上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

Chapter 6

INTERNAL SORTING

Sorting

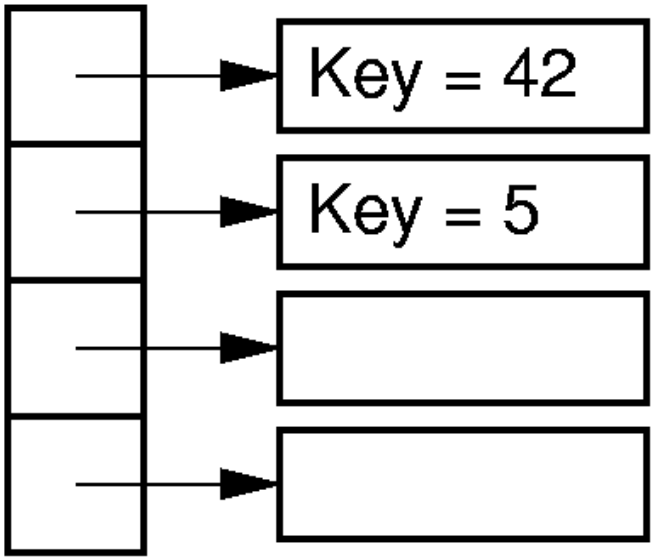
Each record contains a field called the key.

- Linear order: comparison.

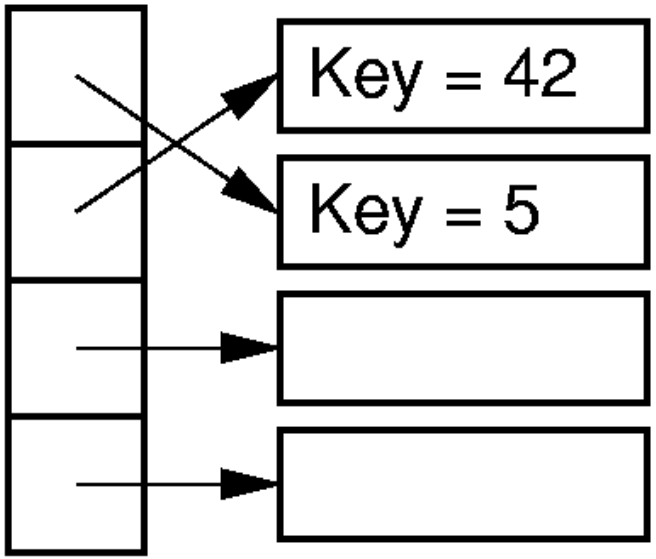
Measures of cost:

- Comparisons
- Swaps

Pointer Swapping



(a)

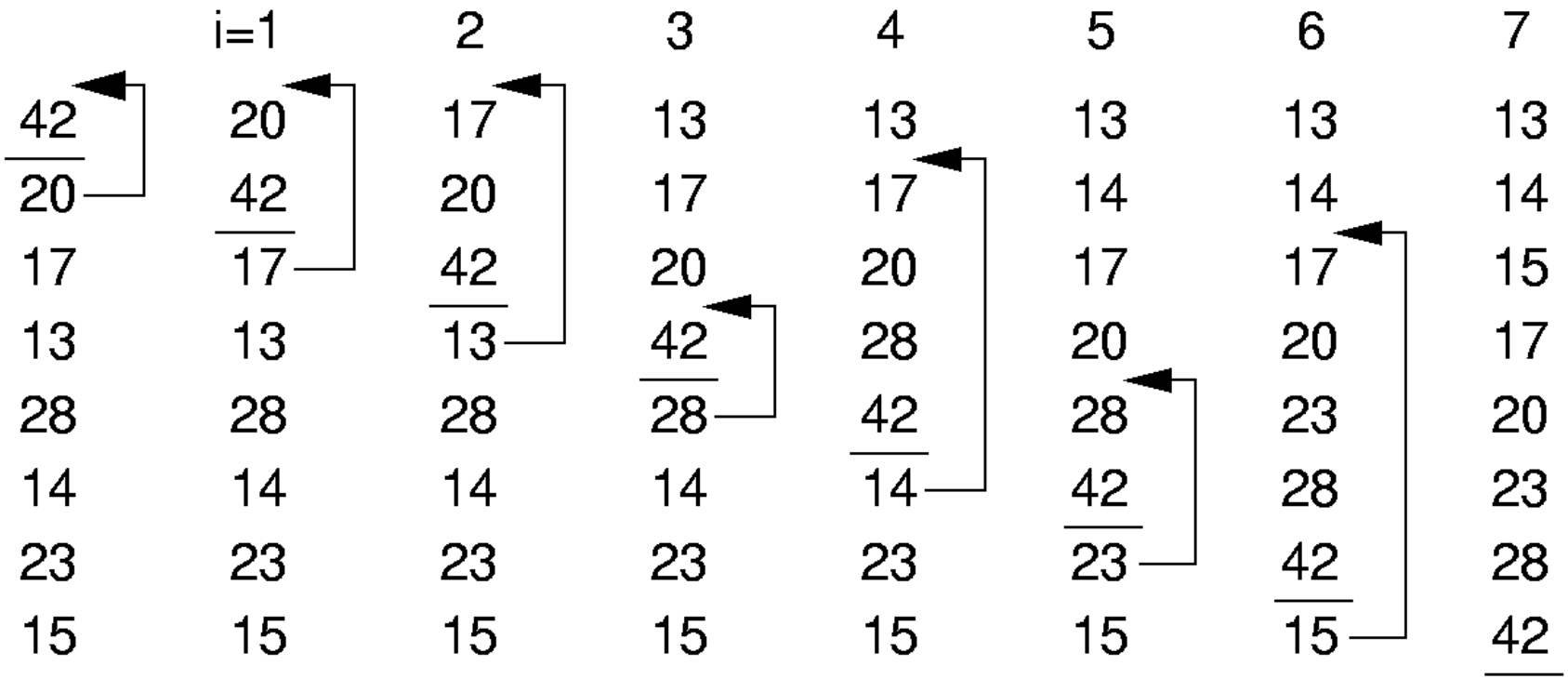


(b)

swap function

```
void swap(int *A, int i, int j)
{
   int temp;
   temp=A[i];
   A[i]=A[j];
   A[j]=temp;
}
```

Insertion Sort (1)



Insertion Sort (2)

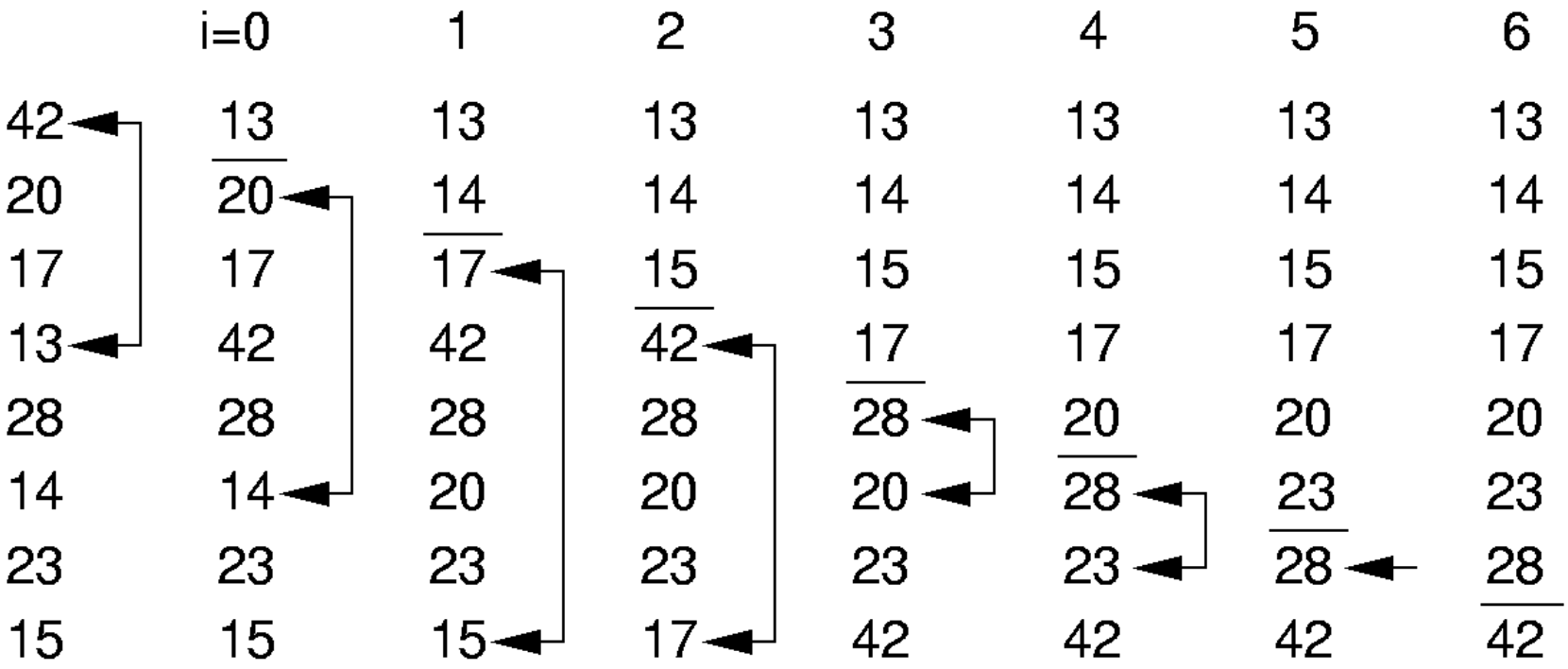
```
void inssort(int* A, int N)
{
    int i,j,temp;
   for (  i = 1; i < N; i++)
   {
        j = i - 1;
        temp = A[i];
        while (j >= 0 && A[j] > temp)
        {
               swap(A, j, j+1);
               j--;
        }

   }
}
```

Best Case: O(n)

Worst Case: $O(n^2)$

Average Case: $O(n^2)$

Selection Sort (1)



Selection Sort (2)

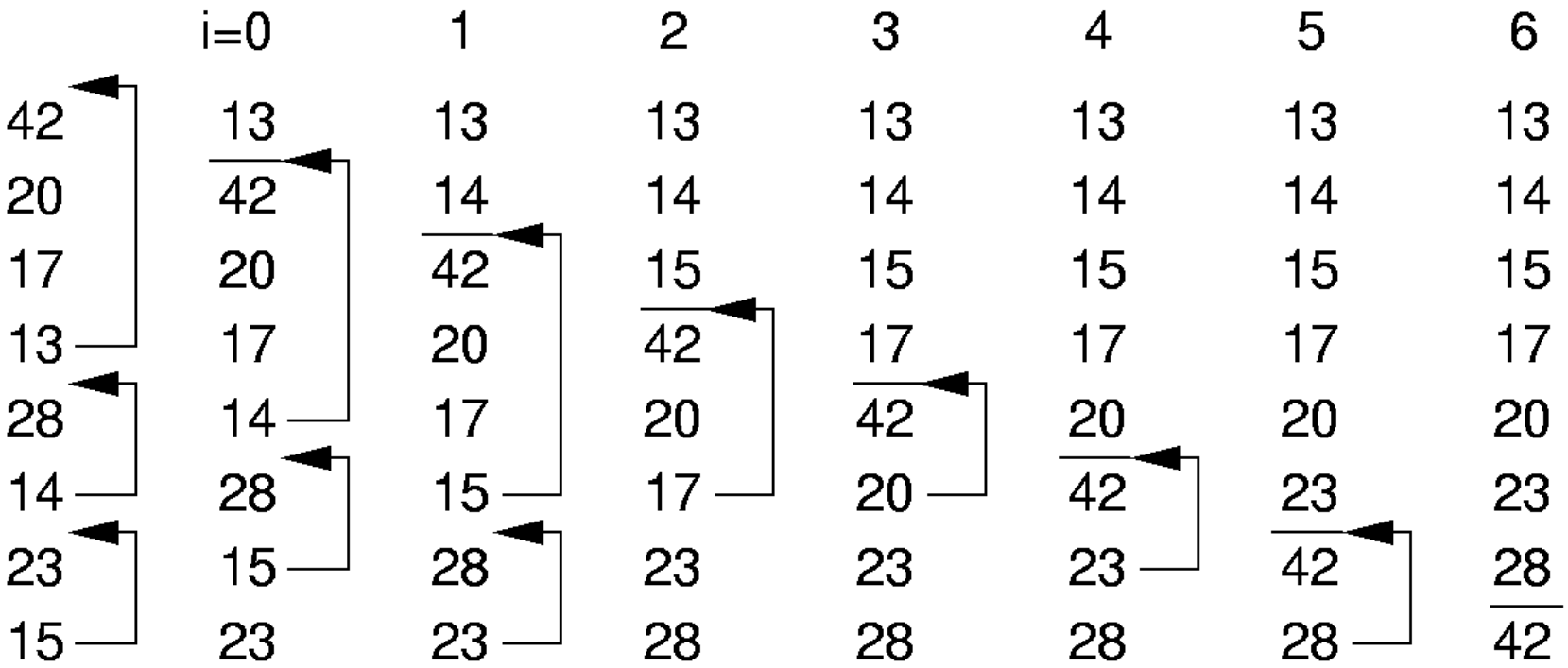
```
void selsort(int* A, int N)
{
     int i, j, index;
   for ( i = 0; i < N - 1; i++)
   {
        index = i;
        for ( j = i + 1; j < N; j++)
                if (A[j] < A[index])
                        index = j;
        if (index != i)
                swap(A, index, i);
   }
}
```

Best Case: $O(n^2)$

Worst Case: $O(n^2)$

Average Case: $O(n^2)$

Bubble Sort (1)



Bubble Sort (2)

```
void bubsort(int* A, int N)
{
    int i,j;
  for ( i = 0; i < N-1; i++)
     for ( j = 0; j < N-i-1; j++)
         if (A[j] > A[j+1] )
             swap(A, j, j+1);
}
```

Best Case: $O(n^2)$

Worst Case: $O(n^2)$

Average Case: $O(n^2)$

Summary

	Insertion	Bubble	Selection
Comparisons:			
Best Case	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$
Average Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Worst Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Swaps			
Best Case	0	0	$\Theta(n)$
Average Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Worst Case	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$

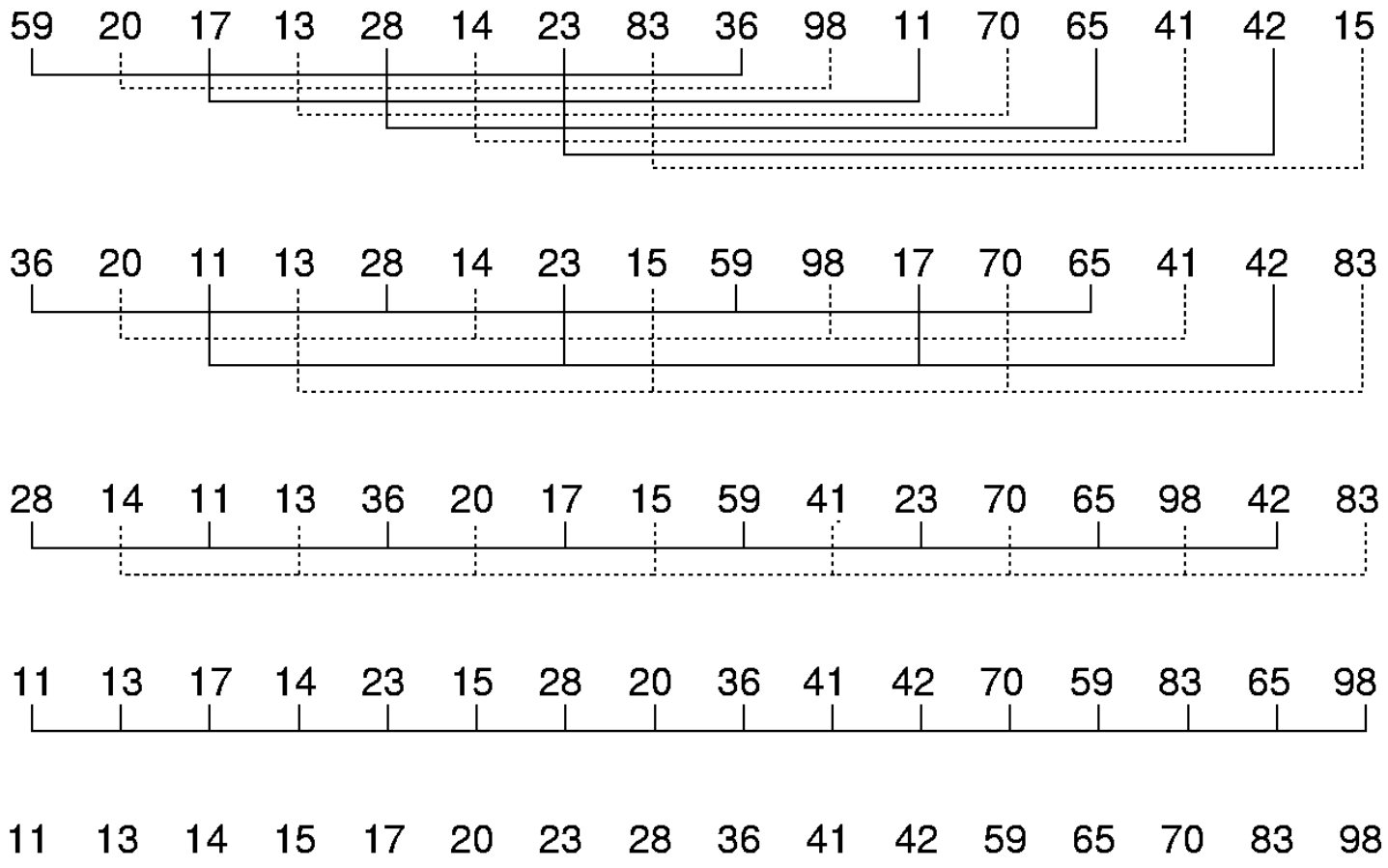
Exchange Sorting

All of the sorts so far rely on exchanges of adjacent records.

What is the average number of exchanges required?

- There are $n!$ permutations
- Consider permuation X and its reverse, X'
- Together, every pair requires $n(n-1)/2$ exchanges.

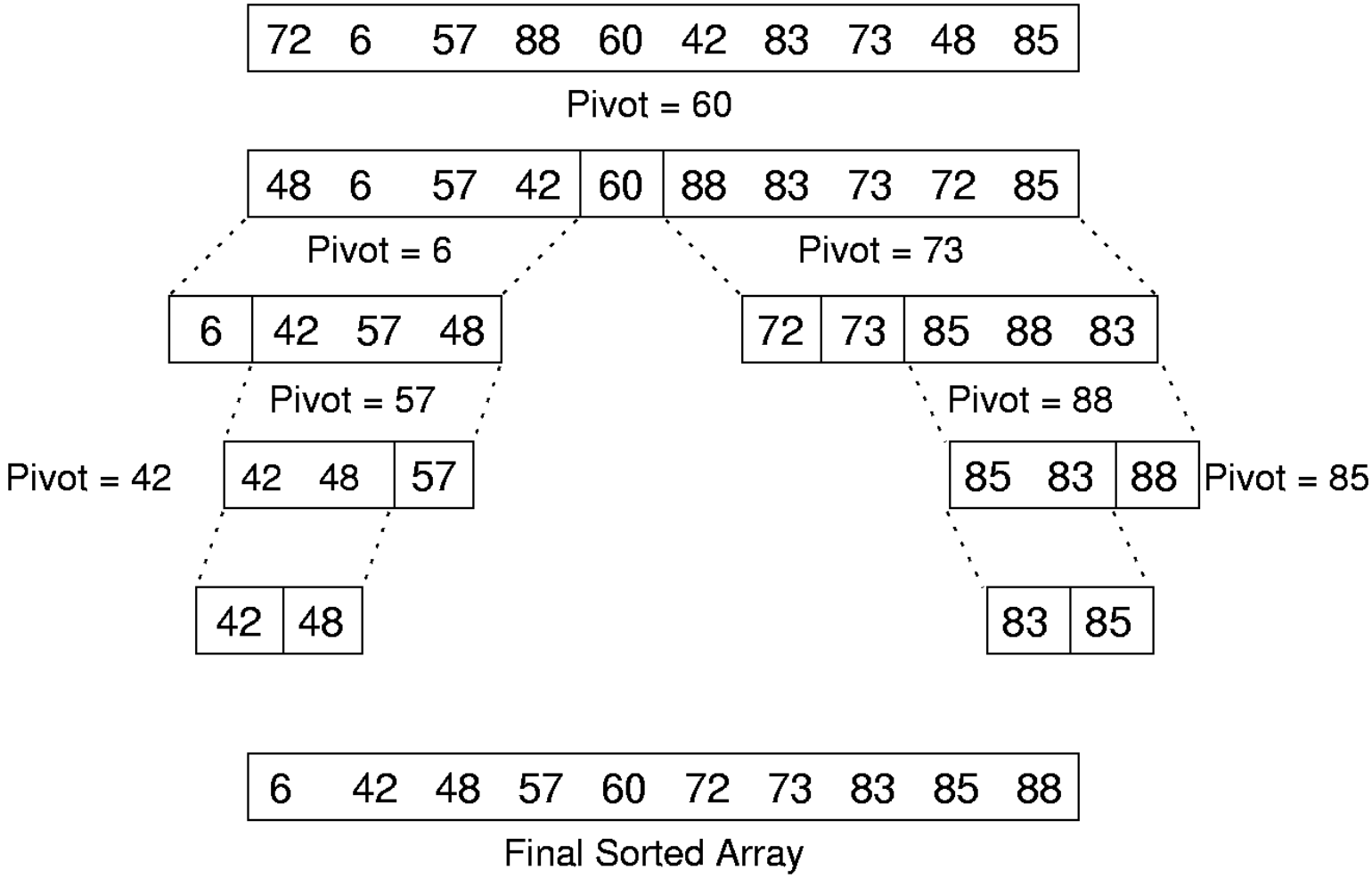
Shellsort


59 20 17 13 28 14 23 83 36 98 11 70 65 41 42 15
36 20 11 13 28 14 23 15 59 98 17 70 65 41 42 83
28 14 11 13 36 20 17 15 59 41 23 70 65 98 42 83
11 13 17 14 23 15 28 20 36 41 42 70 59 83 65 98
11 13 14 15 17 20 23 28 36 41 42 59 65 70 83 98

Shellsort

```
void shesort(int* A, int N)
{
 int i, j, Increment,Tmp;
 for( Increment = N / 2; Increment > 0; Increment /= 2 )
    for( i = Increment; i < N; i++ )
    {
            Tmp = A[ i ];
        for( j = i; j >= Increment; j -= Increment )
            if( Tmp < A[ j - Increment ] )
                A[ j ] = A[ j - Increment ];
                    else
                    break;
            A[ j ] = Tmp;
    }
}
```

Quicksort Example



Quicksort

```
void quisort(int* A, int start, int end)
{
        int i;
        if( end > start )
          {
           i = partition( A, start, end);
           quisort( A, start, i - 1 );
           quisort( A, i + 1, end );
          }
}
```

Quicksort Partition （different elements）

```
int partition(int* A, int left, int right)
{
    int i,j;
    int pivo = A[ (left+right)/2 ];
    i = left;  j = right;

    while( 1 )
    {
      while( A[ i ] < pivo ){ i++; }
      while( A[ j ] > pivo ){ j--; }
      if ( i < j )
        swap(A, i, j);
      else
            break;
    }
    return i;

}
```

The cost for partition is $\Theta(n)$.

Partition Example

Initial	l	72	6	57	88	85	42	83	73	48	60 r
Pass 1		72 l	6	57	88	85	42	83	73	48 r	60
Swap 1		48 l	6	57	88	85	42	83	73	72 r	60
Pass 2		48	6	57	88 l	85	42 r	83	73	72	60
Swap 2		48	6	57	42 l	85	88 r	83	73	72	60
Pass 3		48	6	57	42 r	85 l	88	83	73	72	60

Cost of Quicksort

Best case: Always partition in half.

Worst case: Bad partition.

Average case:

$$T(n) = cn + 1/n \sum_{k=0}^{n-1} (T(k) + T(n-k))$$

Optimizations for Quicksort:

- Better Pivot
- Better algorithm for small sublists
- Eliminate recursion