us a useful property that helps show that the algorithm is correct

Loop invariants

- Initialization：描述 Invariant Condition 在迴圈執行第一個 iteration前，就成立

- Maintenance：描述 Invariant Condition 在迴圈的任一iteration執行前跟執行後都維持成立

- Termination：迴圈執行結束後，Invariant Condition 能夠展示整體演算法的正確性

Loop invariants

- subarray A[1..j-1] is sorted
- Initialization：start at j =2 , A[1] is sorted
- Maintenance：every loop works by moving A[j-1] … until it finds position for A[j]
- Termination：terminal at j = A.length + 1 => A[1..A.length] is sorted

Analysis of insertion sort

- Insertion-Sort(A)

```
for j = 2 to A.length
    key = A[ j ]
    i = j-1
    while i  > 0 and A[ i ] > key
        A [ i+1 ] = A[ i ]
        i=i-1
    A[ i+1 ] = key
```

Chapter 3

Growth of Function

大學以前有教

Chapter 4

Divide-and-Conquer

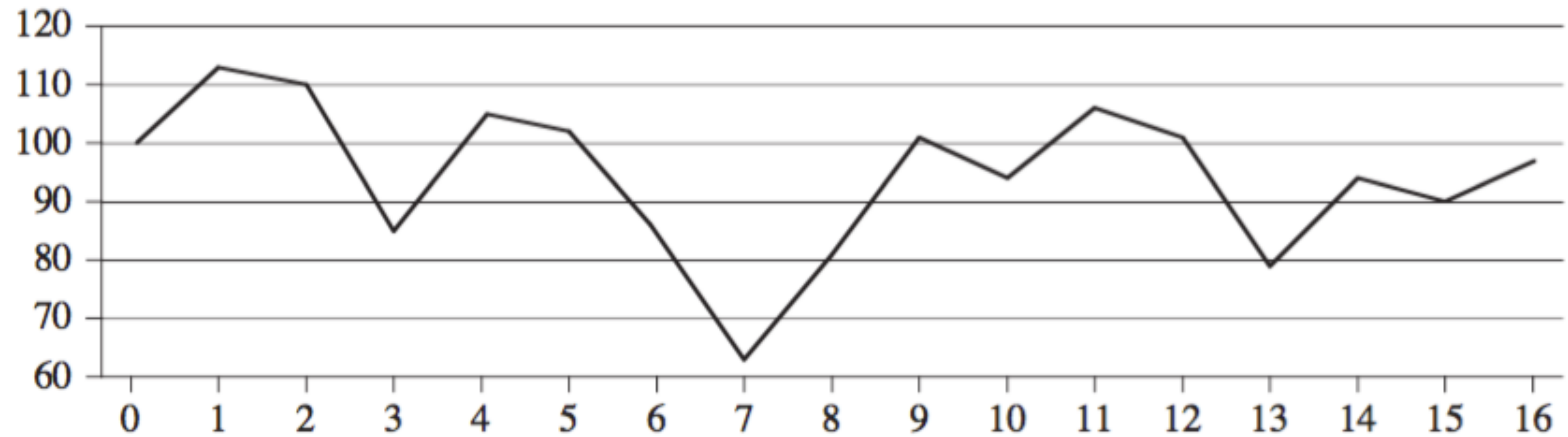
Divide-and-Conquer

- Divide the problem into a number of subproblems that are smaller instances of the same problem.
- Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- Combine the solutions to the subproblems into the solution for the original problem.

Divide-and-Conquer

- Divide 問題使原本的問題變成多個數量級小一點的相同問題
- Conquer 使用遞迴拆解問題直到問題夠小能直接獲得答案
- Combine 把所有子問題的答案組合起來成為原本問題的答案

The maximum-subarray problem



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

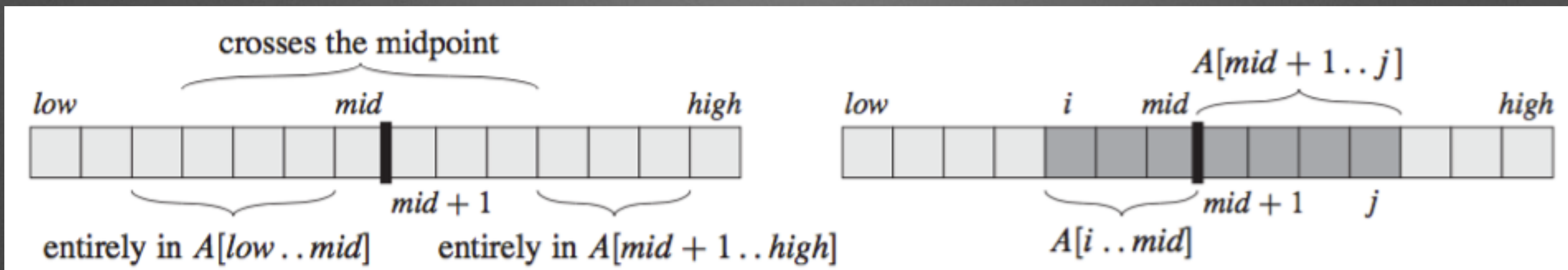
The maximum-subarray problem

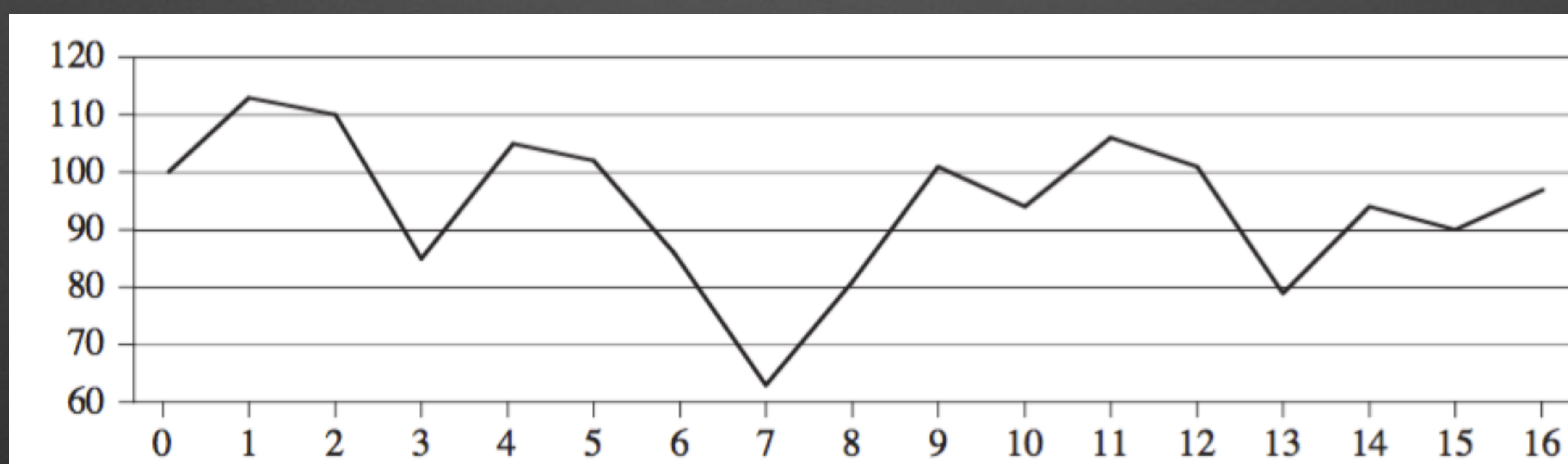
```
int DAYS = 17;
int price[DAYS] = {100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90};
int change[DAYS] = {0};
int max =0;
int buyDay=0;
int sellDay=0;
class Point{
    int max = 0;
    int buyDay = 0;
    int sellDay = 0;
    Point(int max,int buyDay,int sellDay){
        this.max = max;
        this.buyDay = buyDay;
        this.sellDay = sellDay;
    }

}
```

The maximum-subarray problem (Brute-force)

```
Point brute-force-solution(){
    Point ans(0,0,0);
    for(int i = 0 ; i < DAYS-1 ; i++){
        for(int j =i+1 ; j < DAYS ; j++){
            int temp = price[j]-price[i];
            if(ans.max < temp){
                ans.max = temp;
                ans.buyDay = i;
                ans.sellDay = j;
            }
        }
    }
    return ans;
}
```

The maximum-subarray problem (divide-and-conquer)





Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

The maximum-subarray problem (divide-and-conquer)

```
void init(){
    for(int i=1;i<DAYS;i++){
        change[i] = price[i]-price[i-1];
    }
}
Point divide(){
    init();
    return findMaxSubArray(0,DAYS-1);
}
```

The maximum-subarray problem (divide-and-conquer)

```
Point findMaxCrossingSubArray(int low,int mid,int high){
    int leftSum = INT_MIN,maxLeft = mid;
    int sum = 0;
    Point a;
    for(int i = mid ; i >= low ; i--){
        sum += change[i];
        if(sum > leftSum){
            leftSum = sum;
            a.buyDay = i;
        }
    }
    int rightSum = INT_MIN,maxRight = mid;
    sum = 0;
    for(int i = mid+1 ; i < high ; i++){
        sum += change[i];
        if(sum > rightSum){
            rightSum = sum;
            a.sellDay = i;
        }
    }
    a.max = rightSum + leftSum;
    return a;
}
```

The maximum-subarray problem (divide-and-conquer)

```
Point findMaxSubArray(int low, int high){
    if(low == high)return new Point(change[low],low,high);
    int mid = (low + high)/2;
    Point leftP = findMaxSubArray(low,mid);
    Point rightP = findMaxSubArray(mid+1,right);
    Point midP = findMaxCrossingSubArray(low,mid,high);
    return leftP.max >= midP.max ? (leftP.max >= rightP.max ? leftP : rightP) : (rightP.max >= midP.max ? rightP : midP);
}
```

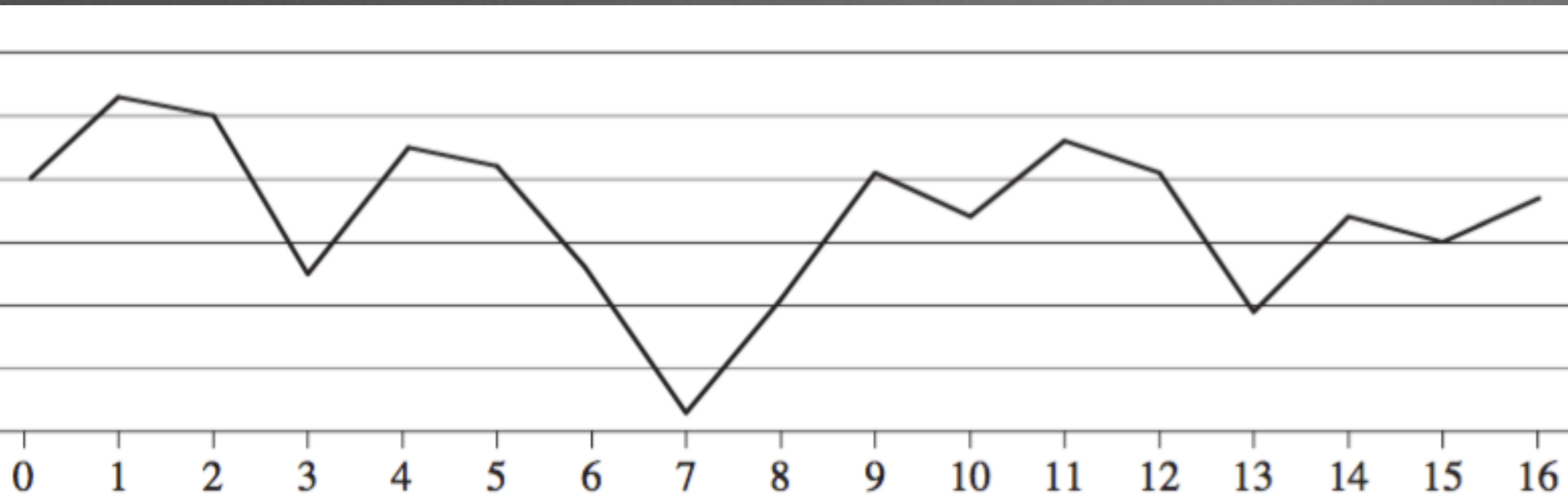
The maximum-subarray problem (divide-and-conquer)

```
void init(){
    for(int i=1;i<DAYS;i++){
        change[i] = price[i]-price[i-1];
    }
}
Point divide(){
    init();
    return findMaxSubArray(0,DAYS-1);
}
```

Analyzing

- $T(1) = O(1)$
- $T(n) = 2T(n/2) + O(n) = O(n \lg n)$

The maximum-subarray problem (dynamic programming)



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
ge		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

- 0,13,10,0,20,17,1,0,18,38,31,43,38,16,31,27,34

```
Point findMaxPointUsingDP(){
    int temp[DAYS] ={0};
    Point ans(0,0,0);
    for(int i=1;i<DAYS;i++){
        temp[i] += change[i];
        if(temp[i] == 0)temp[i] = 0;
        if(temp[i] > max){
            a.max = temp[i];
            a.sell = i;
        }
    }
    for(int i = sell;i>=0;i--){
        if(temp[i] == 0 ){
            a.buy = i;
            break;
        }
    }
    return ans;
}
```

Analyzing

- Time   O(n)
- Space   O(n)

312. Burst Balloons

- Input : nums[]
- output : maxCoins
- nums = [3,1,5,8] --> [3,5,8] --> [3,8] --> [8] --> []
- coins = 3*1*5 + 3*5*8 + 1*3*8 + 1*8*1 = 167

312. Burst Balloons (divide-and-conquer)

```
int maxCoins(int left,int right,vector<int>& nums){
    if(right - left == 1){
        return 0;
    }
    int max =0;
    for(int mid = left+1 ;mid < right; mid++){
        int temp = nums[mid]*nums[left] * nums[right] +
                    maxCoins(left,mid,nums) +
                    maxCoins(mid,right,nums);
        if(max <temp ){
            max = temp;
        }
    }
    return max;
}
```

Analyzing

```
int maxCoins(int left,int right,vector<int>& nums){
    if(right - left == 1){
        return 0;
    }
    int max =0;
    for(int mid = left+1 ;mid < right; mid++){
        int temp = nums[mid]*nums[left] * nums[right] +
                      maxCoins(left,mid,nums) +
                      maxCoins(mid,right,nums);
        if(max <temp ){
            max = temp;
        }
    }
    return max;
}
```

- Time $T(n) = n * 2 *(T(1) + T(2) + \dots T(n-1)) + O(1)$

$T(n-1) = (n-1) * 2 * (T(1) + \dots. + T(n-2)) + O(1)$

$T(n) - T(n-1) \to T(n) = S + (n-1)*(T(n-1))$

$n*T(n) = n*S + n * (n-1) * (T(n-1))$, $T(n) = n * S$

$T(n) = n * (n-1) * T(n-1) / (n-1) = n * T(n-1)$

$T(n) = n!$

312. Burst Balloons (dynamic programming)

```
int maxCoins(int left,int right,vector<int>& nums,int* dp,int size){
    if(dp[left + right * size]>0)return dp[left + right * size];
    if(right - left == 1){
        return 0;
    }
    int max =0;
    for(int mid = left+1 ;mid < right; mid++){
        int temp = nums[mid]*nums[left] * nums[right] +
                   maxCoins(left,mid,nums,dp,size) +
                   maxCoins(mid,right,nums,dp,size);
        if(max <temp ){
            max = temp;
        }
    }
    dp[left + right * size] = max;
    return max;
}
```

Analyzing

```
int maxCoins(int left,int right,vector<int>& nums,int* dp,int size){
    if(dp[left + right * size]>0)return dp[left + right * size];
    if(right - left == 1){
        return 0;
    }
    int max =0;
    for(int mid = left+1 ;mid < right; mid++){
        int temp = nums[mid]*nums[left] * nums[right] +
                   maxCoins(left,mid,nums,dp,size) +
                   maxCoins(mid,right,nums,dp,size);
        if(max <temp ){
            max = temp;
        }
    }
    dp[left + right * size] = max;
    return max;
}
```

- Time   T(n) = O(n*n) + O(n*n) +… + O(n*n)
  = O(n^3)
- Space   O(n^2)

- Chapter 1 : What algorithm is
- Chapter 2 : Sample
- Chapter 3 : 大學以前有教
- Chapter 4 : Divide-and-Conquer

Q & A