

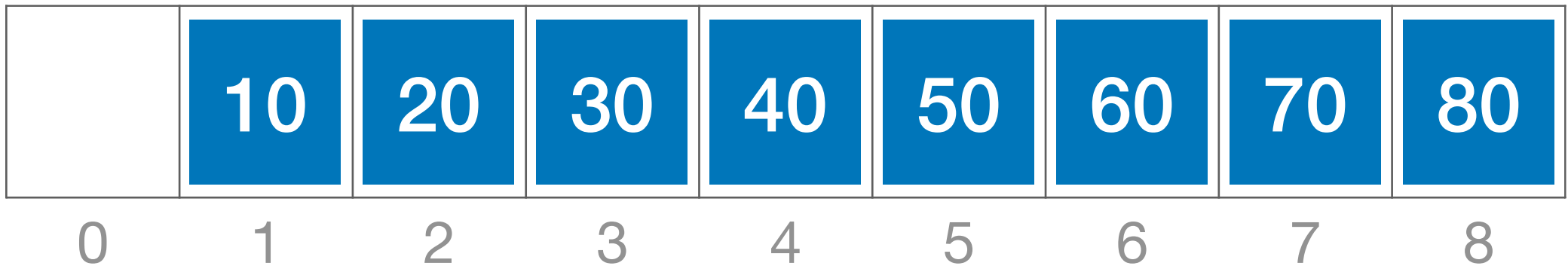
本节内容

# 希尔排序

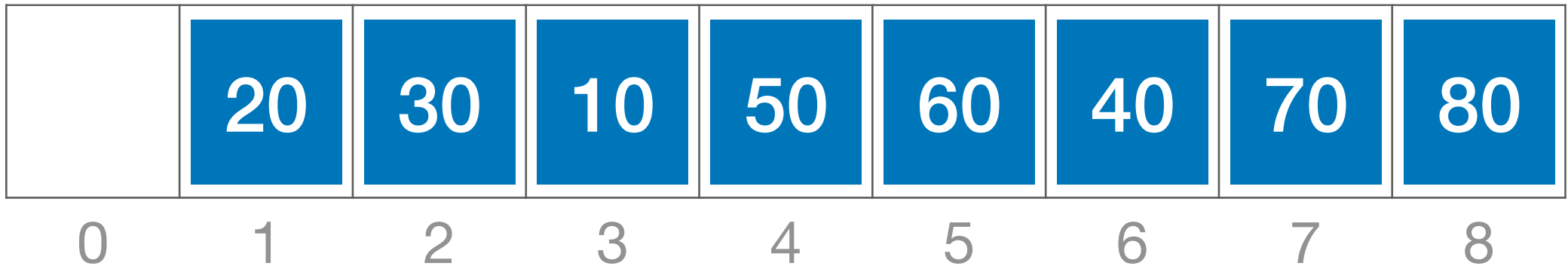
# 希尔排序 (Shell Sort)



最好情况：原本就有序



比较好的情况：基本有序

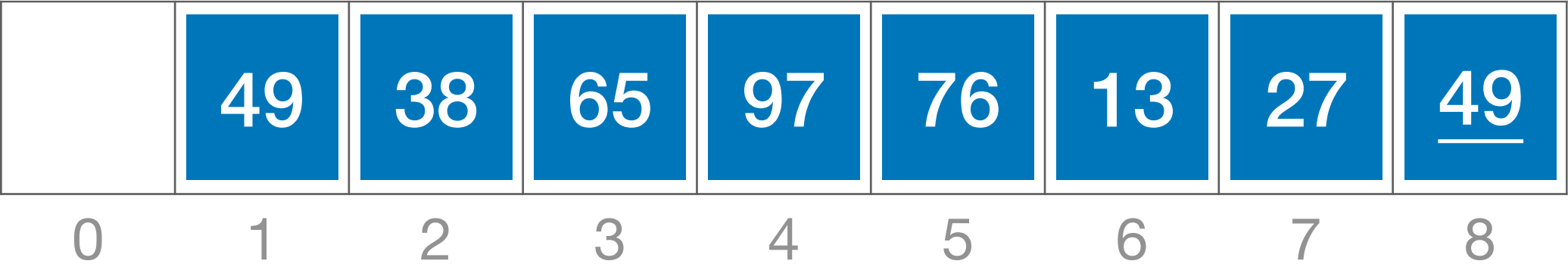


希尔排序：先追求表中元素部分有序，再逐渐逼近全局有序

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。



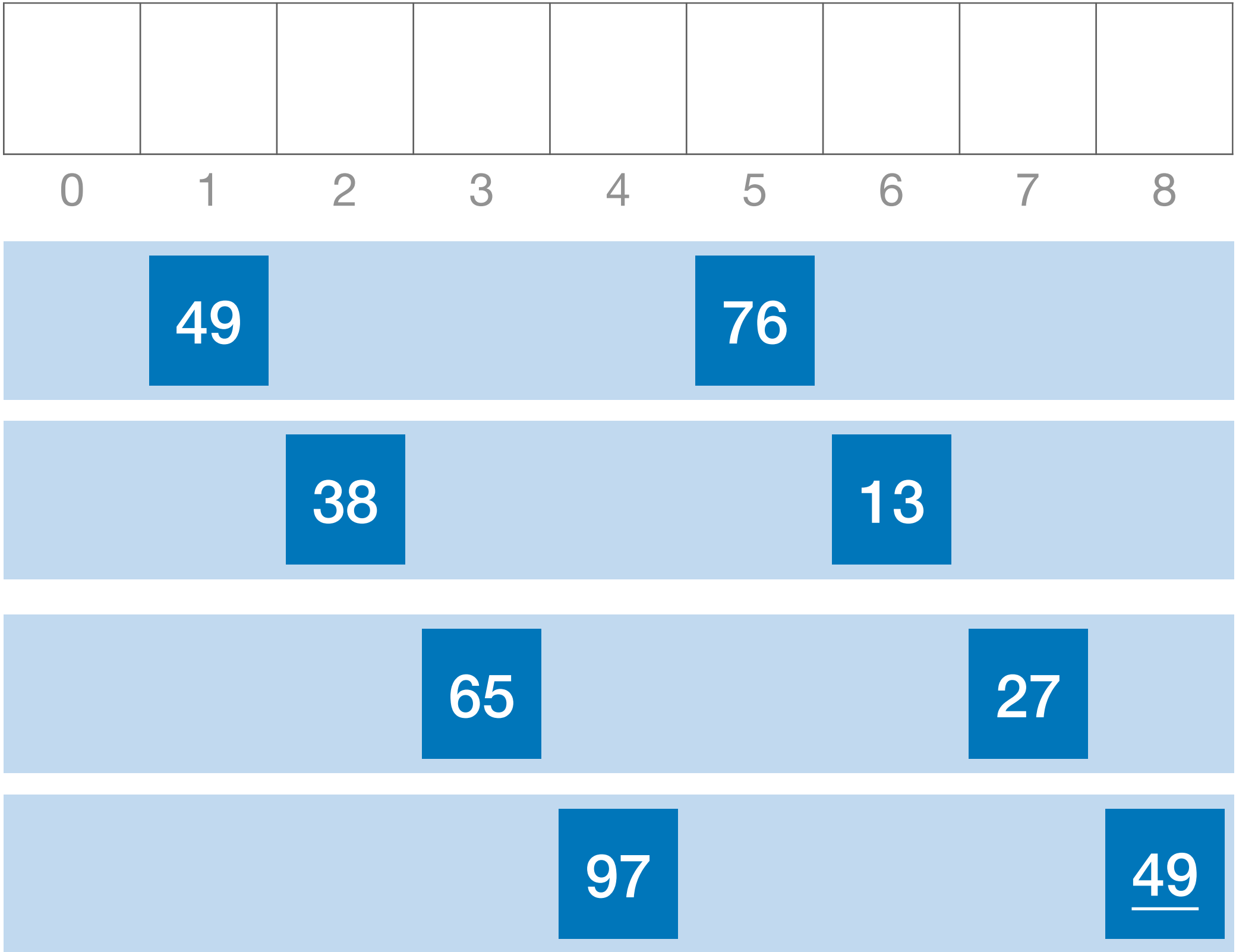
第一趟：  $d_1=n/2=4$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=n/2=4$

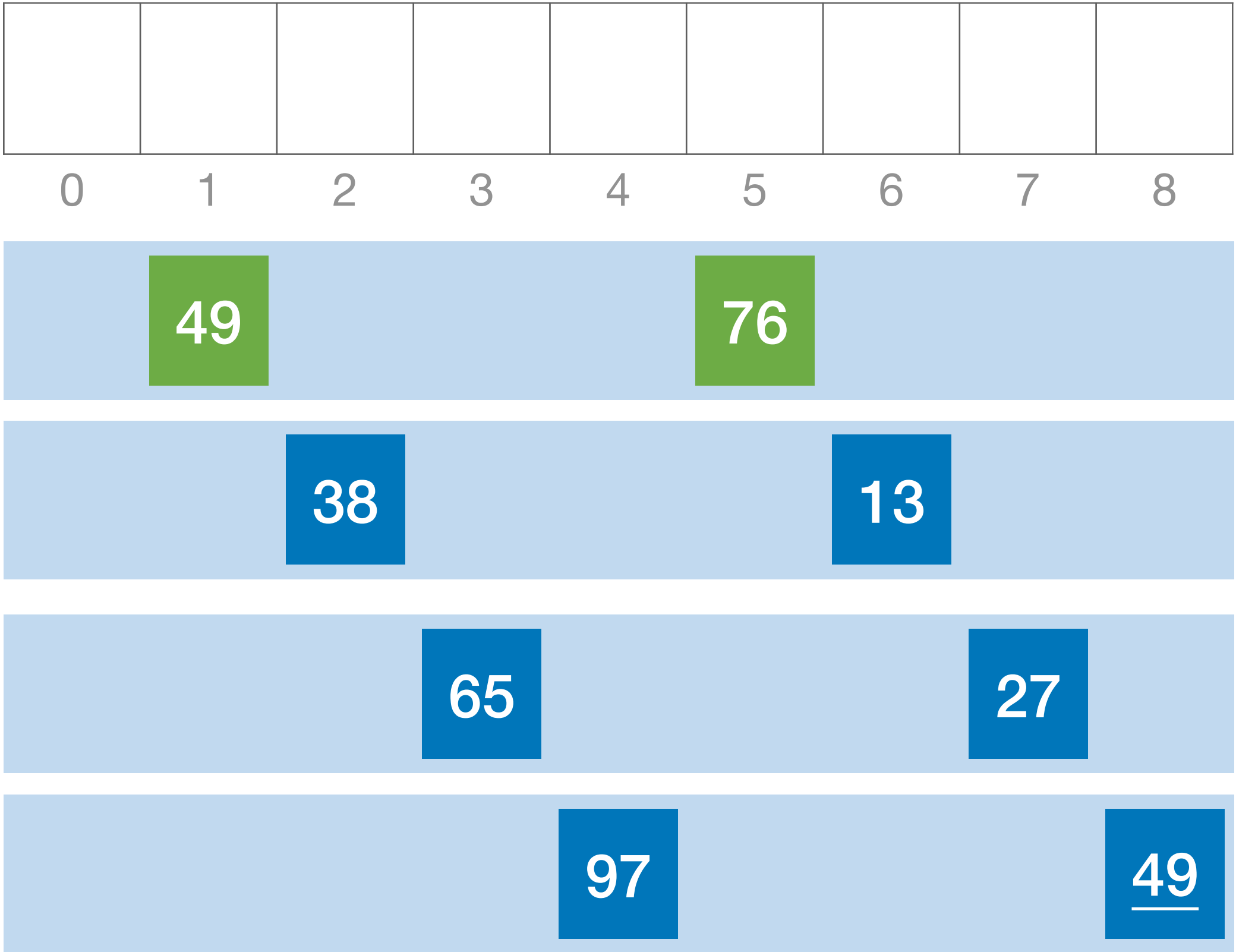


# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=n/2=4$

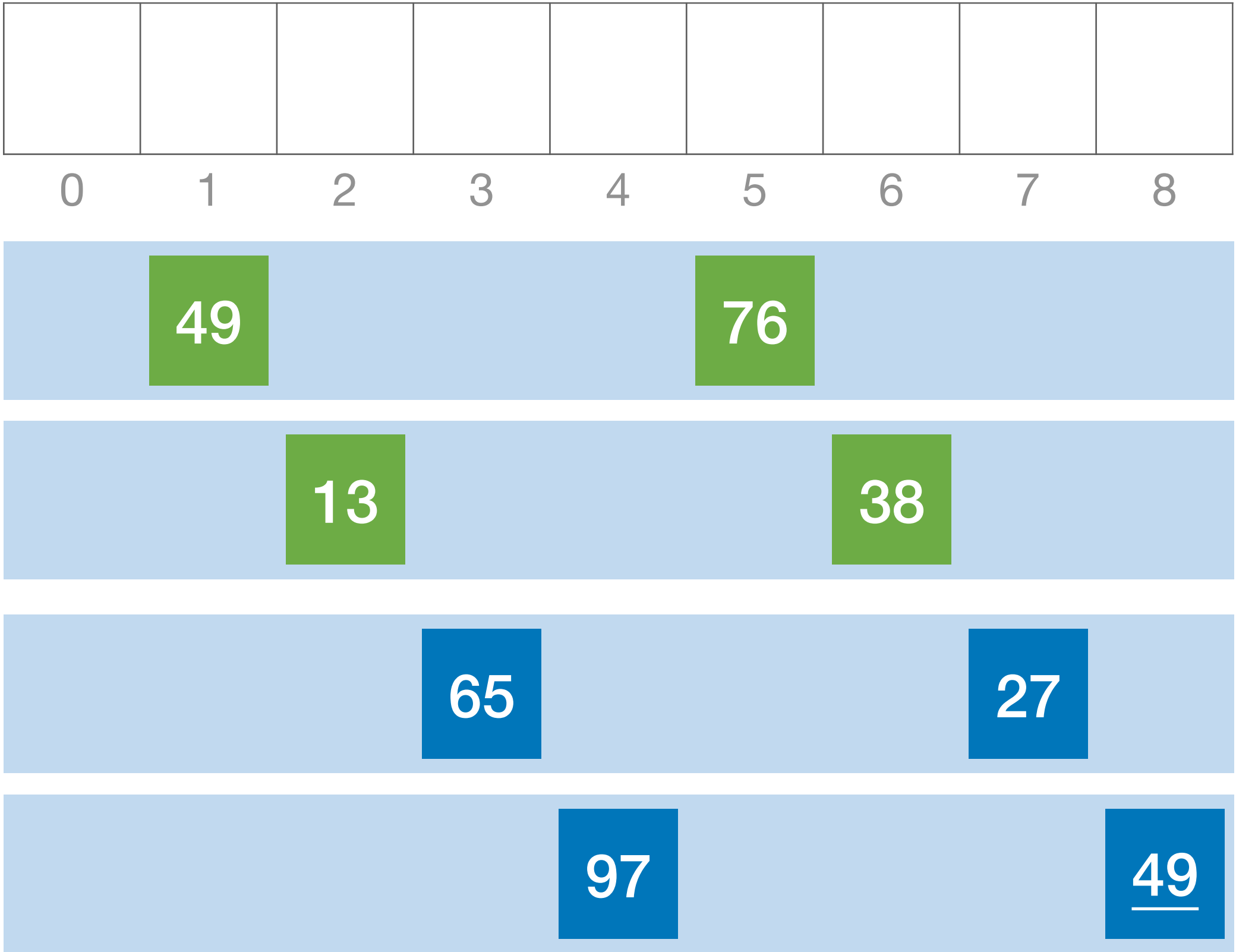


# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=n/2=4$

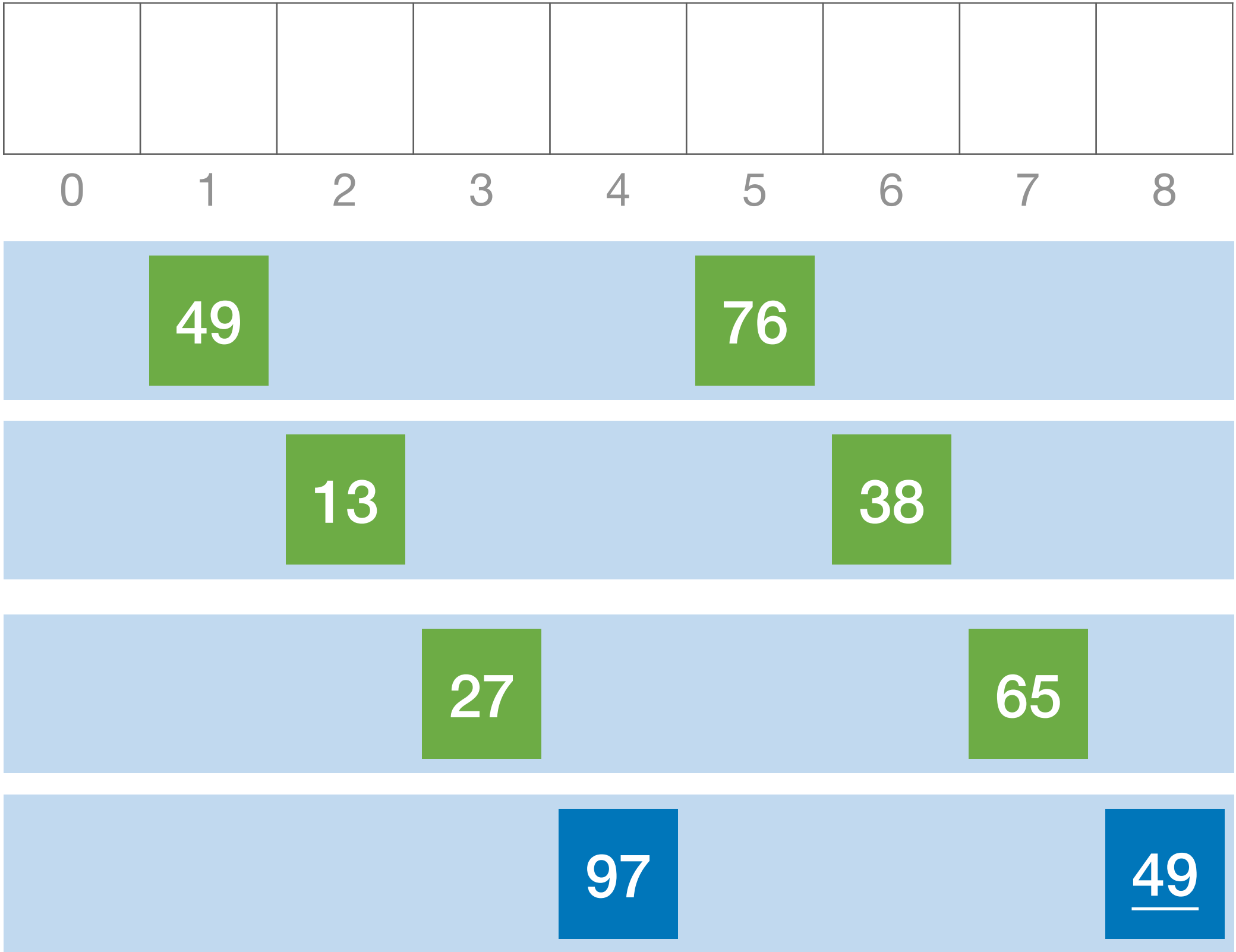


# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=n/2=4$



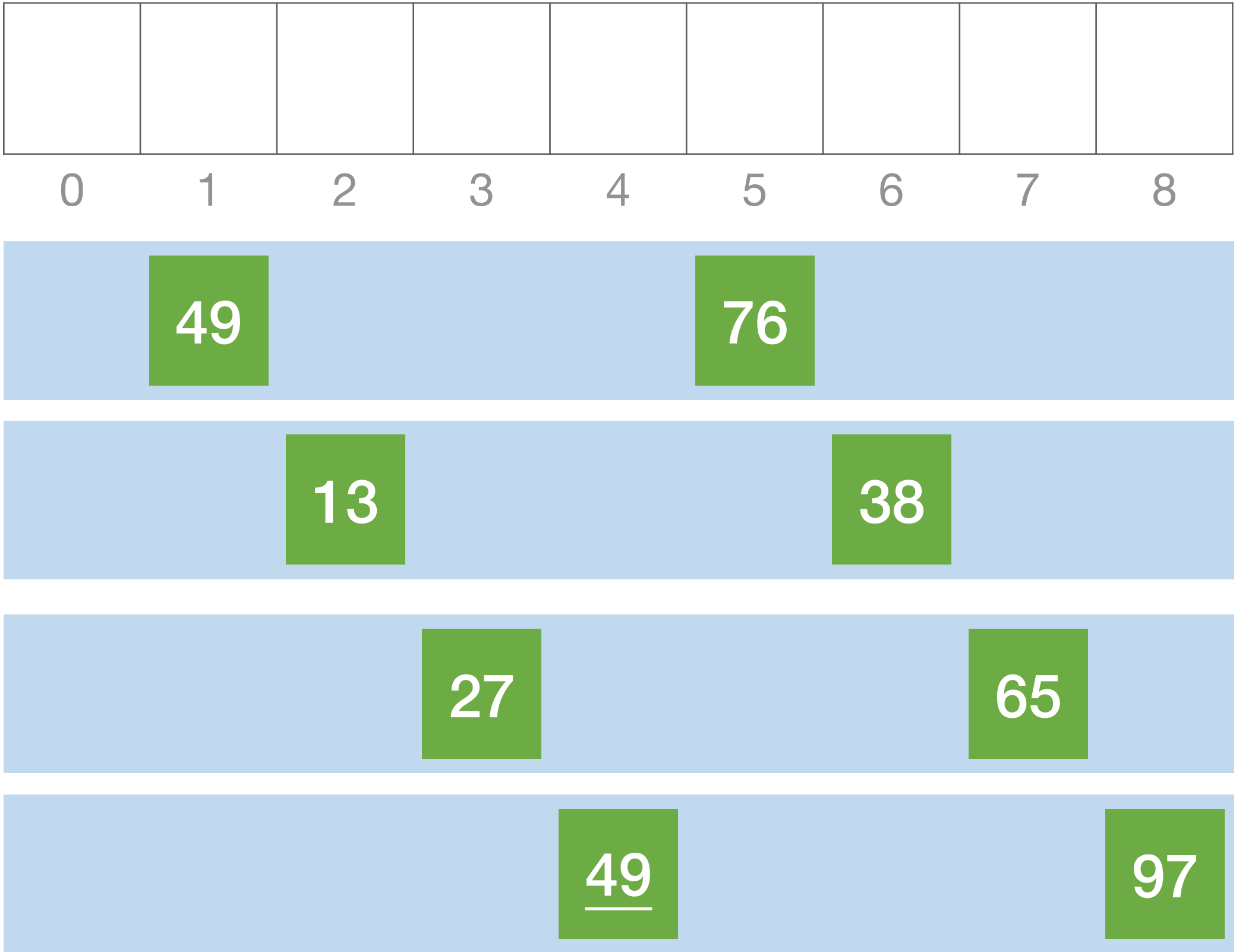
子表1  
子表2  
子表3  
子表4

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=n/2=4$

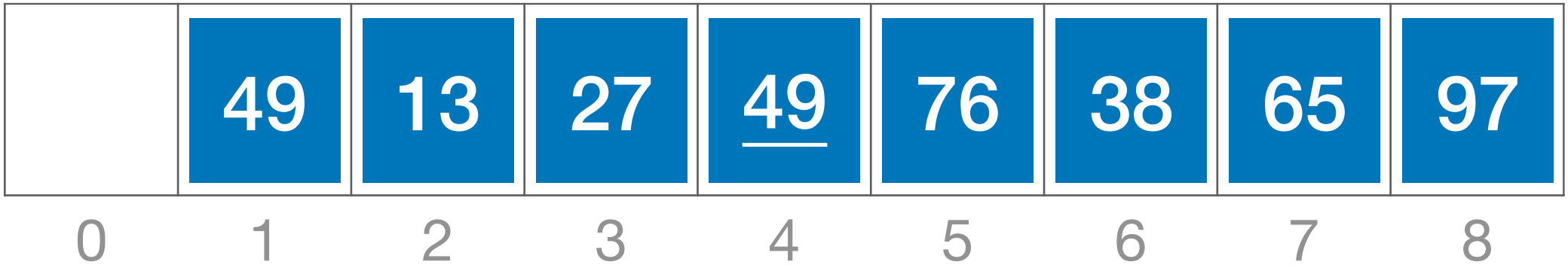




# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。



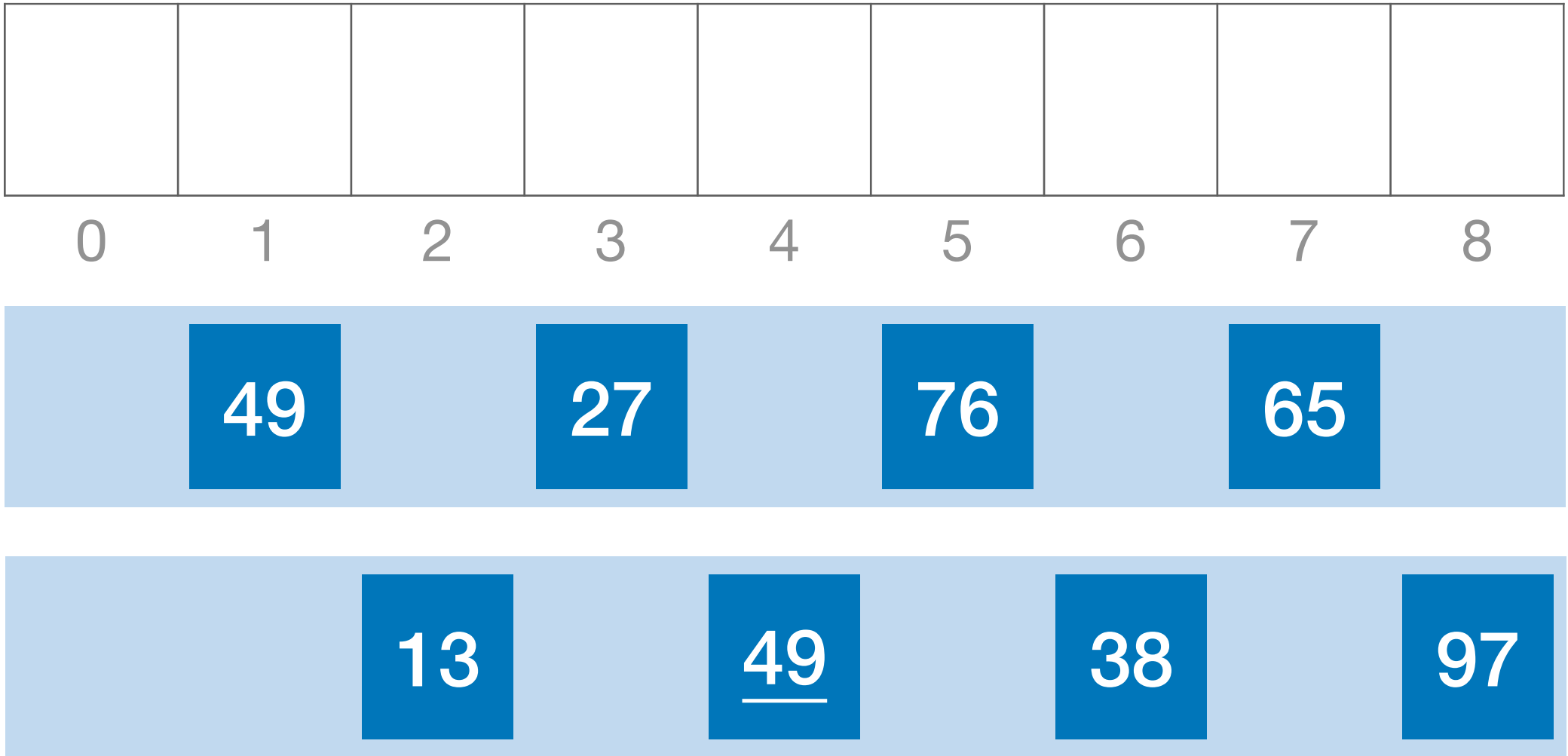
第二趟：  $d_2=d_1/2=2$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第二趟：  $d_2=d_1/2=2$



子表1

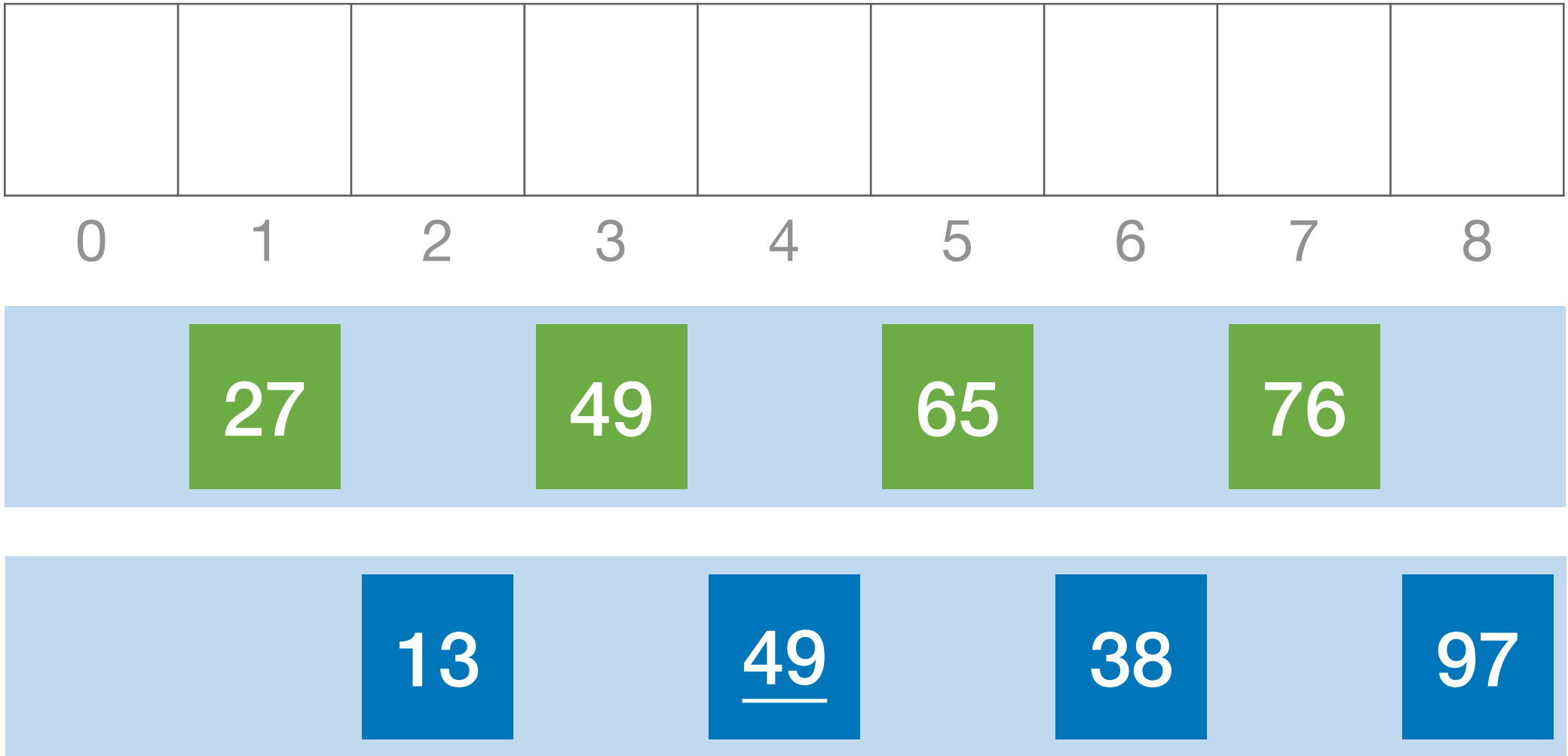
子表2

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第二趟：  $d_2=d_1/2=2$

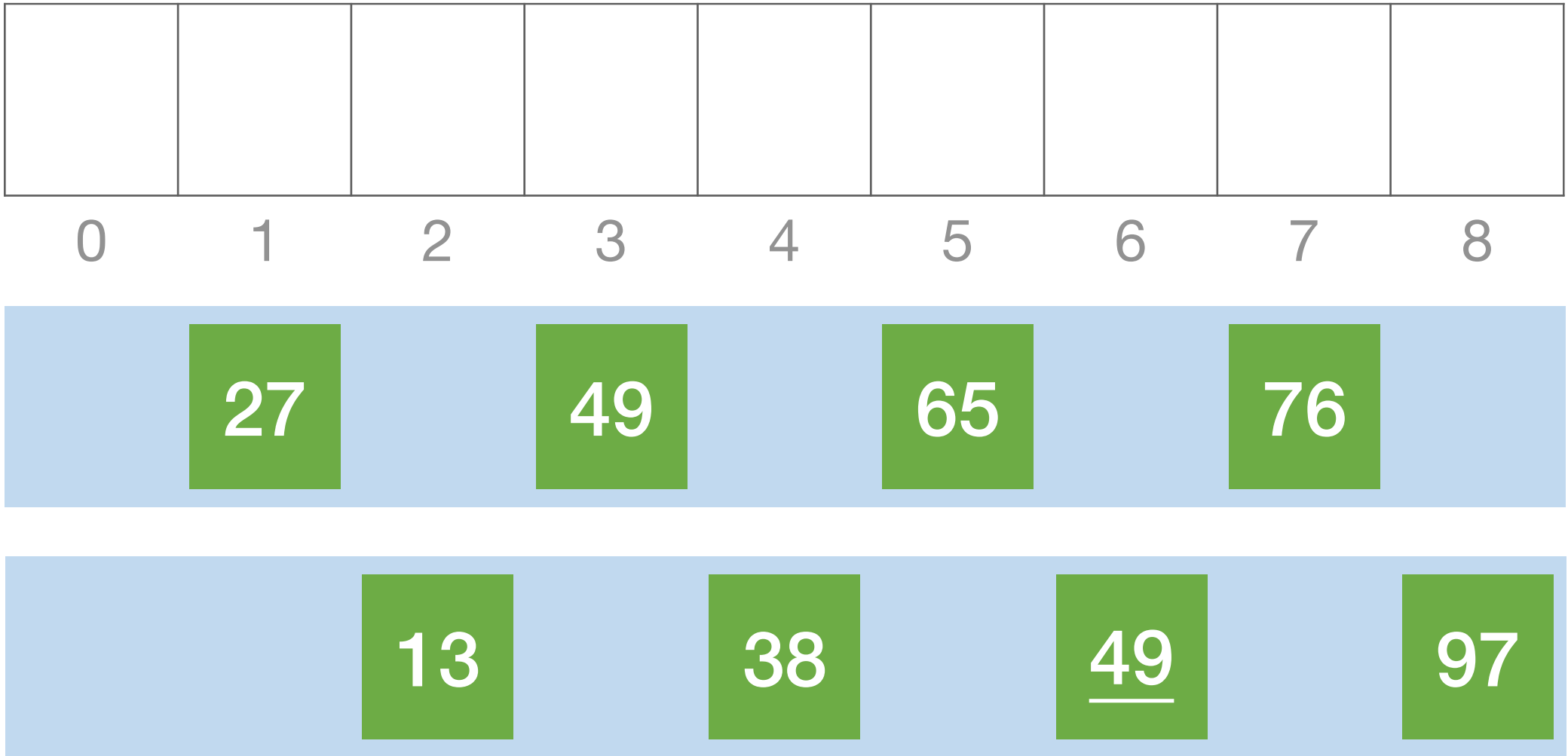


# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第二趟：  $d_2=d_1/2=2$



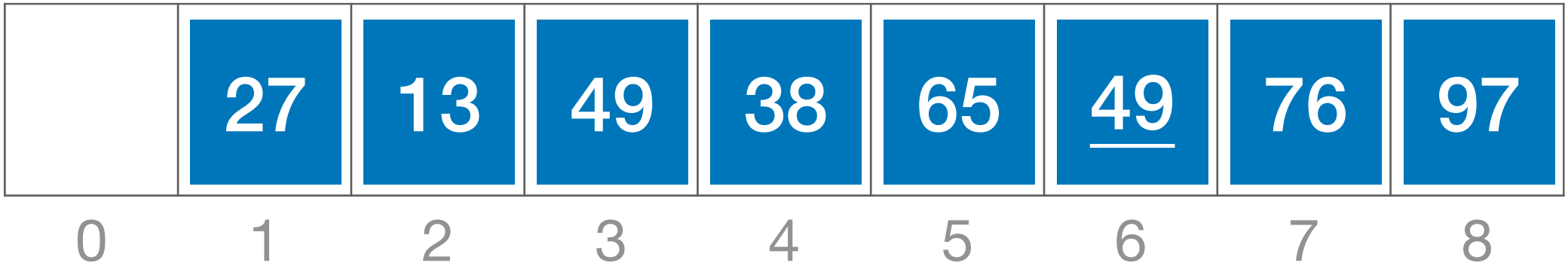
子表1

子表2

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。



第三趟：  $d_3=d_2/2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第三趟：  $d_3=d_2/2=1$

|   |    |    |    |    |    |           |    |    |
|---|----|----|----|----|----|-----------|----|----|
|   |    |    |    |    |    |           |    |    |
| 0 | 1  | 2  | 3  | 4  | 5  | 6         | 7  | 8  |
|   | 27 | 13 | 49 | 38 | 65 | <u>49</u> | 76 | 97 |

子表1

整个表已呈现出“基本有序”，对整体再进行一次“直接插入排序”

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第三趟：  $d_3=d_2/2=1$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   |    |    |    |    |           |    |    |    |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |
|   | 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |

子表1

整个表已呈现出“基本有序”，对整体再进行一次“直接插入排序”

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |



# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

希尔本人建议：每次  
将增量缩小一半

第一趟：  $d_1=n/2=4$

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

|   |    |    |    |           |    |    |    |    |
|---|----|----|----|-----------|----|----|----|----|
|   | 49 | 13 | 27 | <u>49</u> | 76 | 38 | 65 | 97 |
| 0 | 1  | 2  | 3  | 4         | 5  | 6  | 7  | 8  |

第二趟：  $d_2=d_1/2=2$

|   |    |    |    |    |    |           |    |    |
|---|----|----|----|----|----|-----------|----|----|
|   | 27 | 13 | 49 | 38 | 65 | <u>49</u> | 76 | 97 |
| 0 | 1  | 2  | 3  | 4  | 5  | 6         | 7  | 8  |

第三趟：  $d_3=d_2/2=1$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

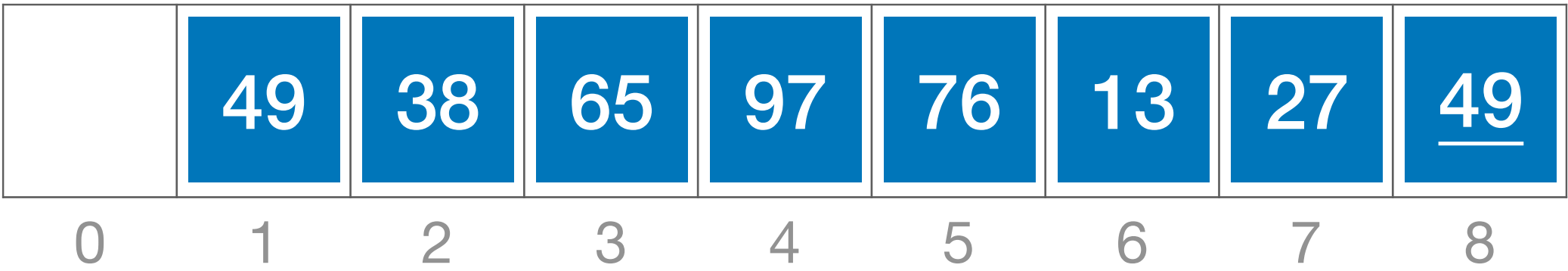
|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第二趟：  $d_2=1$

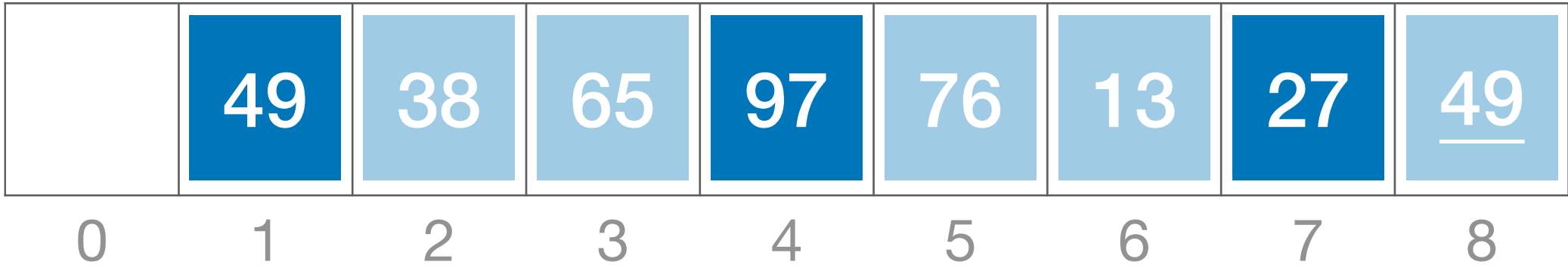
# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。



第一趟：  $d_1=3$



第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 27 | 38 | 65 | 49 | 76 | 13 | 97 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 27 | 38 | 65 | 49 | 76 | 13 | 97 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 27 | 38 | 65 | 49 | <u>49</u> | 13 | 97 | 76 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

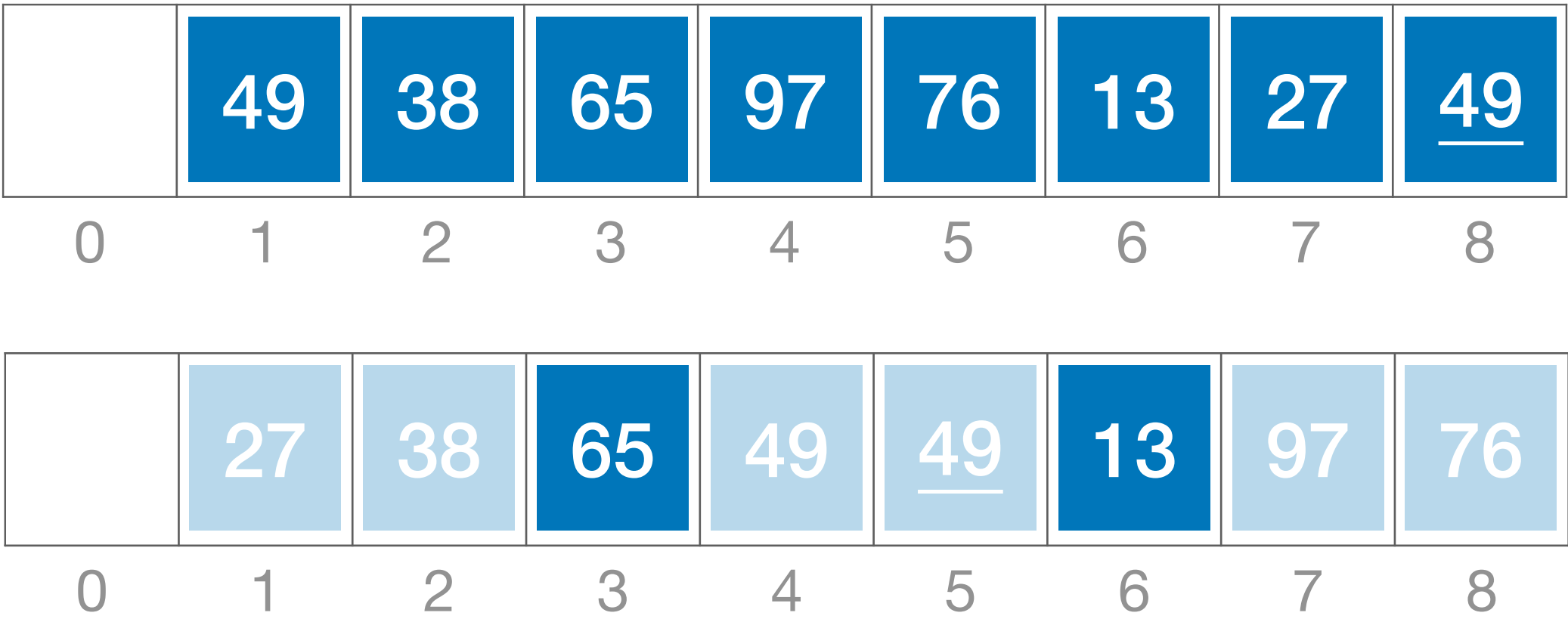
第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

第一趟：  $d_1=3$



第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 27 | 38 | 13 | 49 | <u>49</u> | 65 | 97 | 76 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

第二趟：  $d_2=1$



# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟：  $d_1=3$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 27 | 38 | 13 | 49 | <u>49</u> | 65 | 97 | 76 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

第二趟：  $d_2=1$

# 希尔排序 (Shell Sort)



希尔排序：先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量 $d$ ，重复上述过程，直到 $d=1$ 为止。

考试中可能遇到各种增量

第一趟：  $d_1=3$

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 27 | 38 | 13 | 49 | <u>49</u> | 65 | 97 | 76 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

第二趟：  $d_2=1$

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

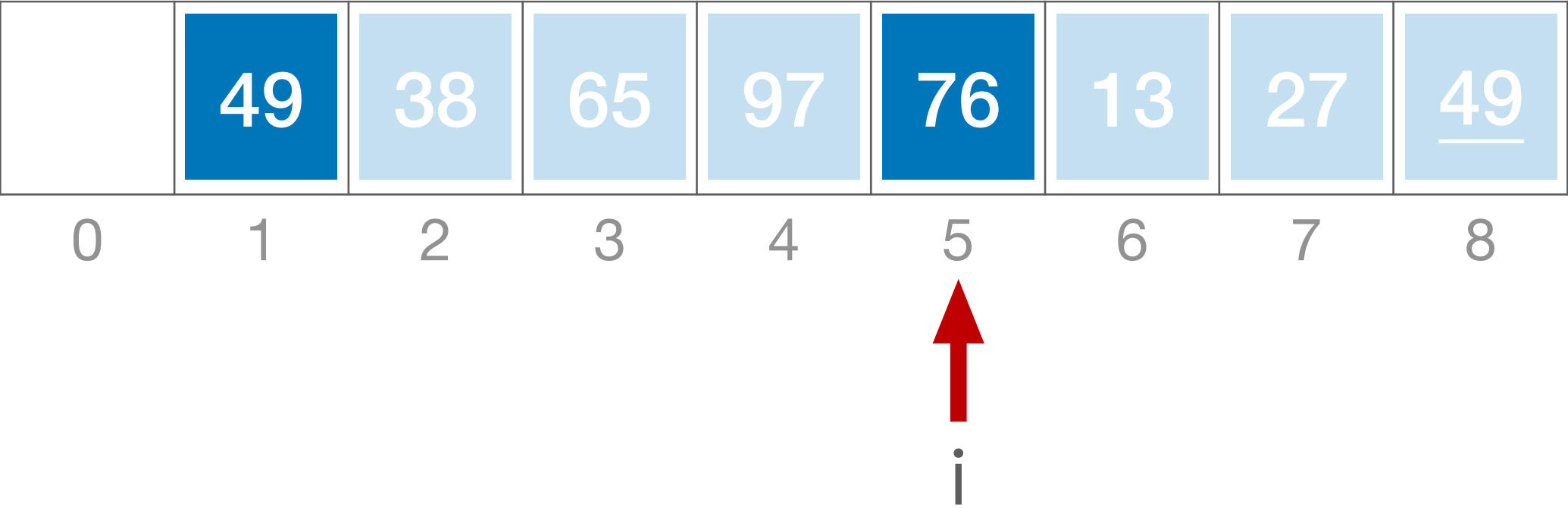
第一趟：d=n/2=4

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

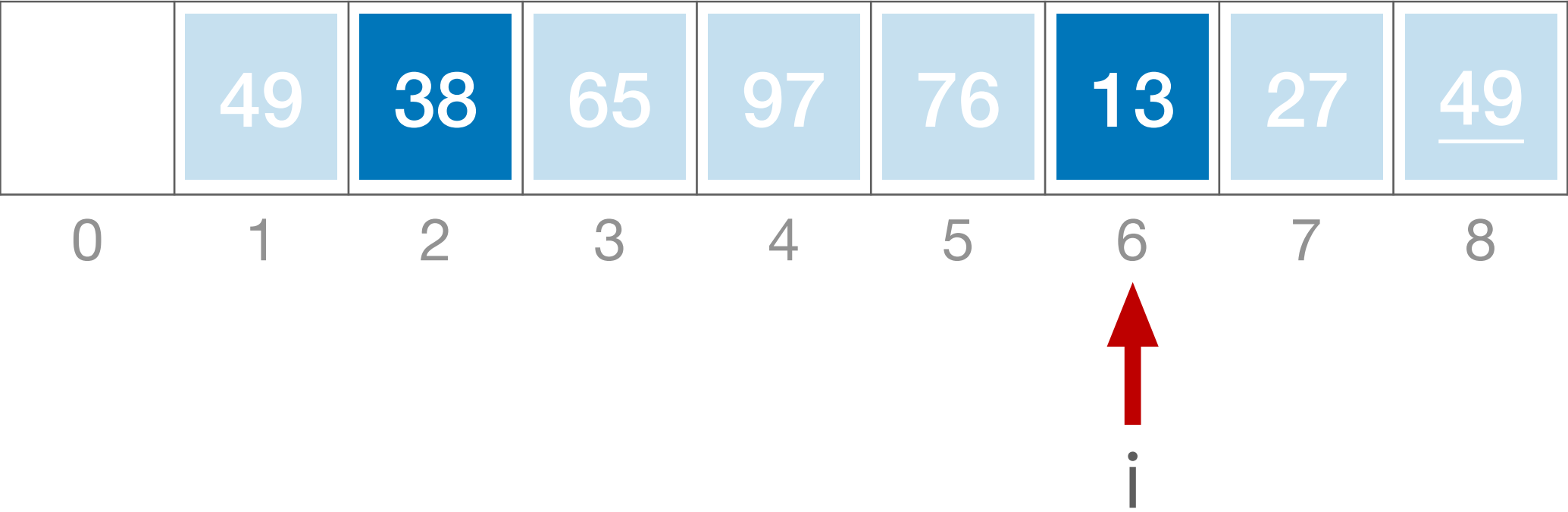
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

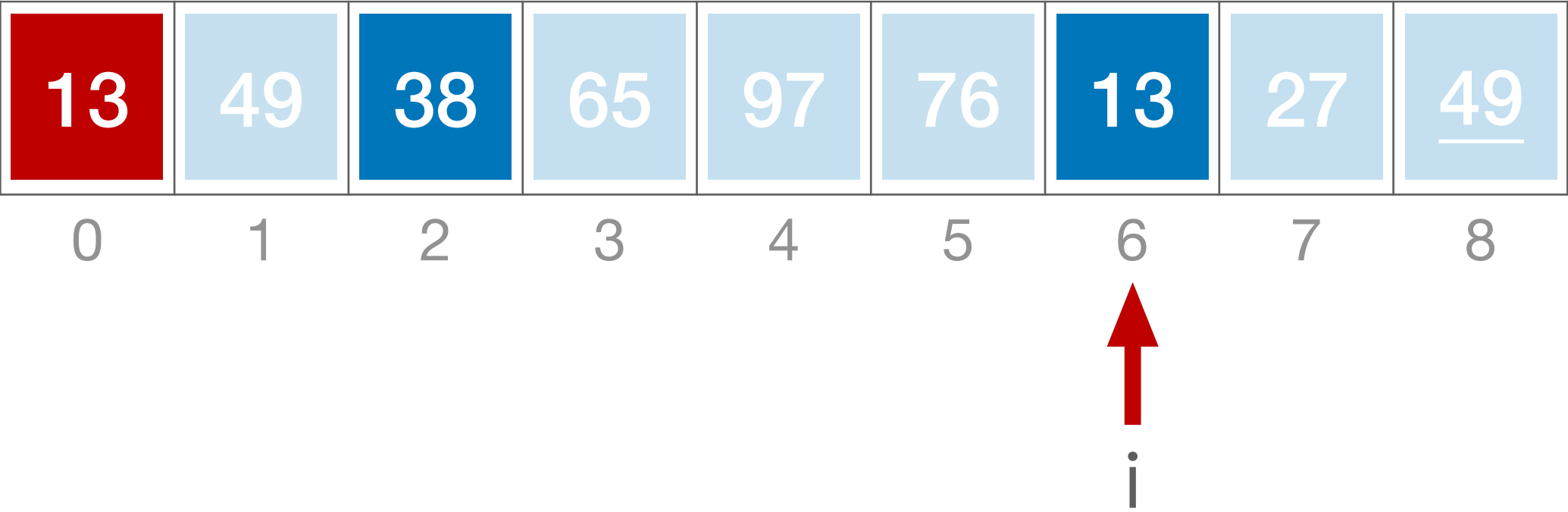
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

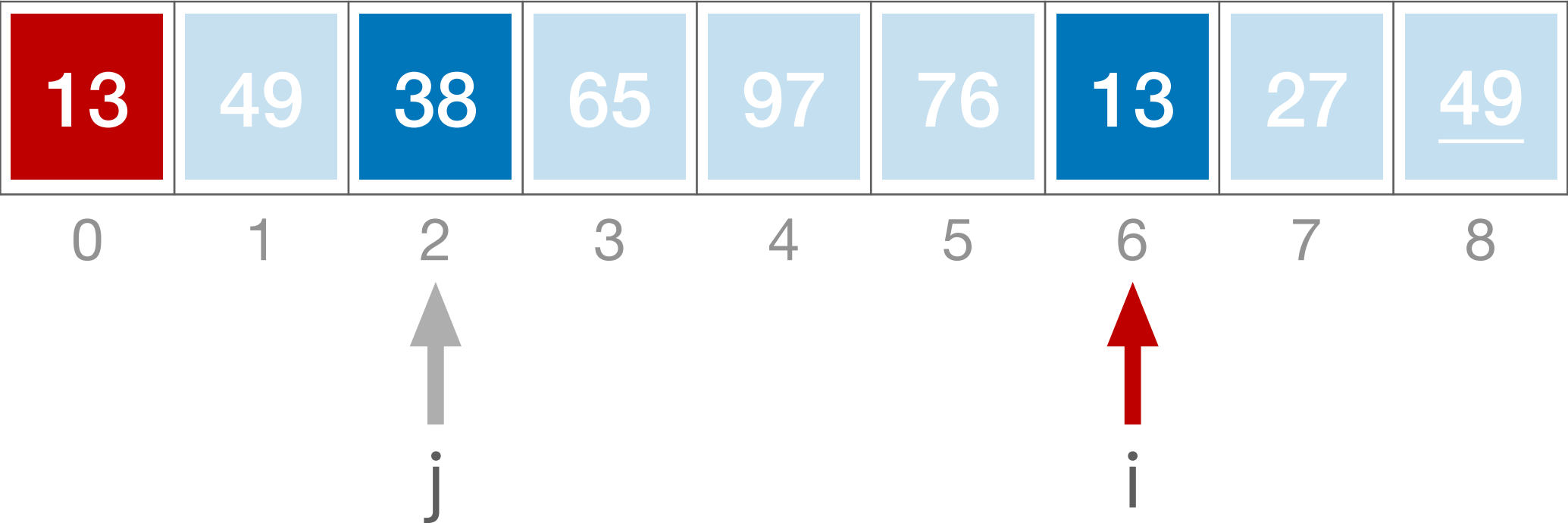
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4



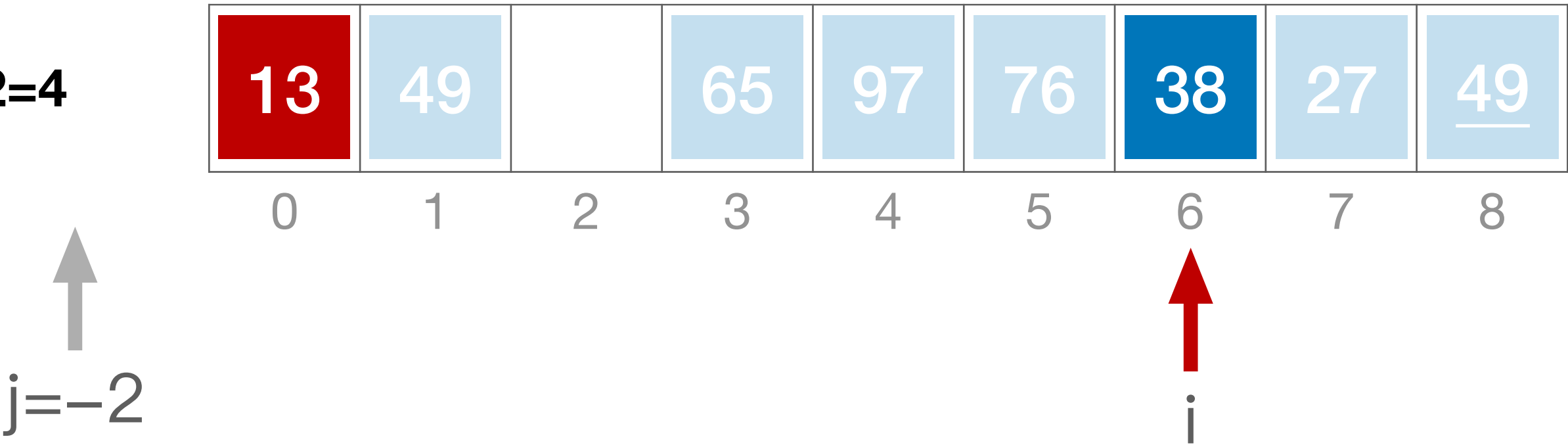




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4

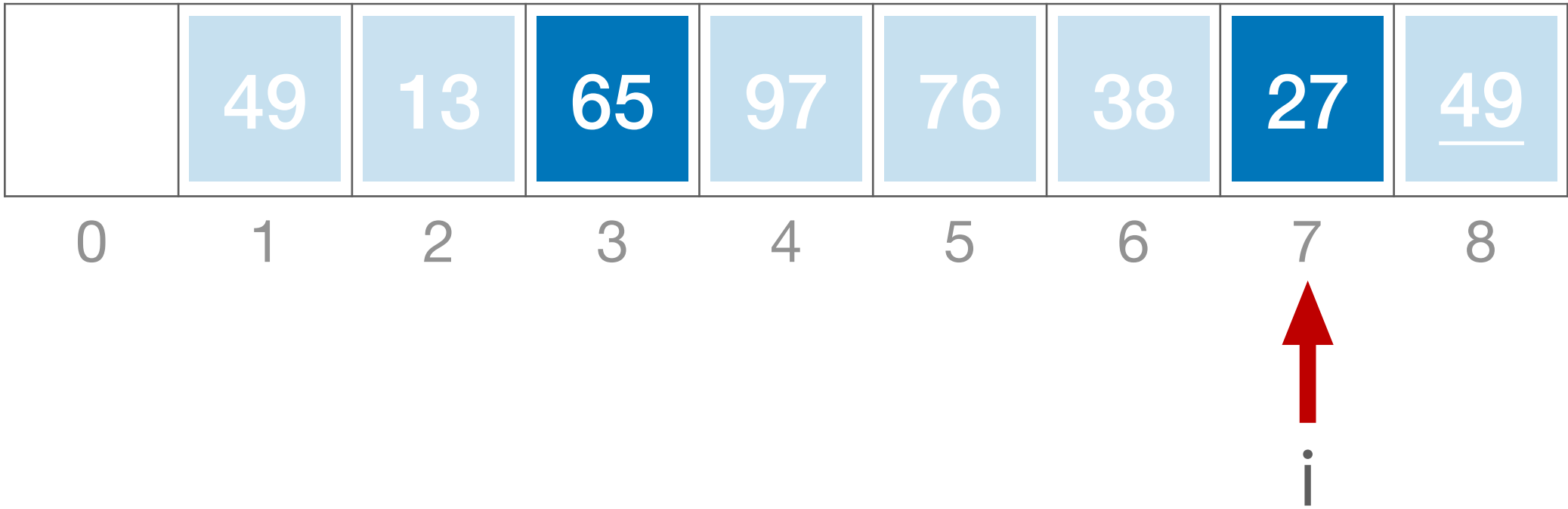




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

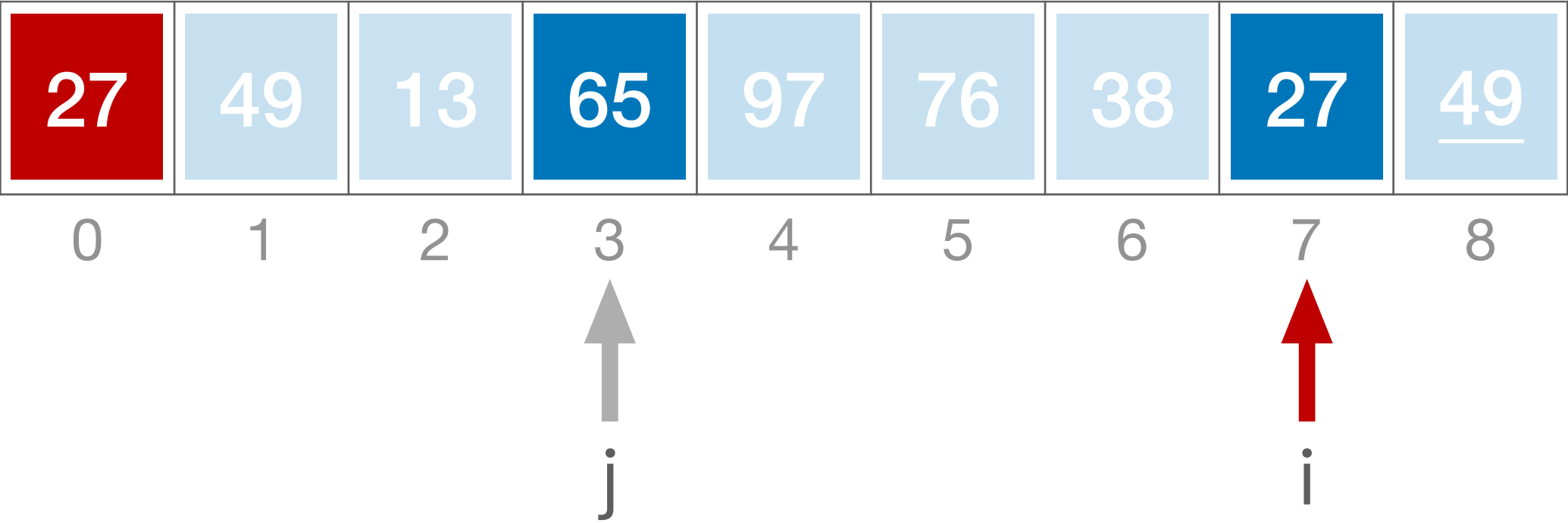
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4



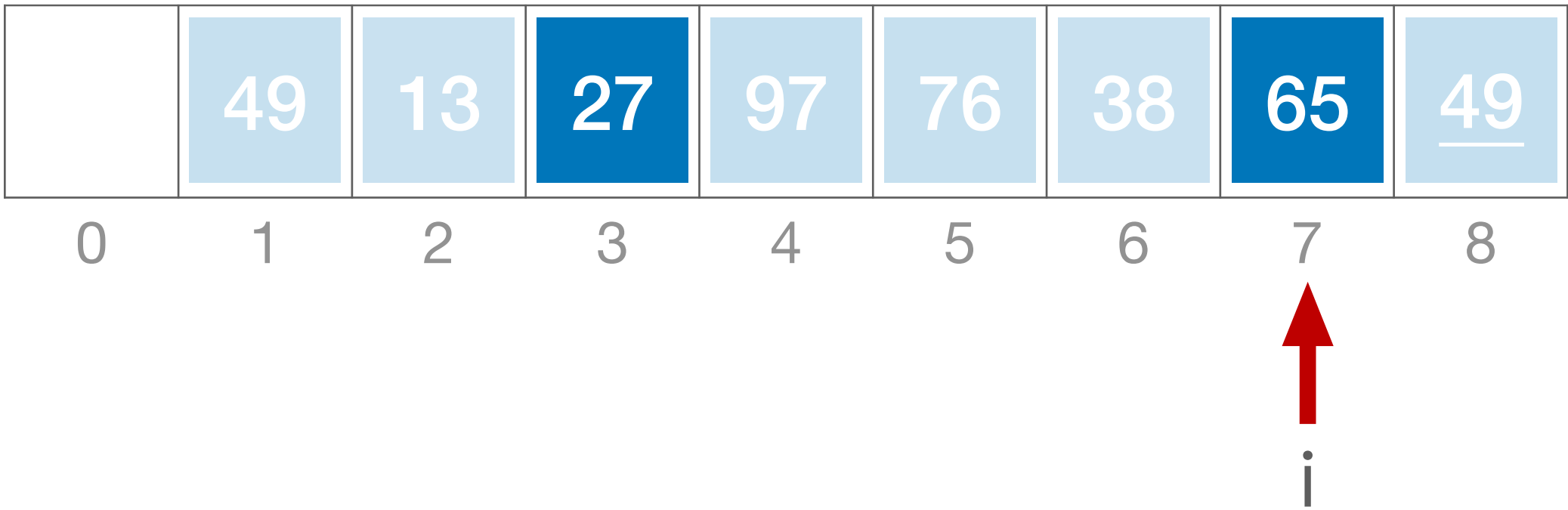




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4

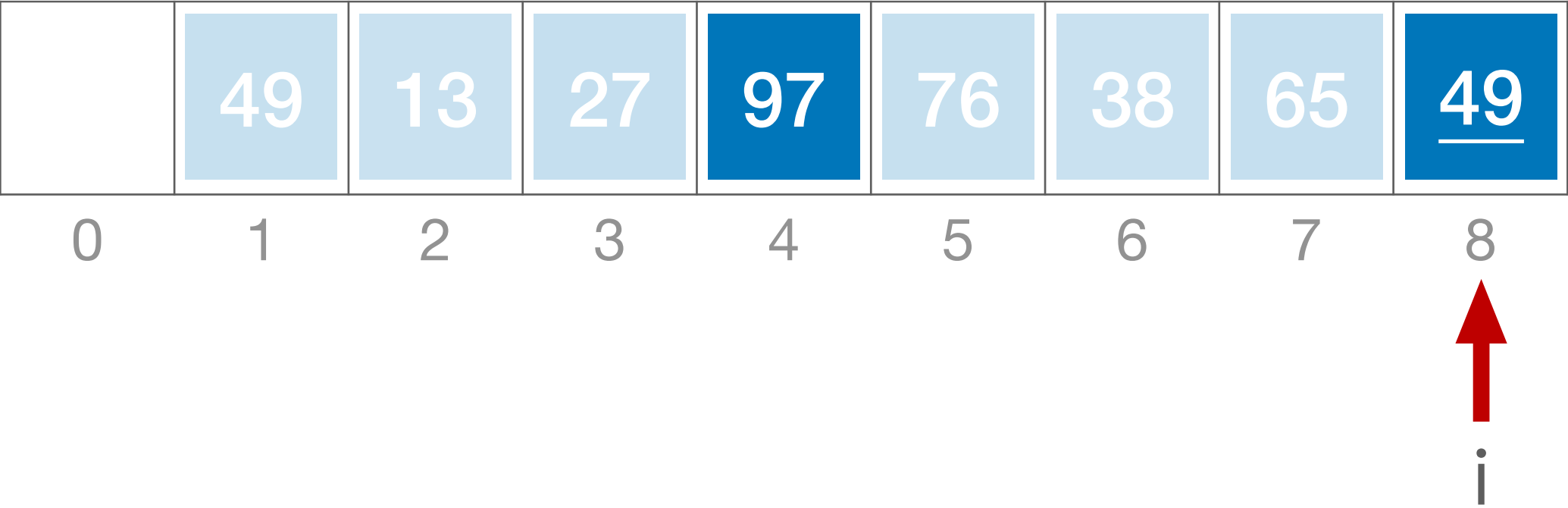




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4





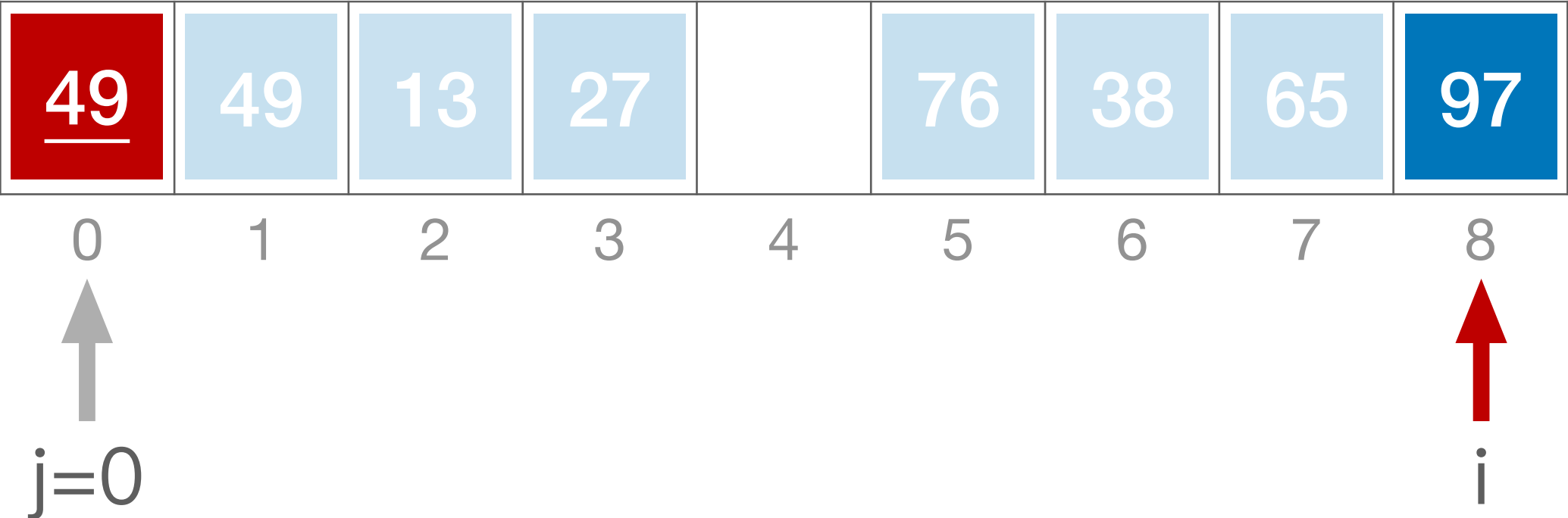




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

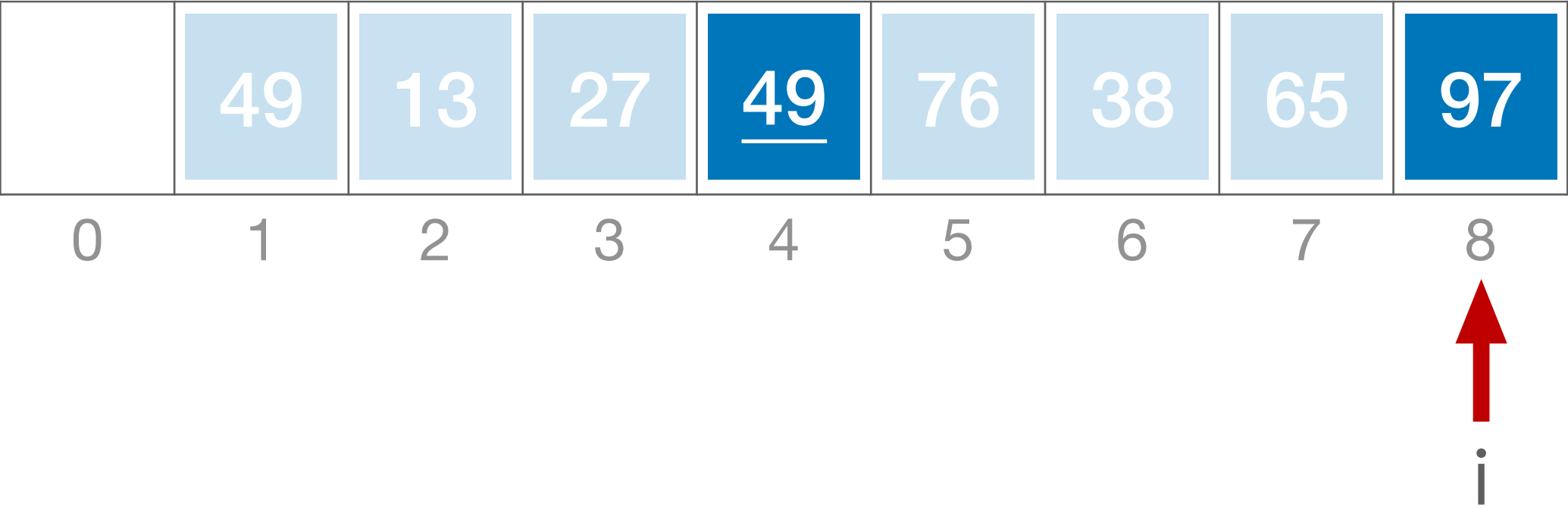
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

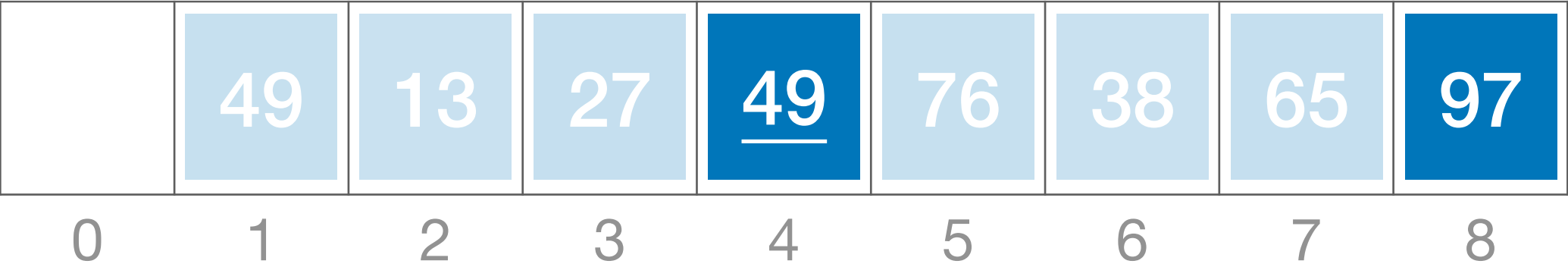
第一趟：d=n/2=4



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第一趟：d=n/2=4



↑  
i

# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

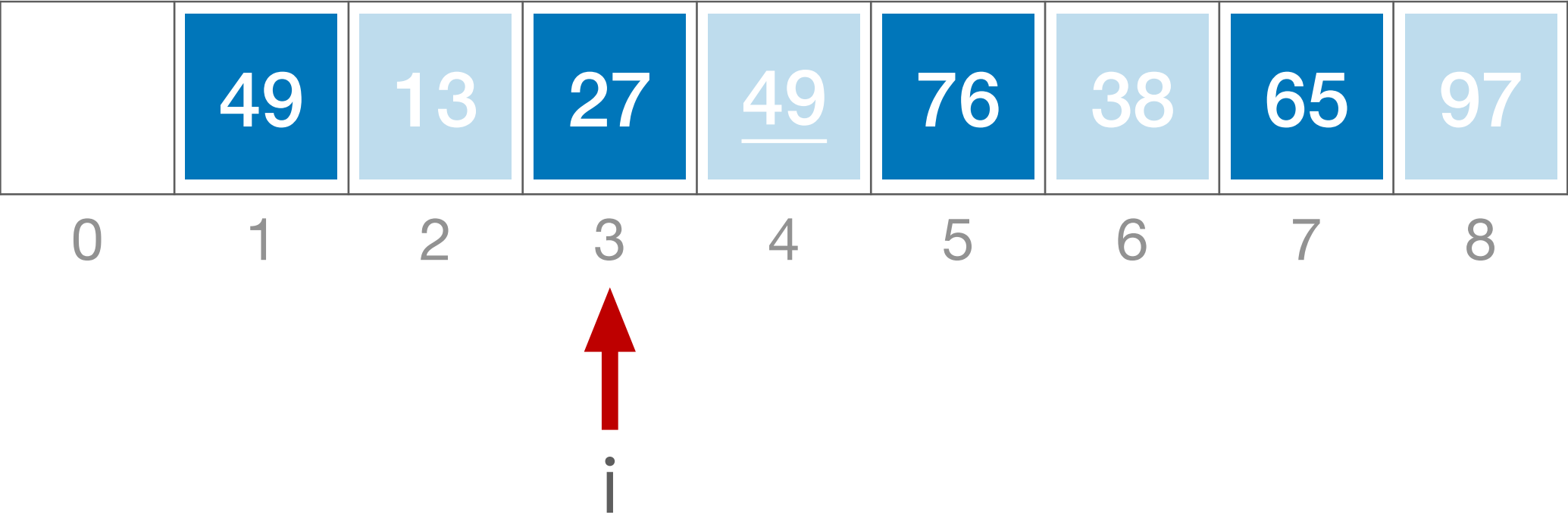
第二趟： d=2

|   |    |    |    |           |    |    |    |    |
|---|----|----|----|-----------|----|----|----|----|
|   | 49 | 13 | 27 | <u>49</u> | 76 | 38 | 65 | 97 |
| 0 | 1  | 2  | 3  | 4         | 5  | 6  | 7  | 8  |

# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2

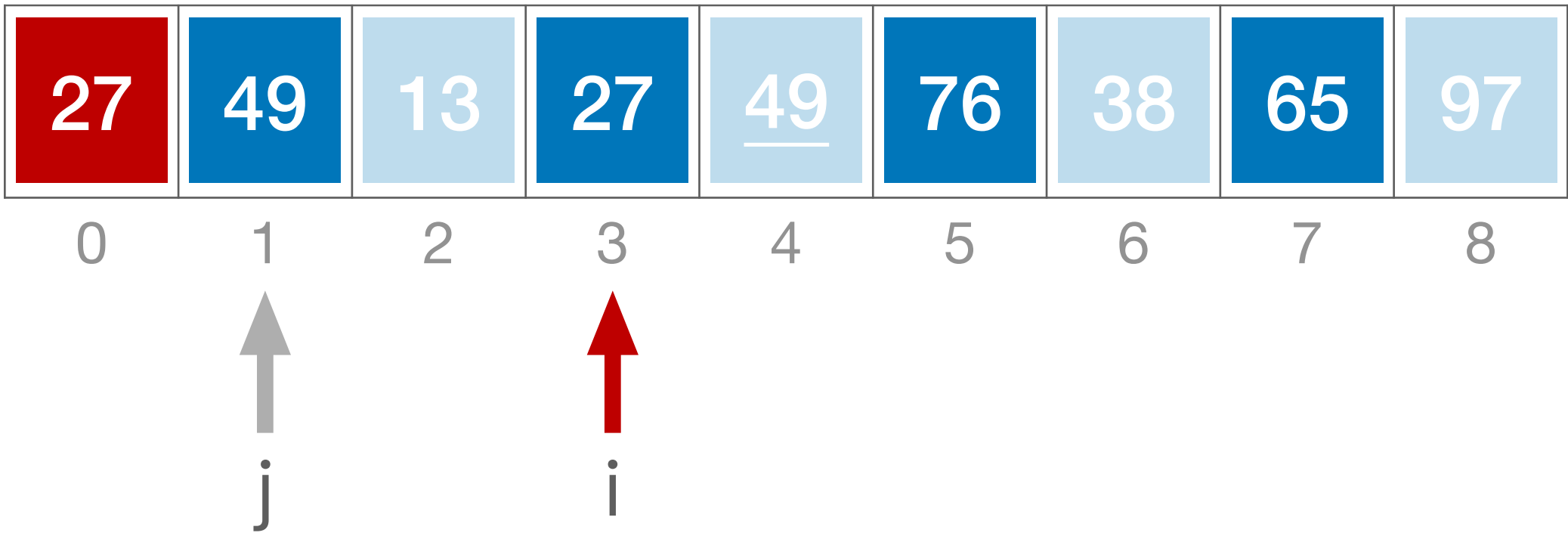




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2

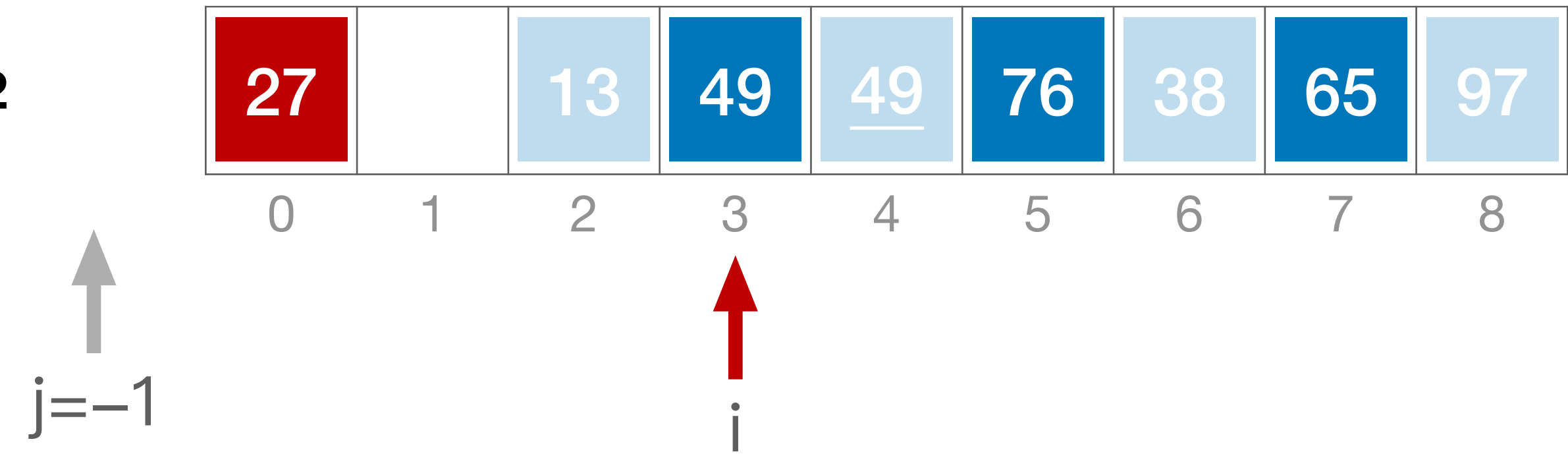




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

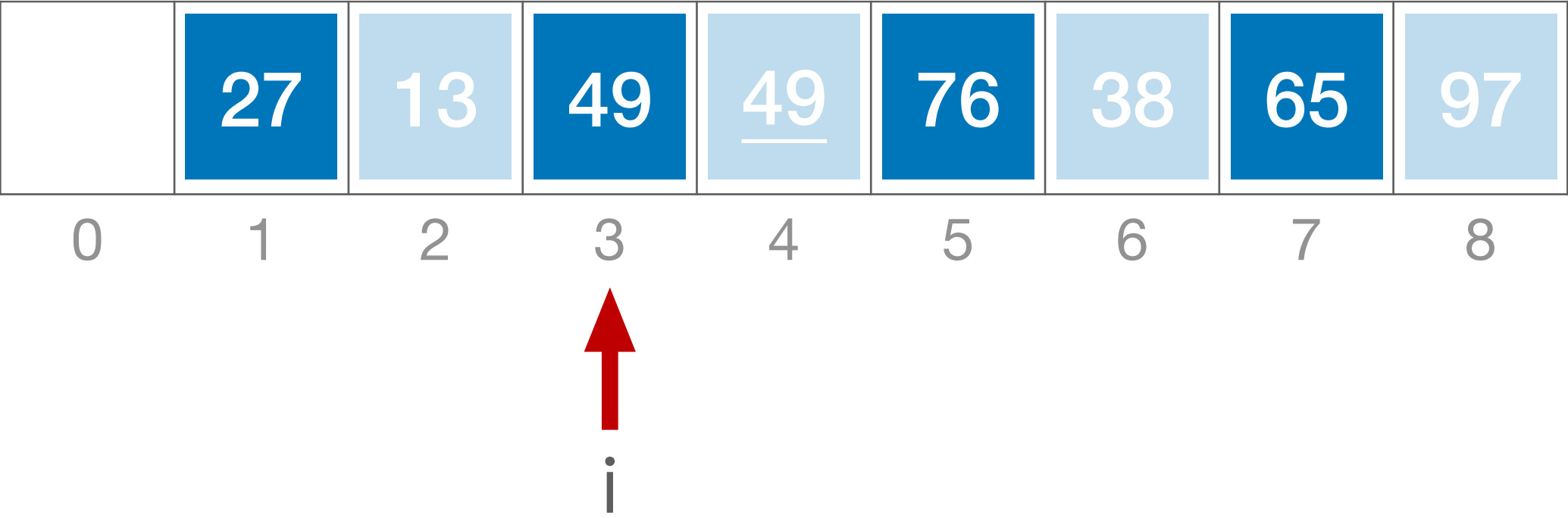
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

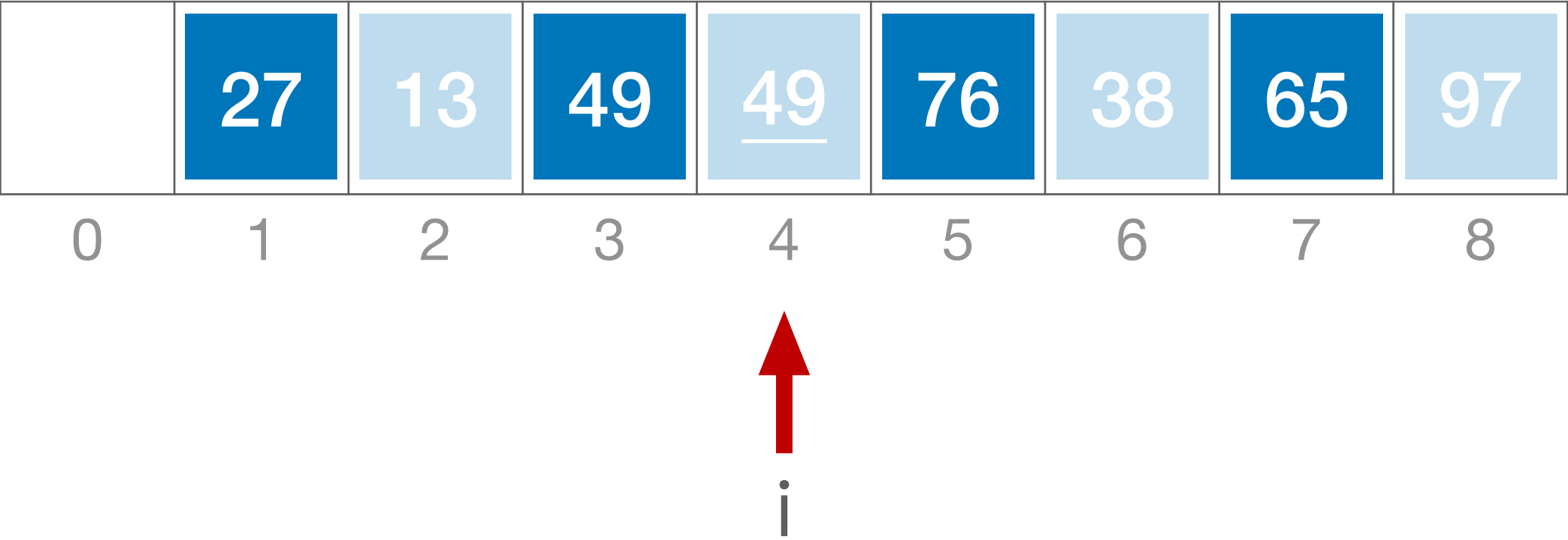
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

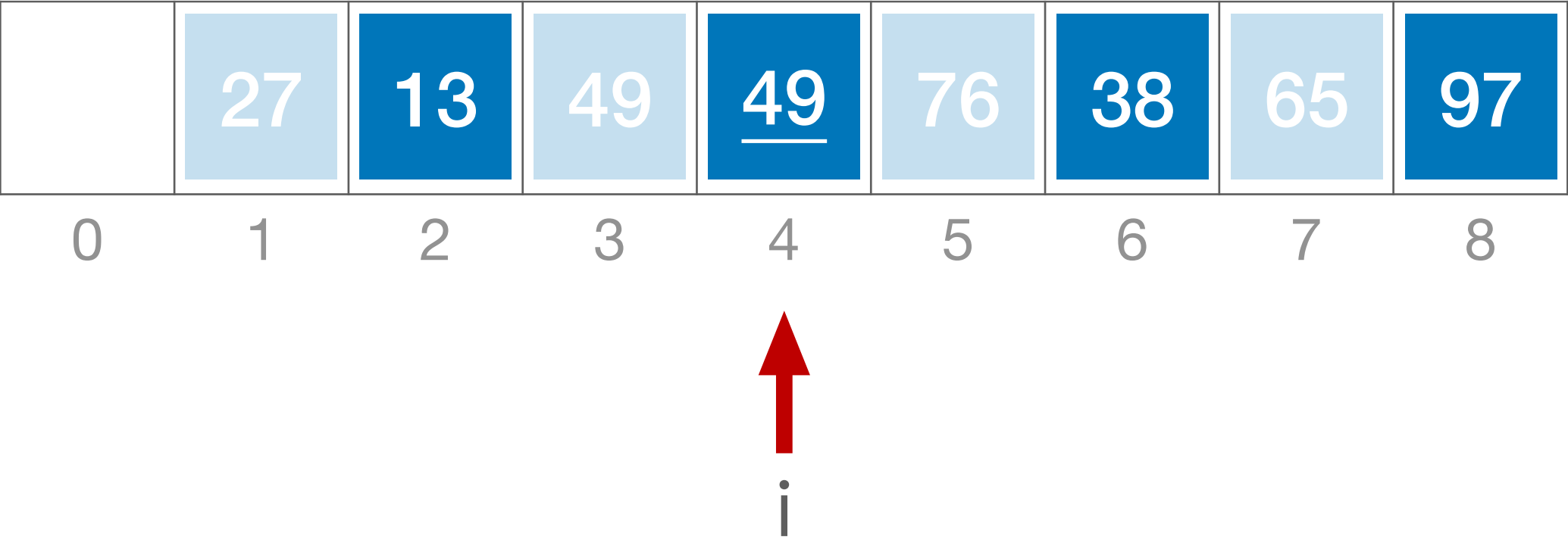
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

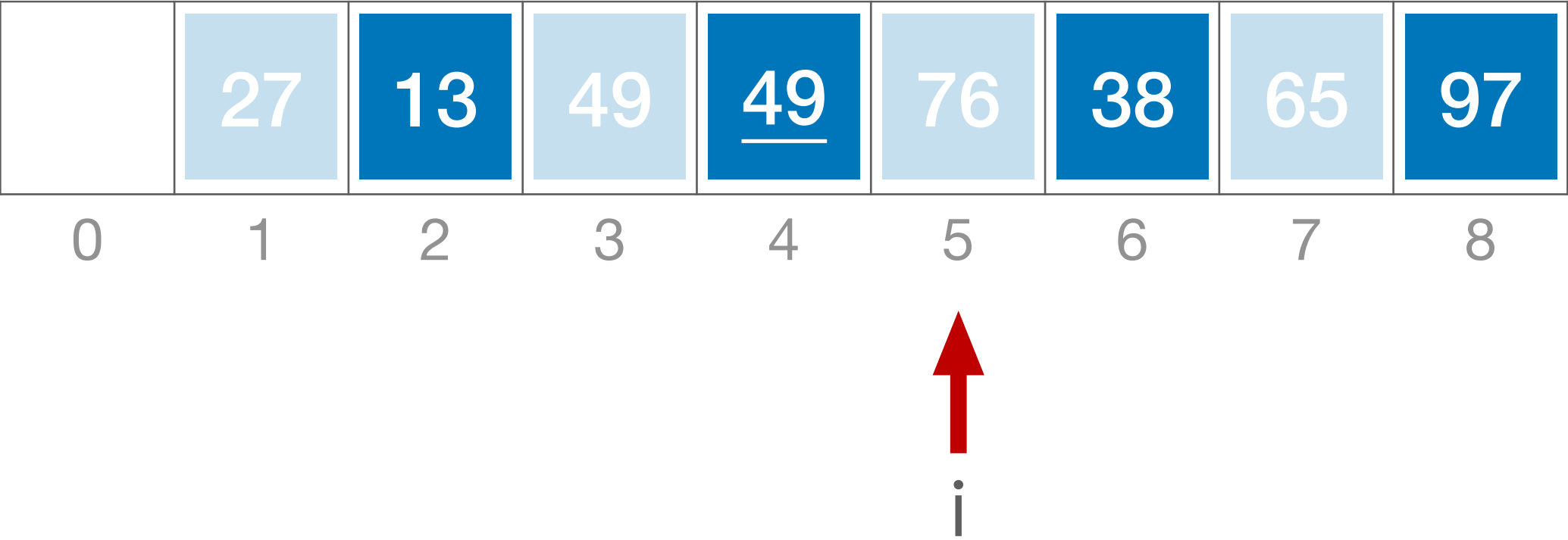
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

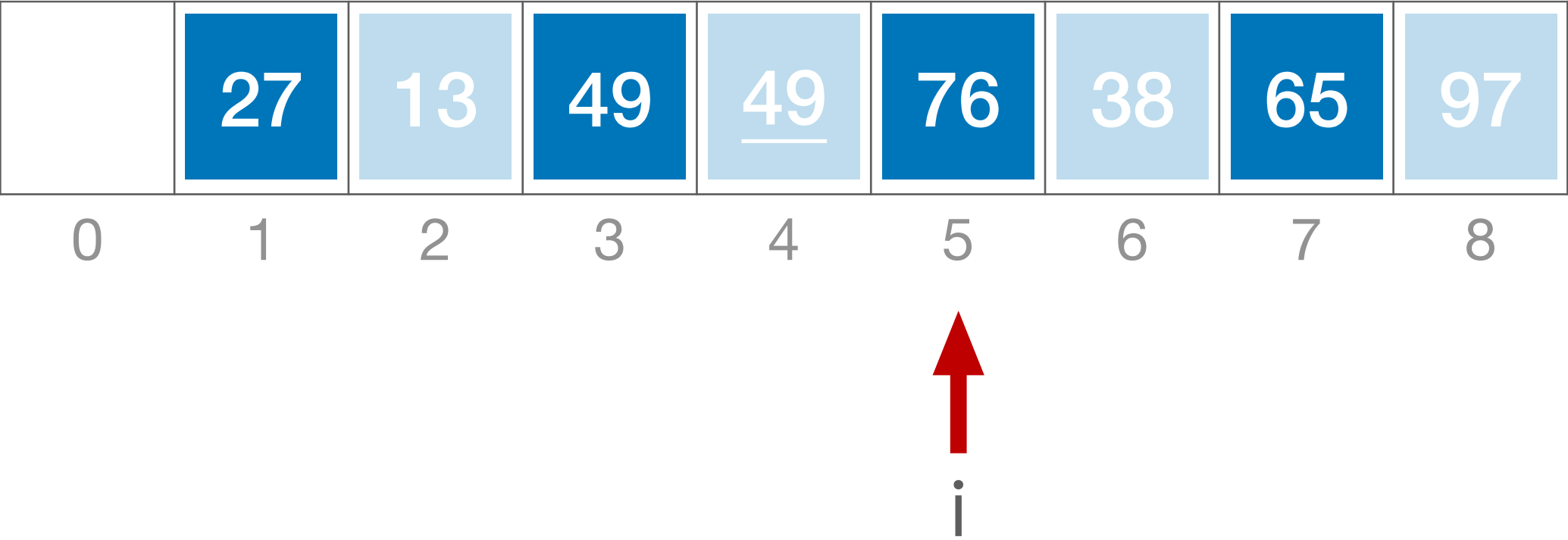
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

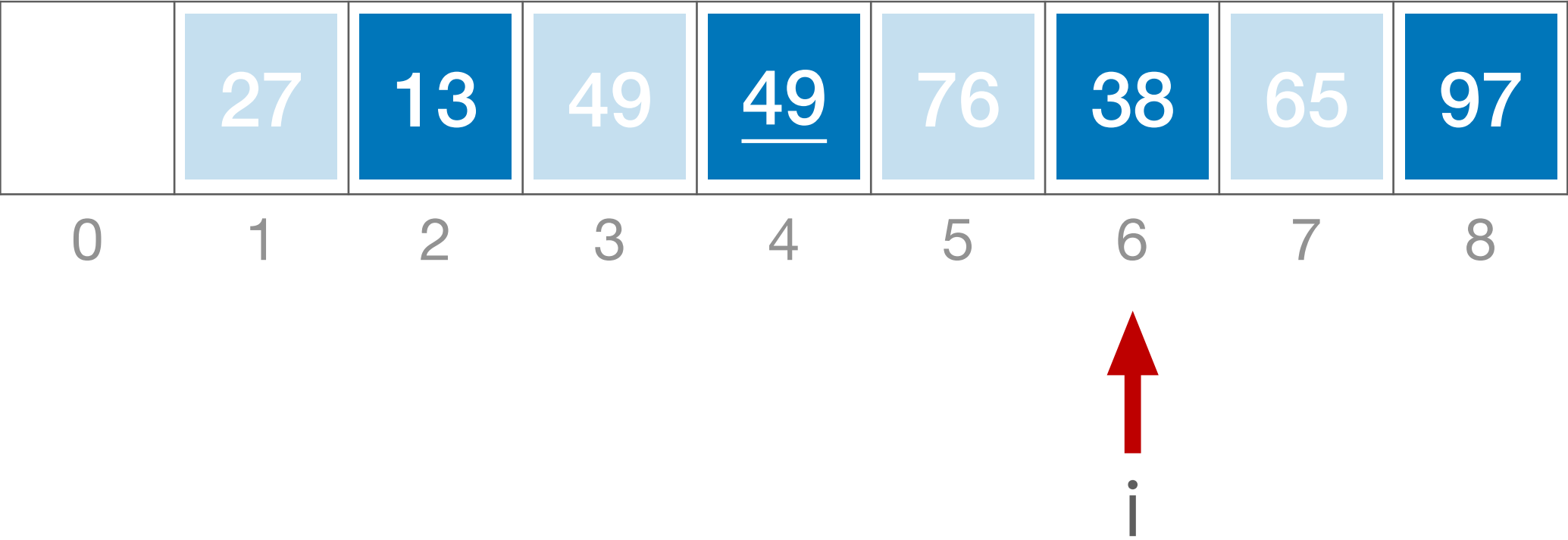
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2

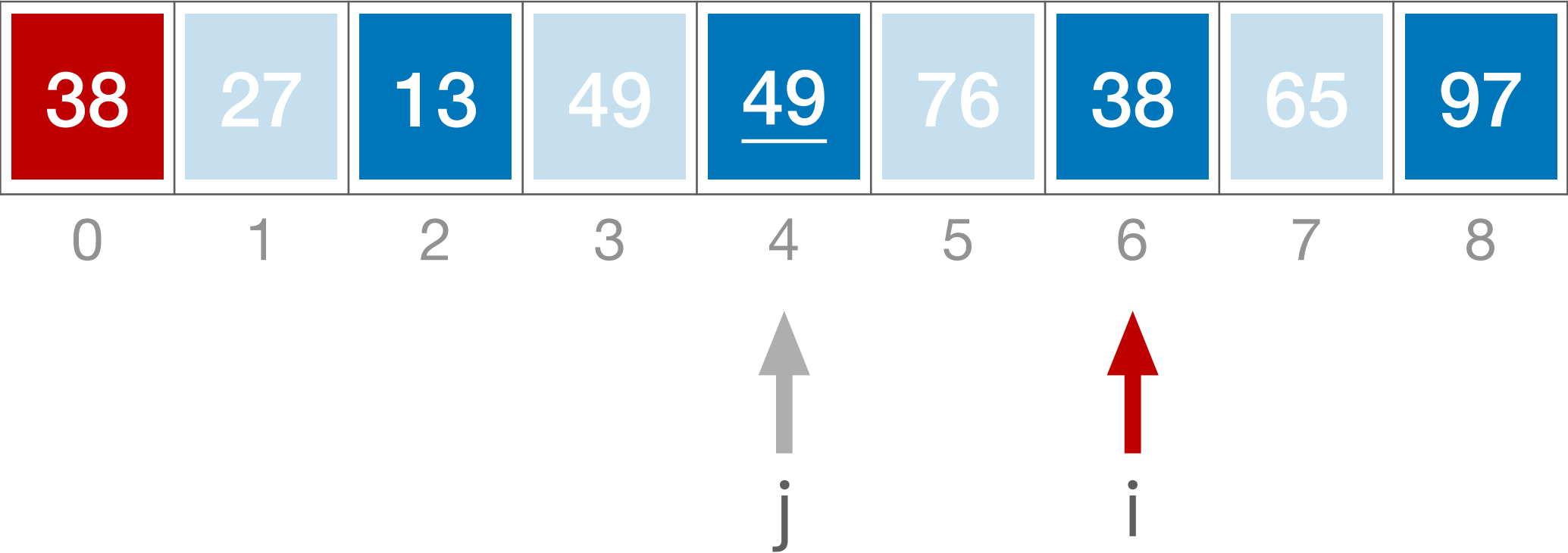




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2

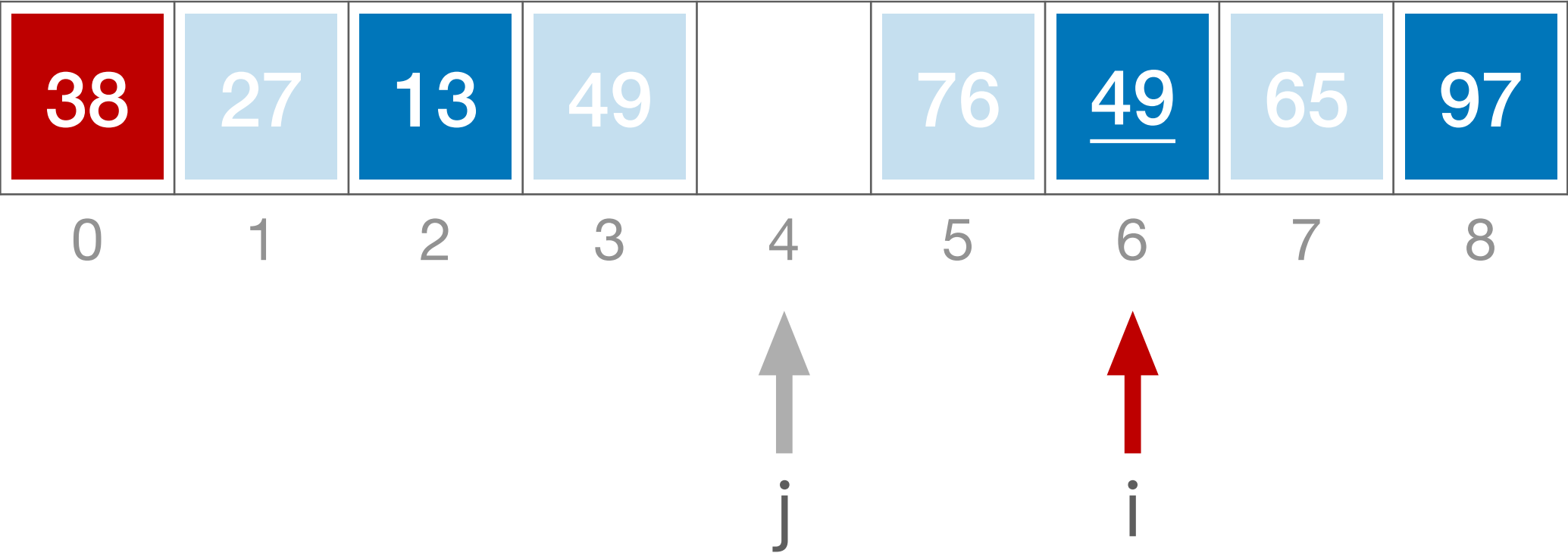




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

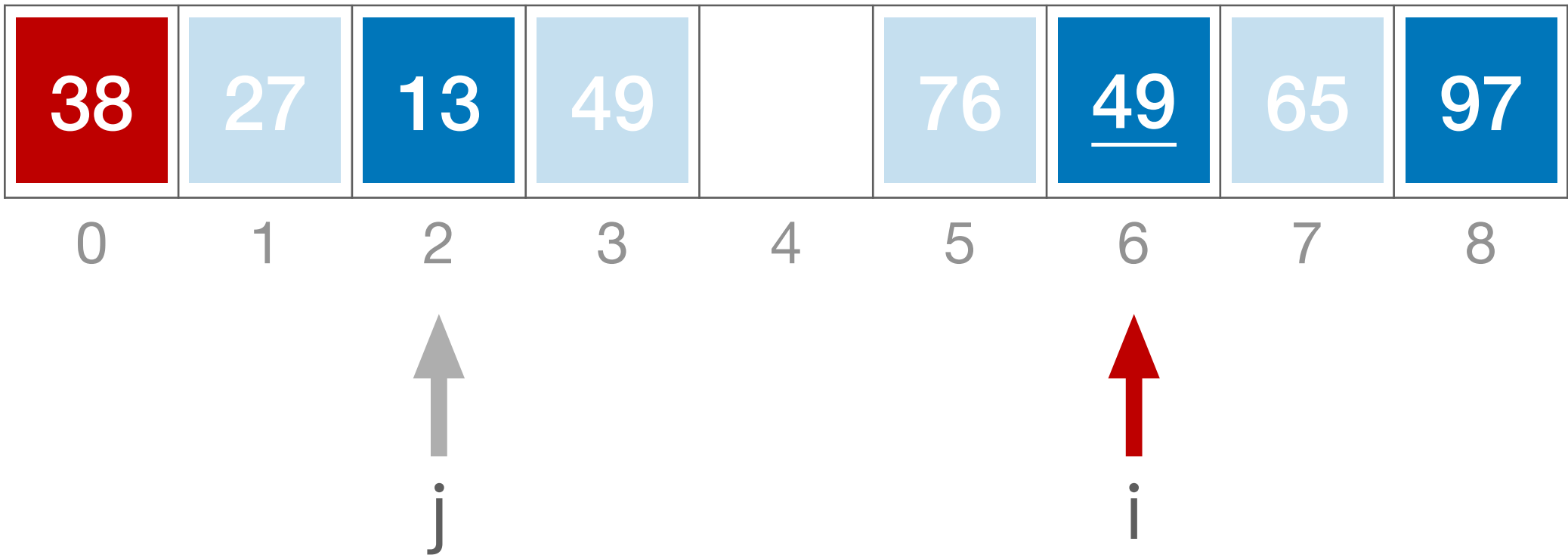
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

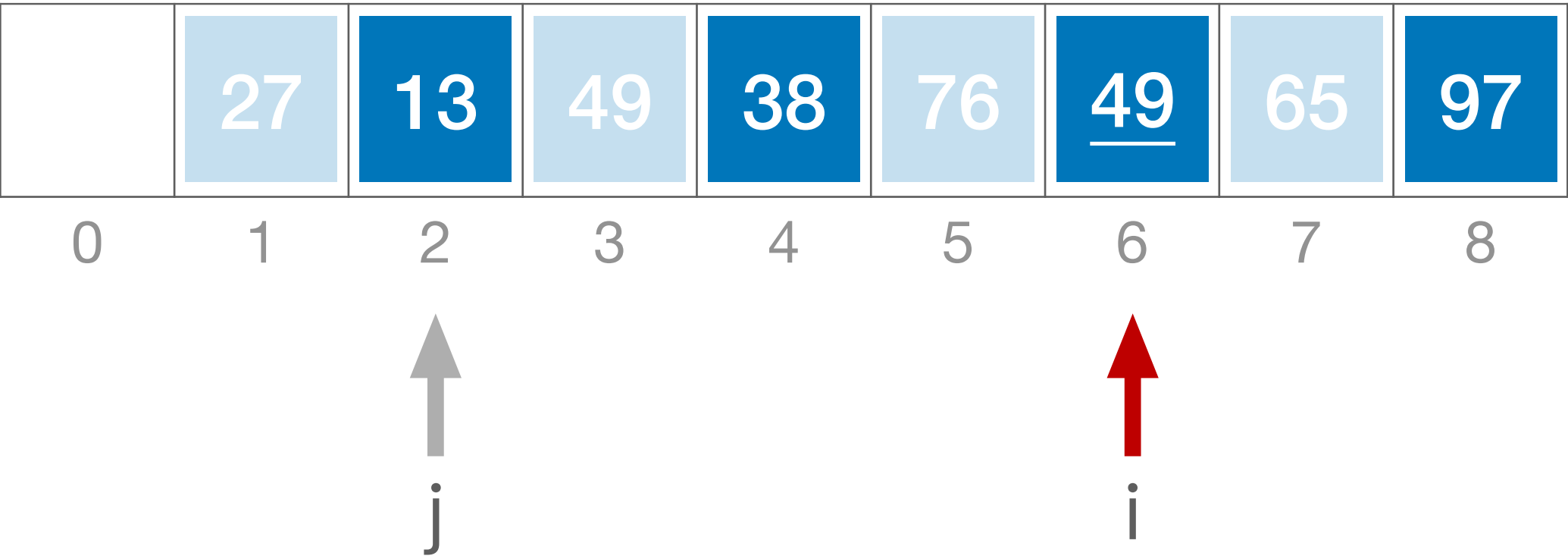
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

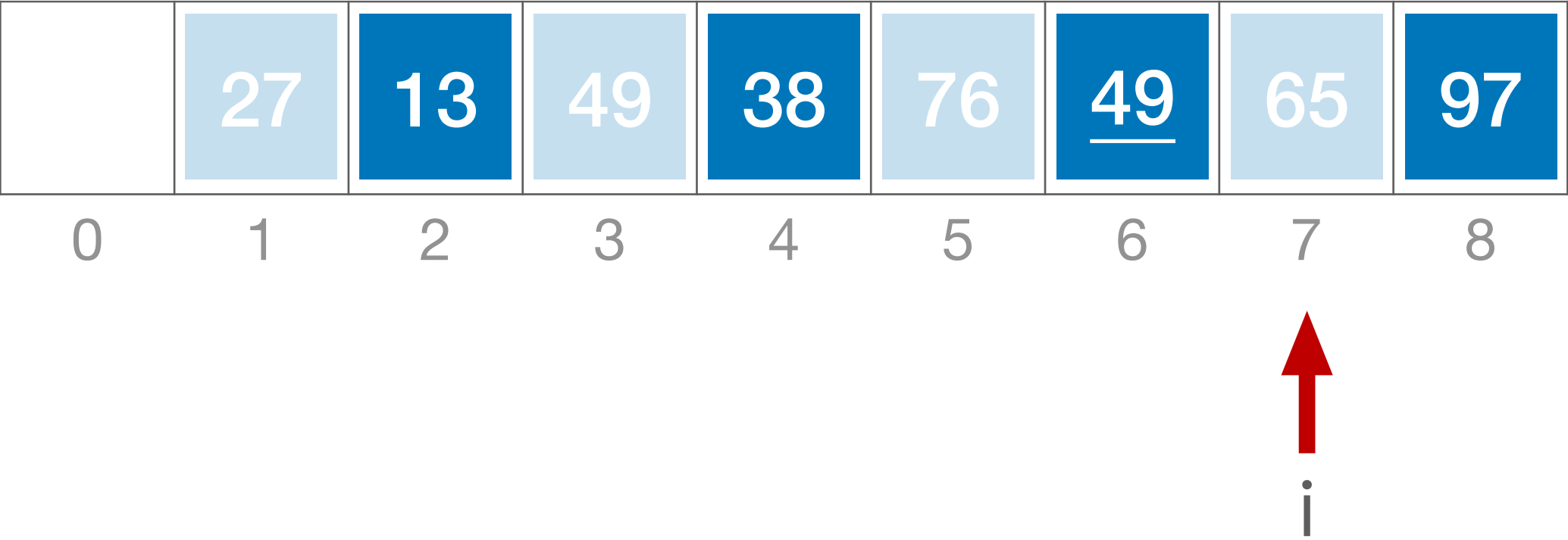
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

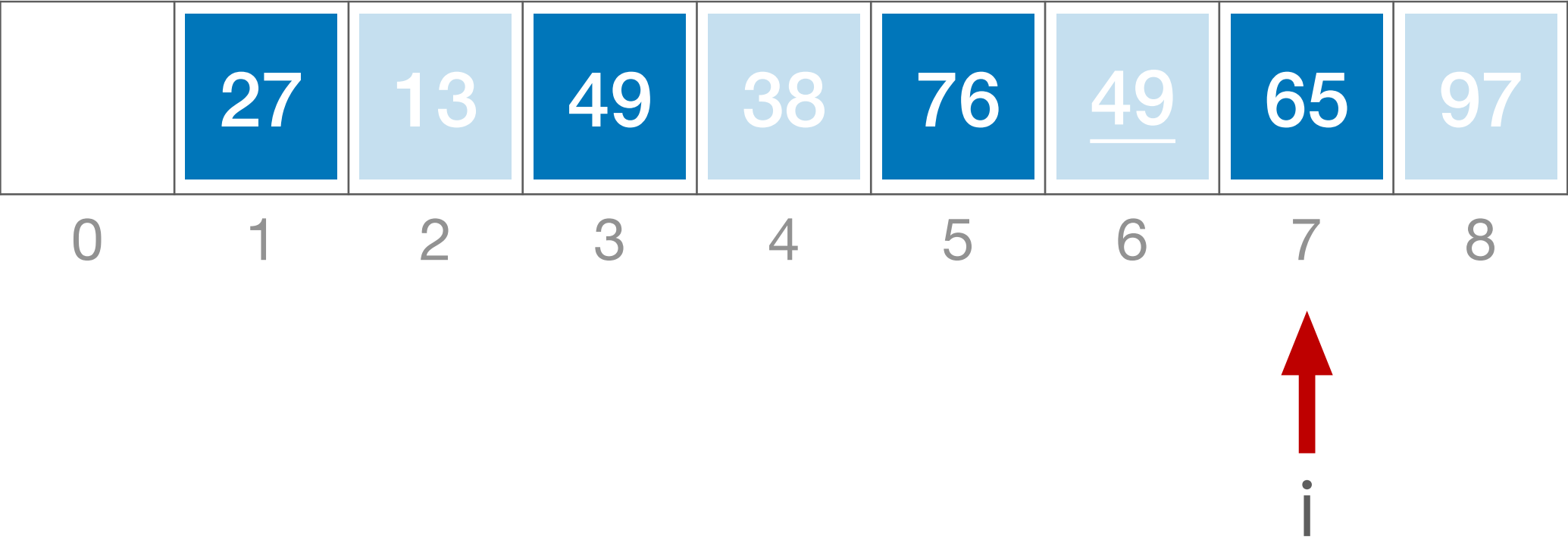
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

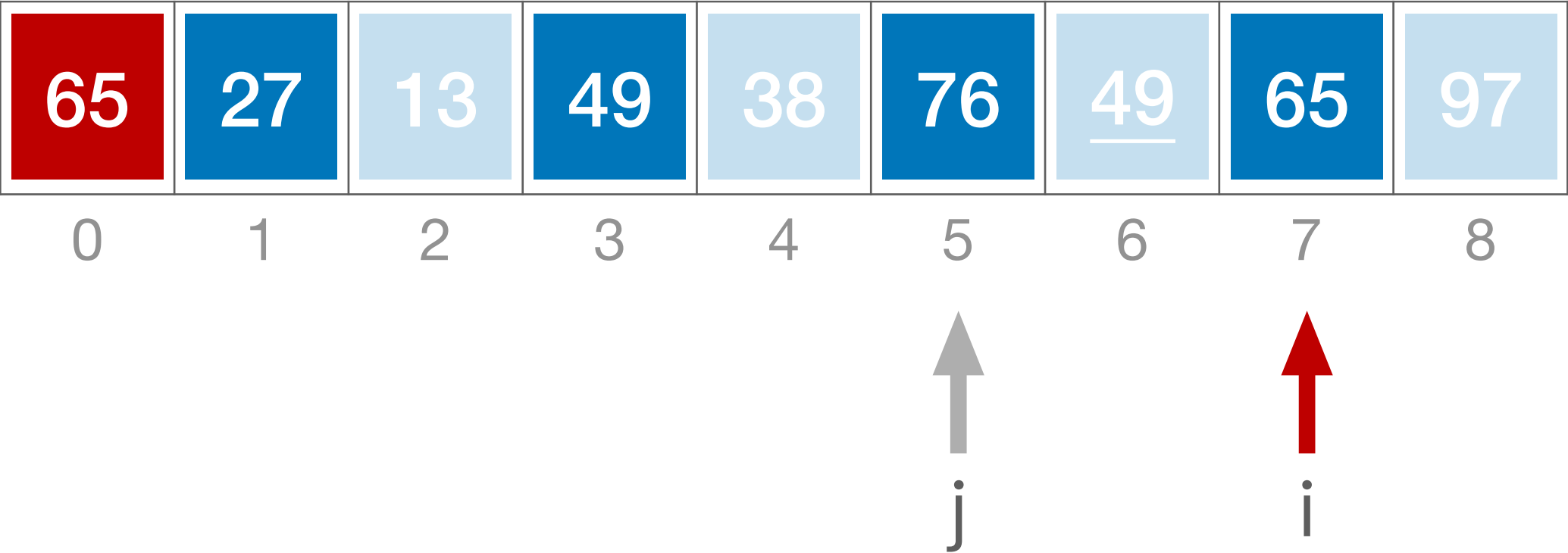
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

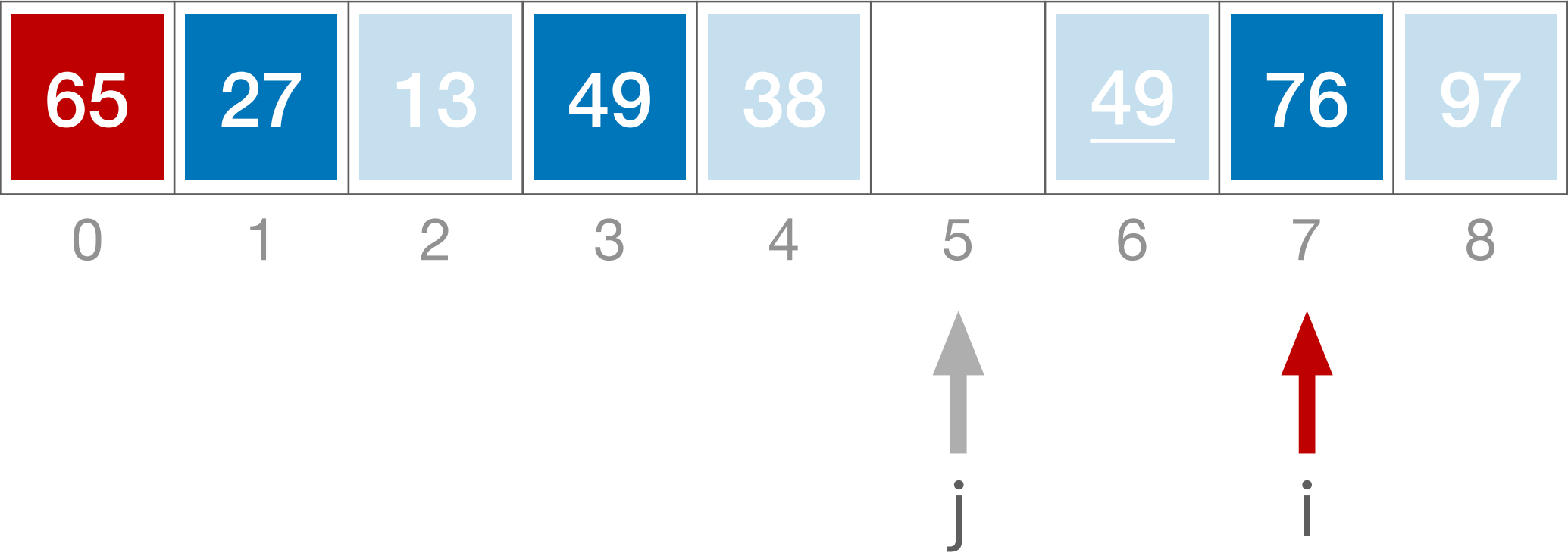
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2







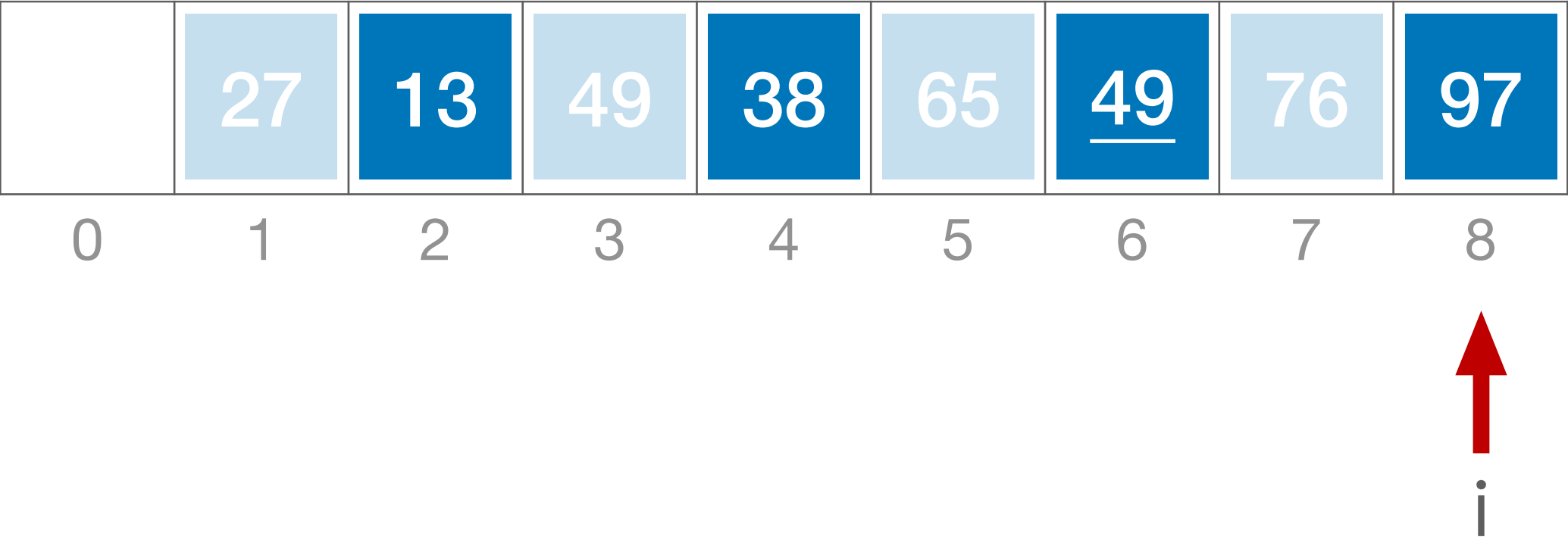




# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

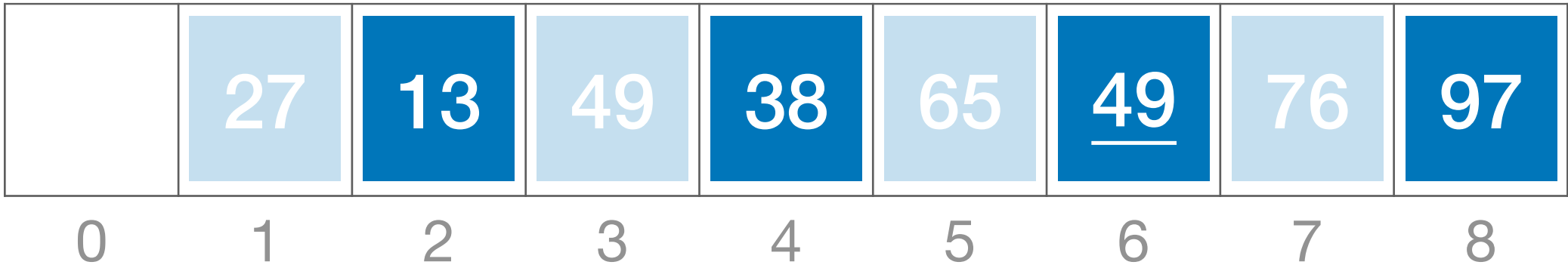
第二趟： d=2



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第二趟： d=2

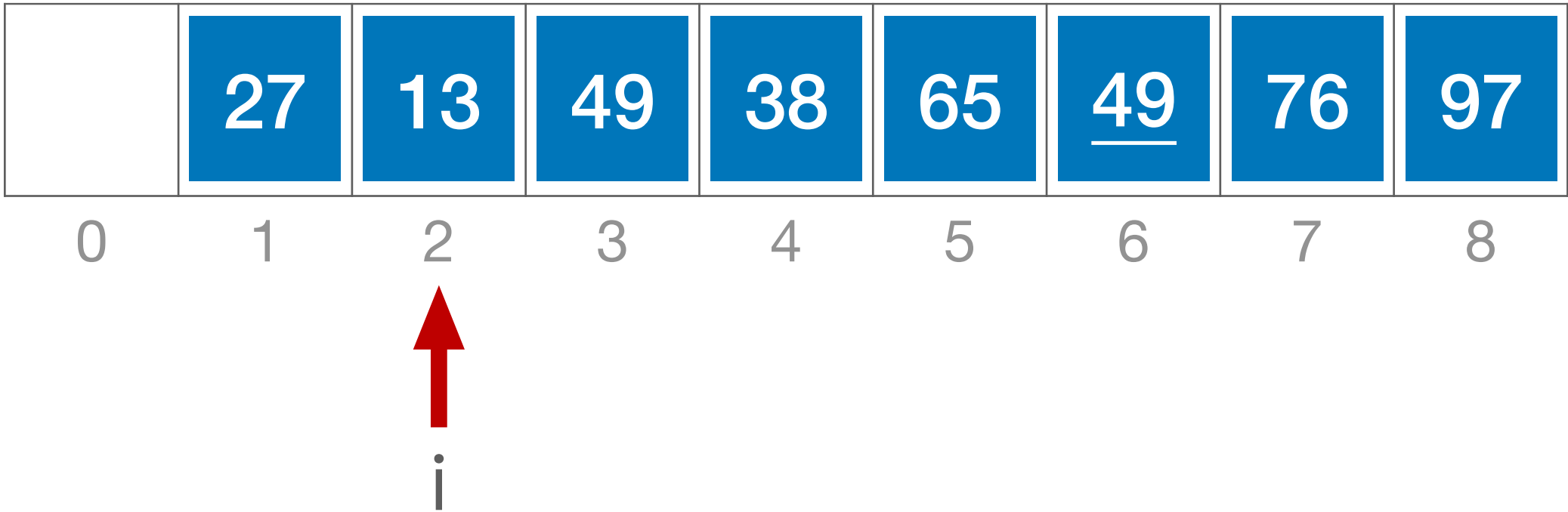


↑  
i

# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第三趟： d=1



# 算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)    //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){    //需将A[i]插入有序增量子表
                A[0]=A[i];    //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];    //记录后移，查找插入的位置
                A[j+d]=A[0];    //插入
            }//if
}
```

第三趟： d=1

|   |    |    |    |    |           |    |    |    |
|---|----|----|----|----|-----------|----|----|----|
|   | 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
| 0 | 1  | 2  | 3  | 4  | 5         | 6  | 7  | 8  |

# 算法性能分析



空间复杂度:  $O(1)$

|   |    |    |    |    |    |    |    |           |
|---|----|----|----|----|----|----|----|-----------|
|   | 49 | 38 | 65 | 97 | 76 | 13 | 27 | <u>49</u> |
| 0 | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8         |

第一趟:  $d_1 = n/2 = 4$

|    |    |    |           |    |    |    |    |
|----|----|----|-----------|----|----|----|----|
| 49 | 13 | 27 | <u>49</u> | 76 | 38 | 65 | 97 |
|----|----|----|-----------|----|----|----|----|

第一趟:  $d_1 = 3$

|    |    |    |    |           |    |    |    |
|----|----|----|----|-----------|----|----|----|
| 27 | 38 | 13 | 49 | <u>49</u> | 65 | 97 | 76 |
|----|----|----|----|-----------|----|----|----|

第二趟:  $d_2 = d_1/2 = 2$

|    |    |    |    |    |           |    |    |
|----|----|----|----|----|-----------|----|----|
| 27 | 13 | 49 | 38 | 65 | <u>49</u> | 76 | 97 |
|----|----|----|----|----|-----------|----|----|

第二趟:  $d_2 = 1$

|    |    |    |    |           |    |    |    |
|----|----|----|----|-----------|----|----|----|
| 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
|----|----|----|----|-----------|----|----|----|

第三趟:  $d_3 = d_2/2 = 1$

|    |    |    |    |           |    |    |    |
|----|----|----|----|-----------|----|----|----|
| 13 | 27 | 38 | 49 | <u>49</u> | 65 | 76 | 97 |
|----|----|----|----|-----------|----|----|----|



不知道呀

时间复杂度: 和增量序列  $d_1, d_2, d_3 \dots$  的选择有关, 目前无法用数学手段证明确切的时间复杂度  
最坏时间复杂度为  $O(n^2)$ , 当  $n$  在某个范围内时, 可达  $O(n^{1.3})$

# 算法性能分析

原始序列:

65 49 49

第一趟:  $d=2$

49 49 65

第二趟:  $d=1$

49 49 65

稳定性: 不稳定!

适用性: 仅适用于顺序表, 不适用于链表



# 知识回顾与重要考点

先将待排序表分割成若干形如  $L[i, i + d, i + 2d, \dots, i + kd]$  的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量  $d$ ，重复上述过程，直到  $d=1$  为止。

## 希尔排序

性能

空间复杂度

⊖

$O(1)$

时间复杂度

⊖

未知，但优于直接插入排序

稳定性

⊖

不稳定

适用性

⊖

仅可用于顺序表

高频题型：给出增量序列，分析每一趟排序后的状态

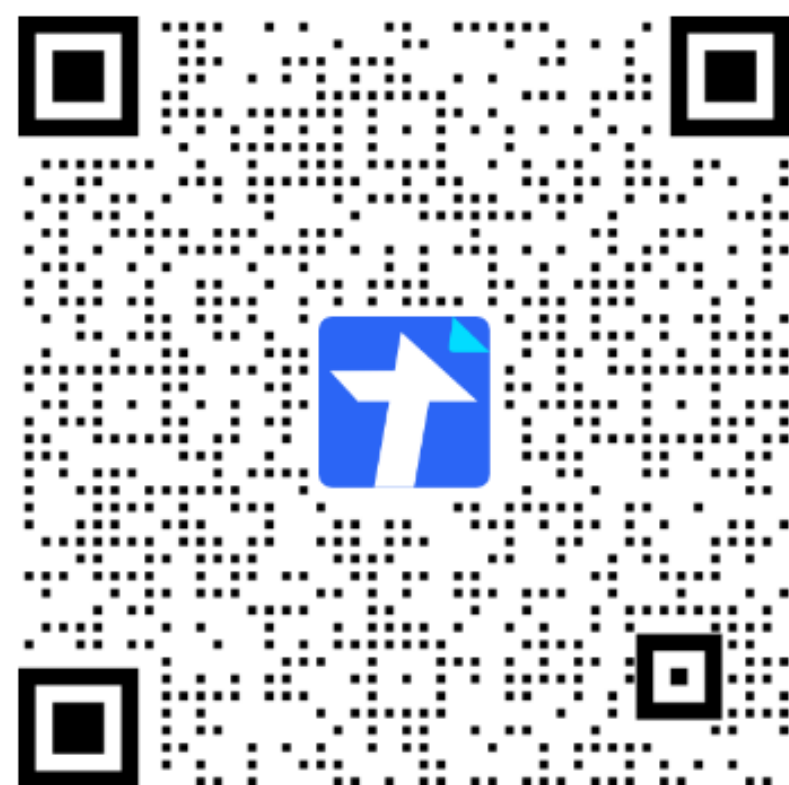


# 欢迎大家对本节视频进行评价~



**学员评分：8.2.3 希尔排序**

扫一扫二维码打开或分享给好友



— 腾讯文档 —

可多人实时在线编辑，权限安全可控



公众号：王道在线



b站：王道计算机教育



抖音：王道计算机考研