§3   Compare the Algorithms

〖**Example**〗 **Given (possibly negative) integers** $A_1, A_2, \ldots,$ A_N**, find the maximum value of** $\sum_{k=i}^{j} A_k$ **.**

Algorithm 1

Max sum is 0 if all

```c
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j,  k;
/* 1*/      MaxSum = 0;   /* initialize the maximum sum */
/* 2*/      for( i = 0; i < N; i++ )  /* start from A[ i ] */
/* 3*/          for( j = i; j < N; j++ ) {   /* end at A[ ]
/* 4*/              ThisSum = 0;
/* 5*/              for( k = i; k <= j; k++ )
/* 6*/                  ThisSum += A[ k ];  /* su                [ j ] */
/* 7*/              if ( ThisSum > MaxSum )
/* 8*/                  MaxSum = ThisSum;  /* up    e max sum */
            }  /* end for-j and for-i */
/* 9*/      return  MaxSum;
}
```

Detailed analysis is given on p.18-19.

$$T(N) = O(N^3)$$

Algorithm 2
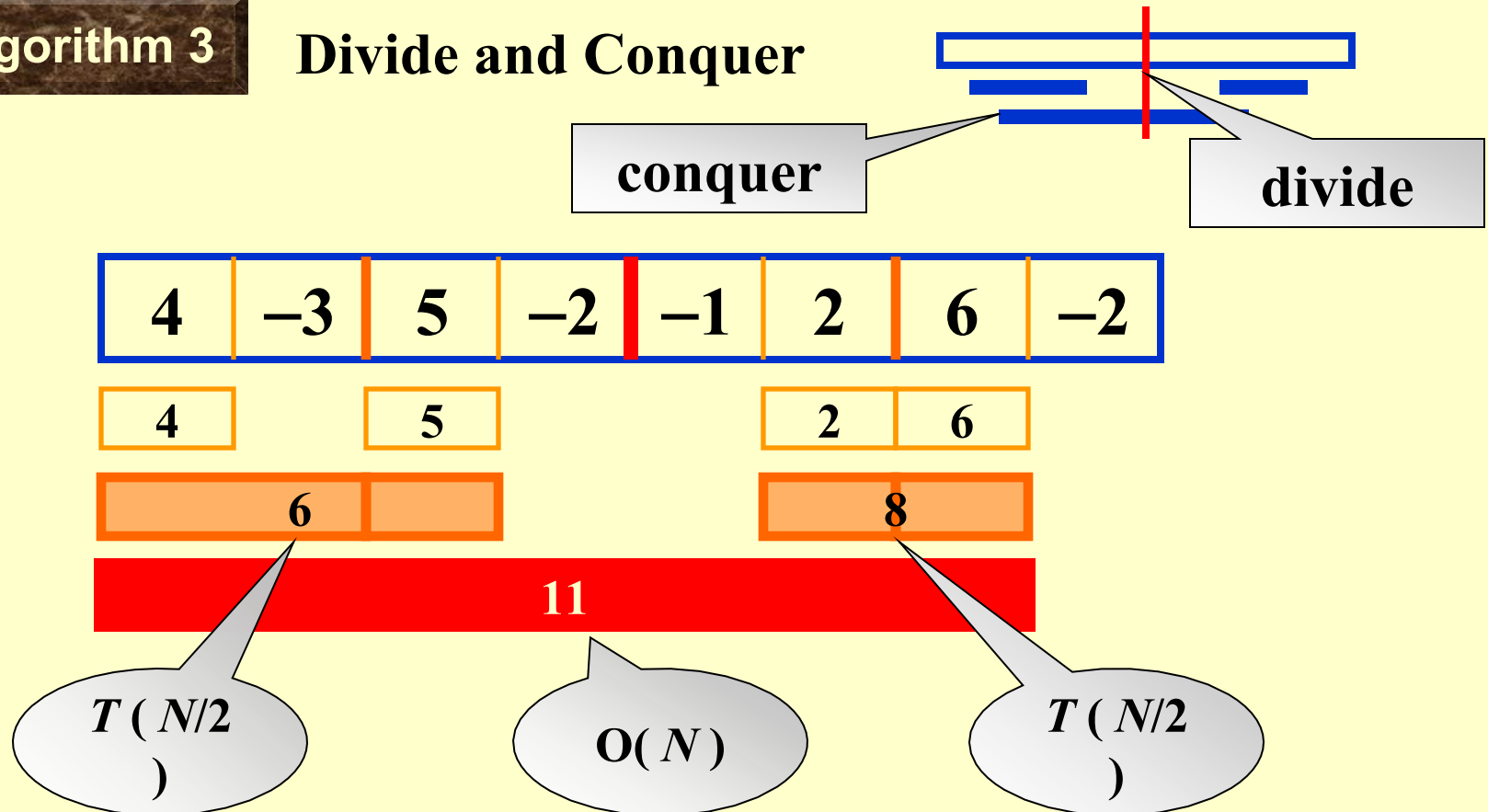
```
int  MaxSubsequenceSum ( const int A[ ],  int  N )
{
        int  ThisSum,  MaxSum,  i,  j;
/* 1*/      MaxSum = 0;   /* initialize the maximum sum */
/* 2*/      for( i = 0; i < N; i++ )  {   /* start from A[ i ] */
/* 3*/          ThisSum = 0;
/* 4*/          for( j = i; j < N; j++ ) {   /* end at A[ j ] */
/* 5*/              ThisSum += A[ j ];  /* sum from A[ i ] to A[ j ] */
/* 6*/              if ( ThisSum > MaxSum )
/* 7*/                  MaxSum = ThisSum;  /* update max sum */
            }  /* end for-j */
        }  /* end for-i */
/* 8*/      return  MaxSum;
}
```

$$T(N) = O(N^2)$$

Algorithm 3   **Divide and Conquer**

conquer   **divide**

4	–3	5	–2	–1	2	6	–2

4		5			2	6

6			8

11

$T (N/2)$

$O(N)$

$T (N/2)$

$T (N) = 2\ T(N/2) + c\ N ,\qquad T(1) = O(1)$

$\qquad\quad = 2\ [2\ T(N/2^2) + c\ N/2] + c\ N$

$\qquad\quad = 2^k\ O(1) + c\ k\ N \qquad$ where $N/2^k$

$\qquad\quad = O(N \log N)$

The program can be found on p.21.

Algorithm 4 **On-line Algorithm**

```
int MaxSubsequenceSum( const int  A[ ],  int  N )
{
            int  ThisSum, MaxSum, j;
/* 1*/      ThisSum = MaxSum = 0;
/* 2*/      for ( j = 0; j < N; j++ ) {
/* 3*/          ThisSum += A[ j ];
/* 4*/           if  ( ThisSum > MaxSum )
/* 5*/                 MaxSum = ThisSum;
/* 6*/           else if ( ThisSum < 0 )
/* 7*/                 ThisSum = 0;
            }  /* end for-j */
/* 8*/      return MaxSum;
}
```

| −1 | 3 | −2 | 4 | −6 | 1 | 6 | −1 |

At any point in time, the algorithm can correctly give an answer to the **subsequence** problem for the data it has already read.

$T(N) = O(N)$

A[ ] is scanned **once** only.

Running times of several algorithms for maximum subsequence sum (in seconds)

Algorithm		1	2	3	4
Time		$O(N^3)$	$O(N^2)$	$O(N \log N)$	$O(N)$
Input Size	$N=10$	0.00103	0.00045	0.00066	0.00034
	$N=100$	0.47015	0.01112	0.00486	0.00063
	$N=1,000$	448.77	1.1233	0.05843	0.00333
	$N=10,000$	NA	111.13	0.68631	0.03042
	$N=100,000$	NA	NA	8.0113	0.29832

Note: The time required to read the input is not included.

§4 Logarithms in the Running Time

〖**Example**〗 **Binary Search:**

Given: A [0] $\leq$ A [1] $\leq$ …… $\leq$ A [N − 1] ; **X**

Task: Find **X**

Output: i if **X** = = A [i]

−1 if **X** is not found

low   mid   high

X ~ A [mid]

< == >

low   high = mid − 1 | low = mid + 1   high

mid
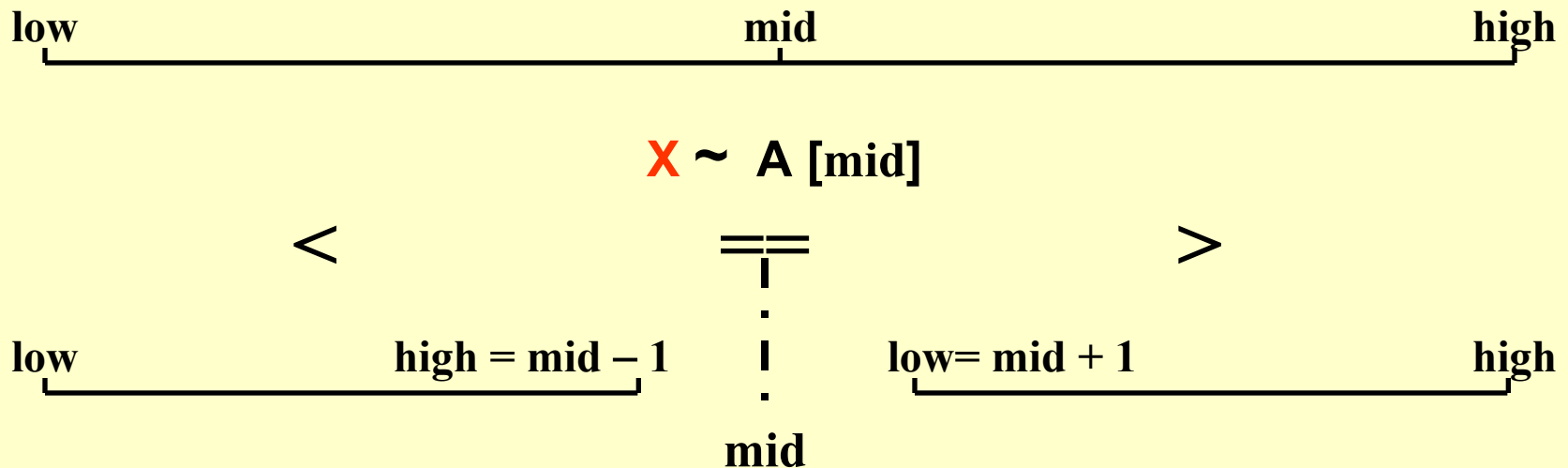
```
int BinarySearch ( const ElementType  A[ ],
                            ElementType  X,  int  N )
{
        int  Low, Mid, High;
/* 1*/  Low = 0;  High = N - 1:
/* 2*/  while (
/* 3*/
/* 4*/
```

Very useful in
e data are

Home work:
Self-study Euclid's Algorithm
and Exponentiation

```
/* 8*/
        }  /* end
/* 9*/  return  NotFound; /* NotFound is defined as -1 */
}
```

$$T_{worst}(N) = O(\log N)$$

§5 Checking Your Analysis

Method 1

When $T(N) = O(N)$, check if $T(2N)/T(N) \approx 2$

When $T(N) = O(N^2)$, check if $T(2N)/T(N) \approx 4$

When $T(N) = O(N^3)$, check if $T(2N)/T(N) \approx 8$

… …

Method 2

When $T(N) = O(f(N))$, check if

$$\lim_{N \to \infty} \frac{T(N)}{f(N)} \approx \text{Constant}$$

Read the example given on p.28 (Figures 2.12 & 2.13).