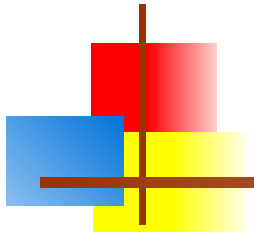
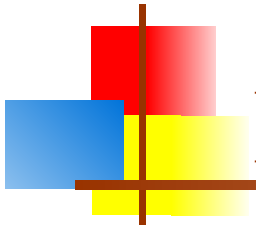


Data Structures (Sorting)
CSE-207



Bubble sort

- Compare each element (except the last one) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the largest element at the very end
  - The last element is now in the correct and final place
- Compare each element (except the last *two*) with its neighbor to the right
  - If they are out of order, swap them
  - This puts the second largest element next to last
  - The last two elements are now in their correct and final places
- Compare each element (except the last *three*) with its neighbor to the right
  - Continue as above until you have no unsorted elements on the left

Sorting: Bubble sort

- Bubble sort compares the numbers in pairs from left to right exchanging when necessary.
- Suppose we have array A[1],A[2],……A[N]
- Pass 1
  - Compare A[1] and A[2] arrange them as A[1]<A[2]
  - Then A[2] and A[3] So that A[2]<A[3]
  - … Continue so that A[N-1]<A[N].
- Pass 2
  - Repeat step 1 with one less comparison. So A[N-1] in its right place
- Pass N-1
  - Compare A[1] and A[2] so that ….A[1]<A[2].

Bubble Sort Example

Pass-1

9, 6, 2, 12, 11, 9, 3

6, 9, 2, 12, 11, 9, 3

6, 2, 9, 12, 11, 9, 3

6, 2, 9, 12, 11, 9, 3

6, 2, 9, 11, 12, 9, 3

6, 2, 9, 11, 9, 12, 3

6, 2, 9, 11, 9, 3, 12

Bubble Sort Example

Pass-2

6, 2, 9, 11, 9, 3, 12

2, 6, 9, 11, 9, 3, 12

2, 6, 9, 11, 9, 3, 12

2, 6, 9, 11, 9, 3, 12

2, 6, 9, 9, 11, 3, 12

2, 6, 9, 9, 3, 11, 12

2, 6, 9, 9, 3, 11, 12

Bubble Sort Example

Pass-3

2, 6, 9, 9, 3, 11, 12

2, 6, 9, 9, 3, 11, 12

2, 6, 9, 9, 3, 11, 12

2, 6, 9, 9, 3, 11, 12

2, 6, 9, 3, 9, 11, 12

2, 6, 9, 3, 9, 11, 12

Bubble Sort Example

Pass-4

2, 6, 9, 3, 9, 11, 12

2, 6, 9, 3, 9, 11, 12

2, 6, 9, 3, 9, 11, 12

2, 6, 3, 9, 9, 11, 12

2, 6, 3, 9, 9, 11, 12

Bubble Sort Example

Pass-5

2, 6, 3, 9, 9, 11, 12

2, 6, 3, 9, 9, 11, 12

2, 3, 6, 9, 9, 11, 12

2, 3, 6, 9, 9, 11, 12

Bubble Sort Example

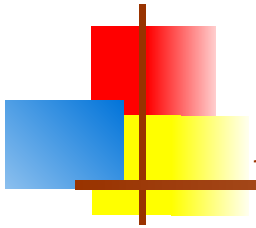
Pass-6

2, 3, 6, 9, 9, 11, 12

2, 3, 6, 9, 9, 11, 12

2, 3, 6, 9, 9, 11, 12

So we have done!!!!



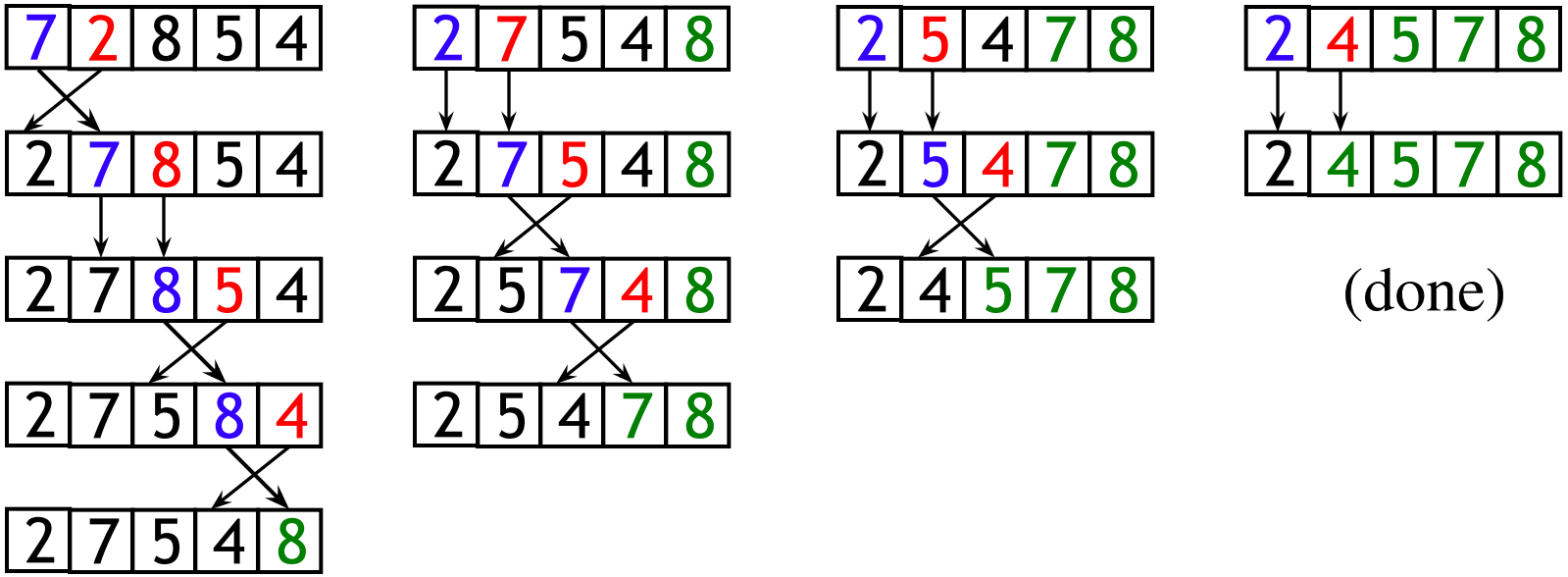
Algorithm

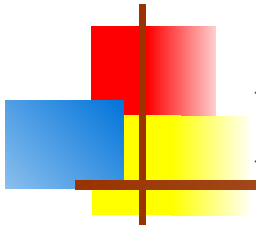
Algorithm: Bubble_Sort(List, N)

Here List is the collection of items and N is the total no. of items.

1. Repeat steps 2 and 3 for I = 1….. N
2. Repeat step 3 for J = 1…….N
3. If List[J] > List[J+1] then swap(List[J], List[J+1]).
4. End.

Example of bubble sort





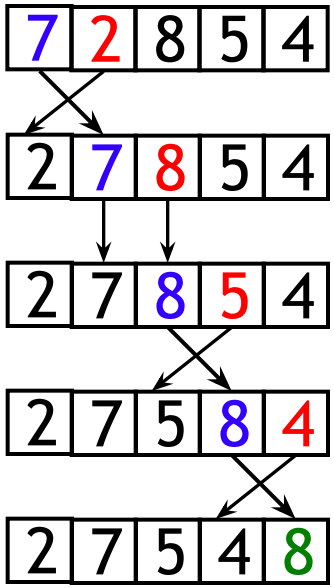
Before we go…..

- Swap two variable

```
if (a[j] > a[j+1])
    {
        temp = a[j+1];
        a[j+1] = a[j];
        a[j] = temp;
    }
```

Code for bubble sort

```
for (j = 0; j < array_size - 1; j++ )
{
       if (a[j] > a[j+1])
    {
      temp = a[j+1];
      a[j+1] = a[j];
      a[j] = temp;
    }
}
```



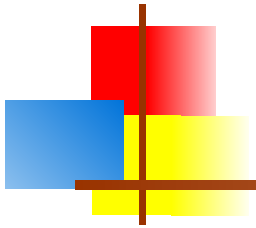
Code for bubble sort

```
for (i = 0; i < (array_size - 1); ++i)
{
    for (j = 0; j < array_size - 1; j++ )
    {
          if (a[j] > a[j+1])
        {
          temp = a[j+1];
          a[j+1] = a[j];
          a[j] = temp;
        }
    }
}
```

Code for bubble sort

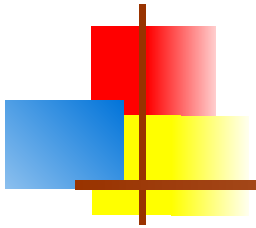
Efficient

```
for (i = 0; i < (array_size - 1); ++i)
{
    for (j = 0; j < array_size - 1-i; j++ )
    {
          if (a[j] > a[j+1])
        {
          temp = a[j+1];
          a[j+1] = a[j];
          a[j] = temp;
        }
    }
}
```



Selection sort

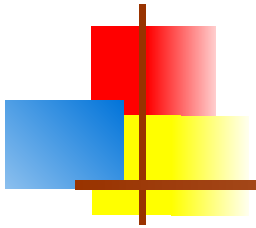
- Selection is a simple sorting algorithm.
- It works by first finding the smallest element using a linear scan and swapping it into the first position in the list.
- Then finding the second smallest element by scanning the remaining elements, and so on.



Selection sort

Algorithm: Selection_Sort (List, N)

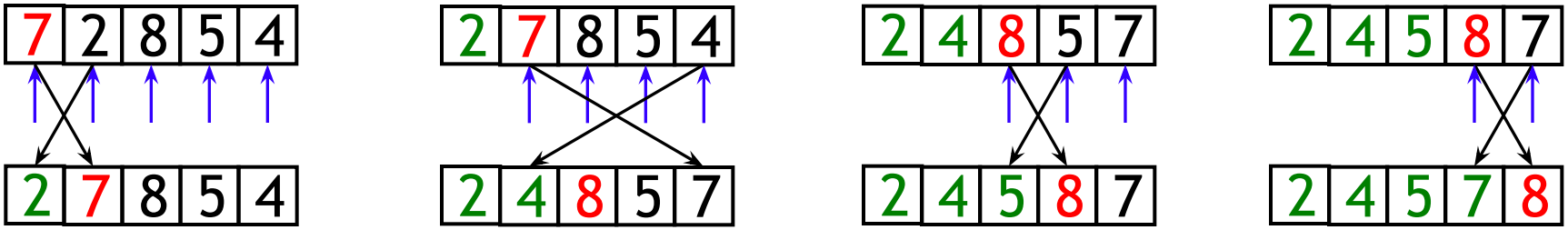
1. Repeat steps 2 to 6 for I = 1 to N
2. Set Min := I
3. Repeat steps 4 and 5 for J = I+1 to N
4. If List [J] < List [Min] then
5. Set Min := J
6. swap(List[I], List[Min])
7. End

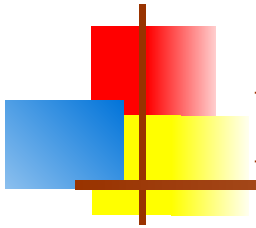


Selection sort

- Given an array of length n,
  - Search elements 0 through n-1 and select the smallest
    - Swap it with the element in location 0
  - Search elements 1 through n-1 and select the smallest
    - Swap it with the element in location 1
  - Search elements 2 through n-1 and select the smallest
    - Swap it with the element in location 2
  - Search elements 3 through n-1 and select the smallest
    - Swap it with the element in location 3
  - Continue in this fashion until there's nothing left to search

Example and analysis of selection sort

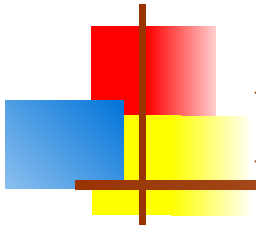




Before we go…..

- Find the min value and its position

```
min = 0;
for (j = min+1; j < array_size; j++)
{
   if (a[min]>a[j])
   min = j;
  }
 printf("%d %d", min, a[min]);
```



Before we go…..

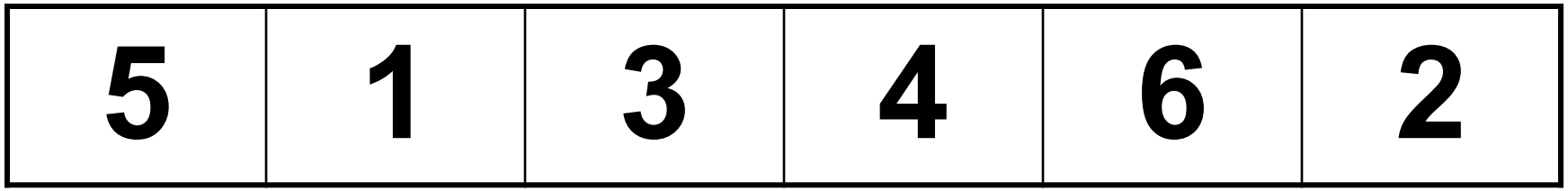
- Find the min value and its position
- swap it with first position

```
min = 0;
for (j = min+1; j < array_size; j++)
{
   if (a[min]>a[j])
   min = j;
  }
 printf("%d %d", min, a[min]);
 temp = a[0];
 a[0] = a[min];
 a[min] = temp;
```

Selection sort

```
for (i = 0; i < array_size - 1; ++i)
{
      min = i;//0
     for (j = min+1; j < array_size; j++)
   {
       if (a[min]>a[j])
       min = j;
       }
       temp = a[i]; //a[0]
          a[i] = a[min];
          a[min] = temp;
}
```

Selection Sort

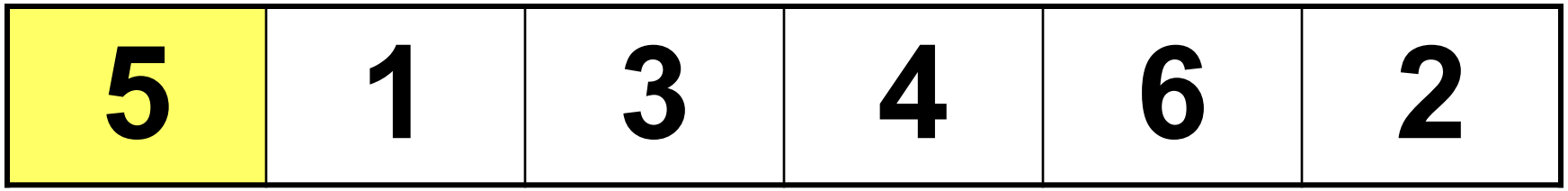


Comparison

Data Movement

Sorted

Selection Sort

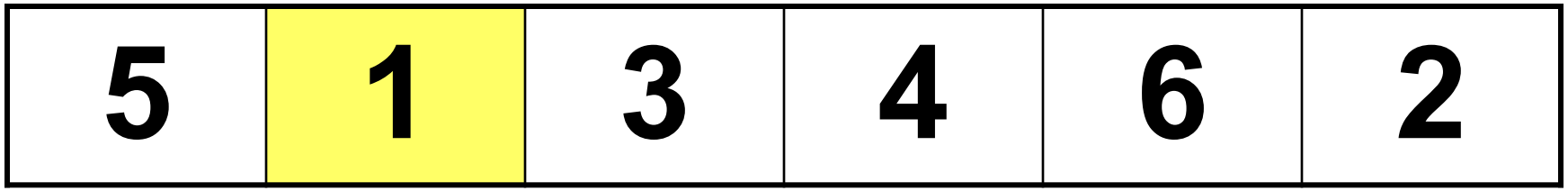


Comparison

Data Movement

Sorted

Selection Sort

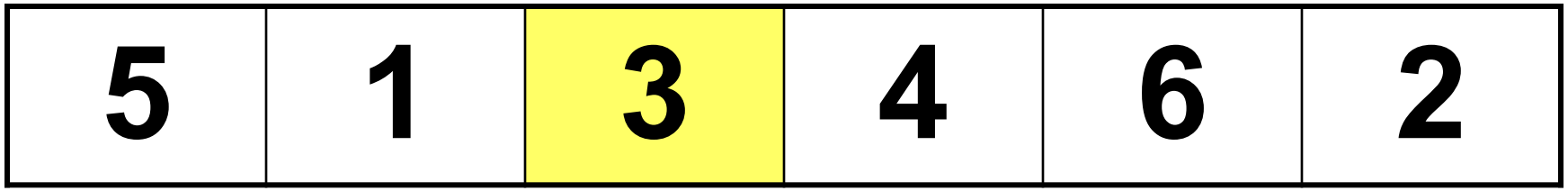


Comparison

Data Movement

Sorted

Selection Sort

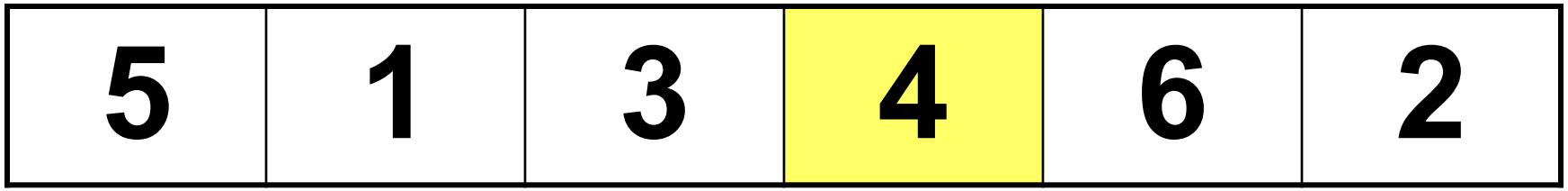


Comparison

Data Movement

Sorted

Selection Sort

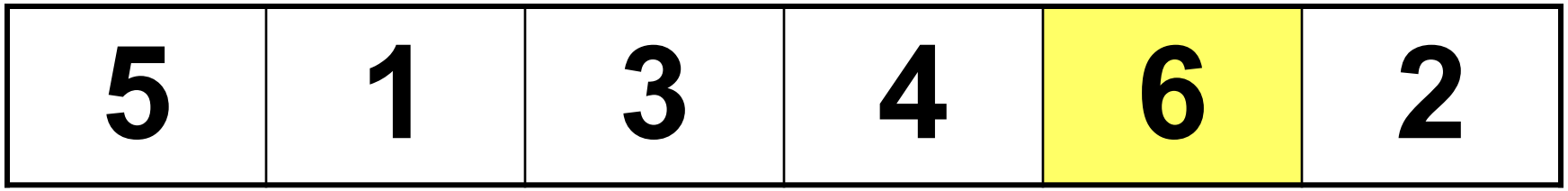


Comparison

Data Movement

Sorted

Selection Sort

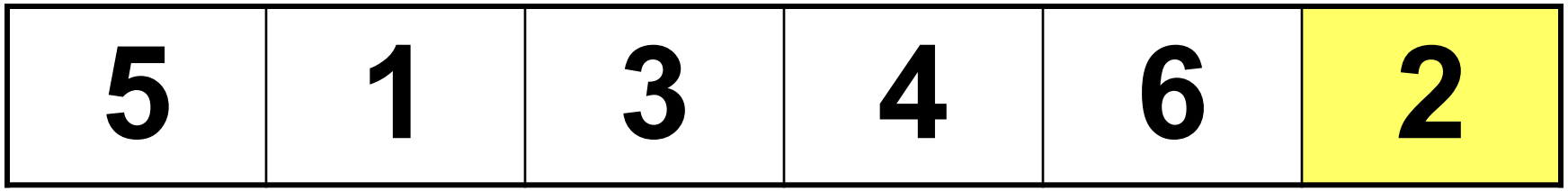


Comparison

Data Movement

Sorted

Selection Sort

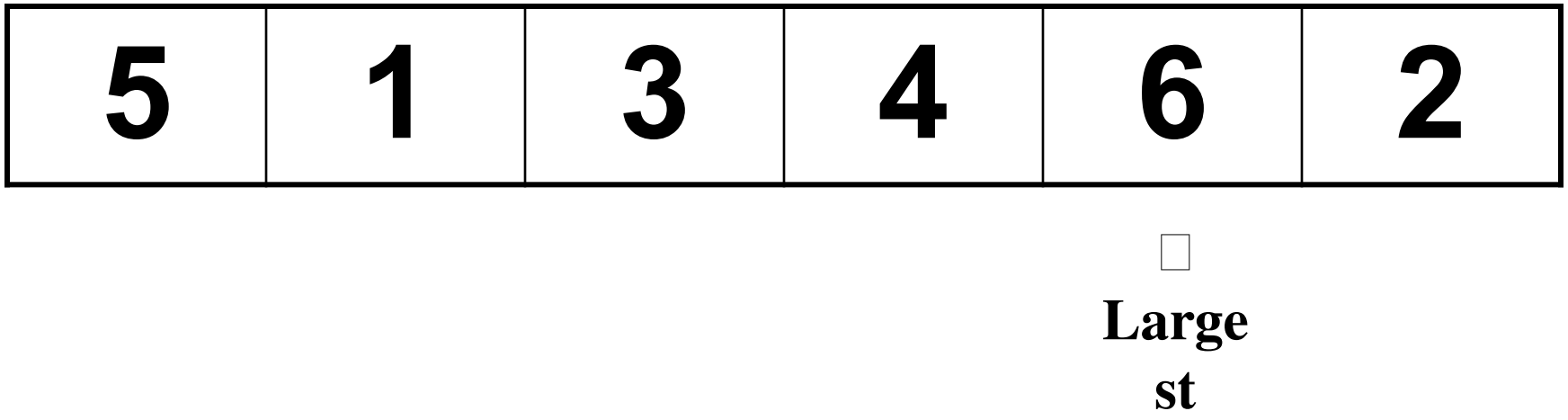


Comparison

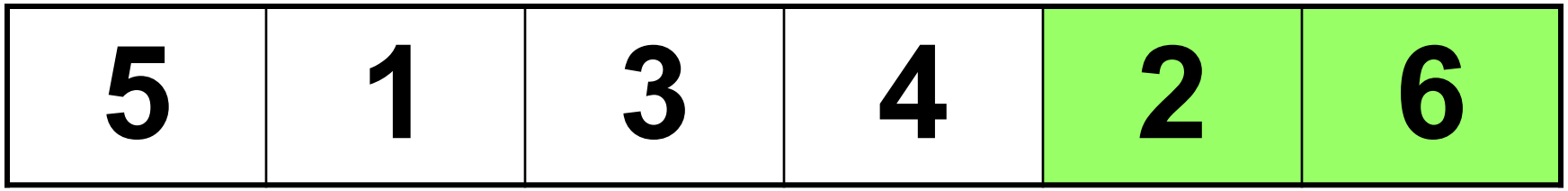
Data Movement

Sorted

Selection Sort



Selection Sort

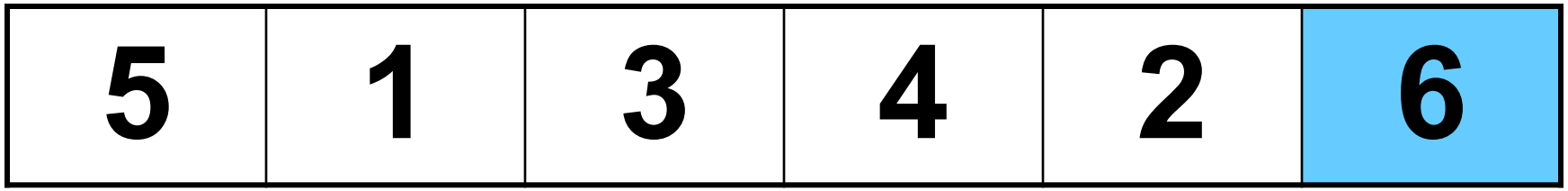


Comparison

Data Movement

Sorted

Selection Sort

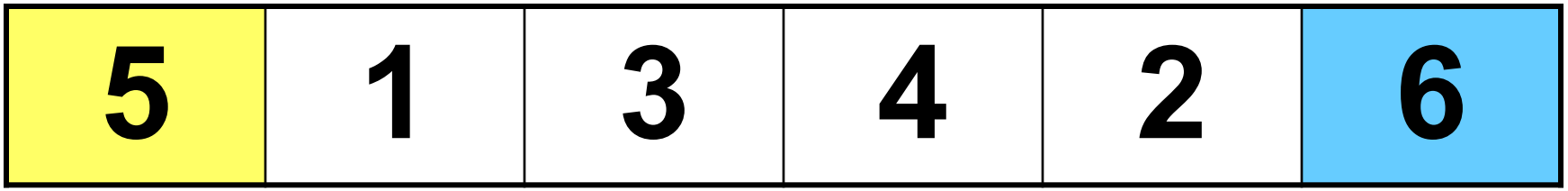


Comparison

Data Movement

Sorted

Selection Sort

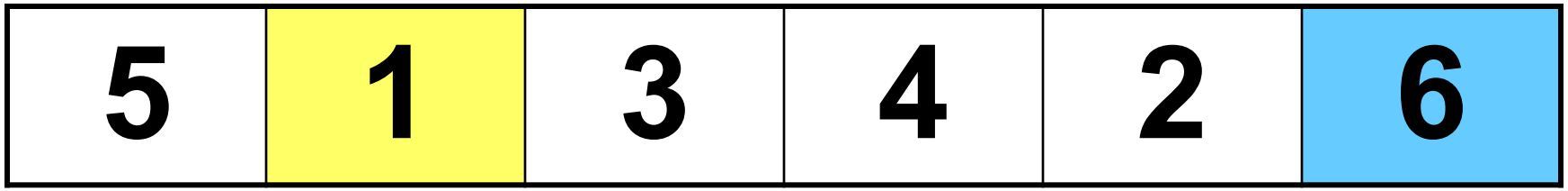


Comparison

Data Movement

Sorted

Selection Sort

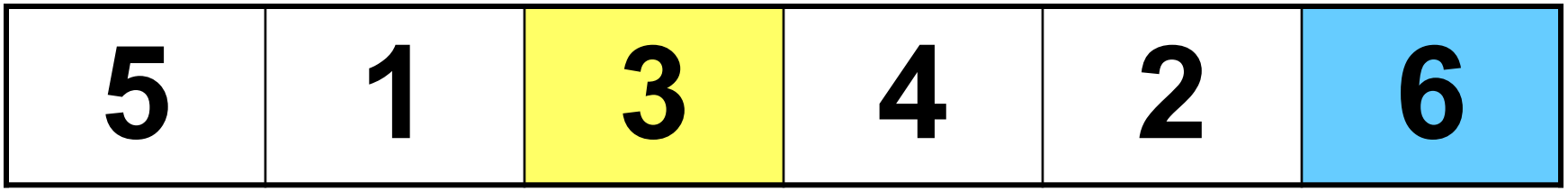


Comparison

Data Movement

Sorted

Selection Sort

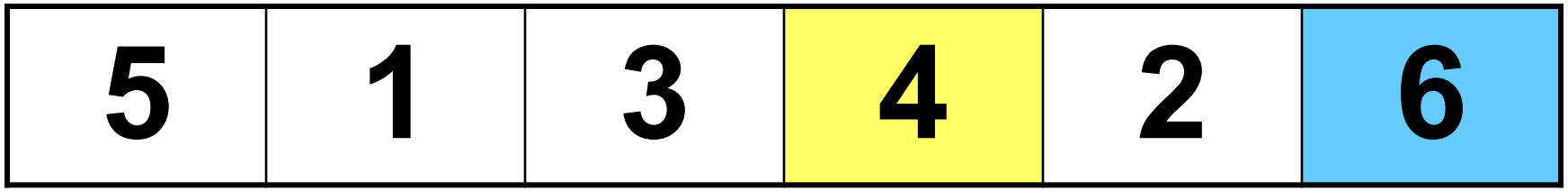


Comparison

Data Movement

Sorted

Selection Sort

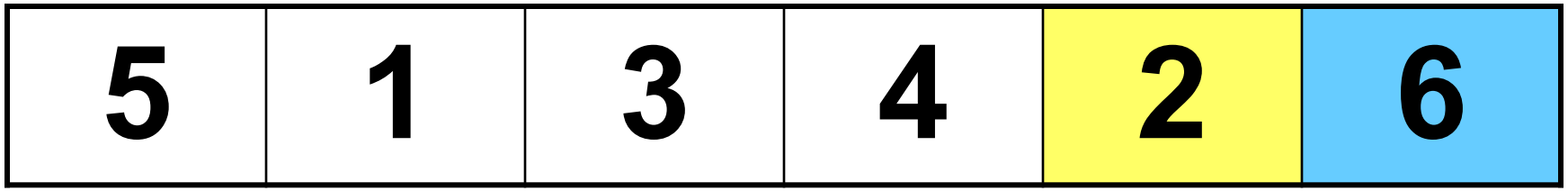


Comparison

Data Movement

Sorted

Selection Sort

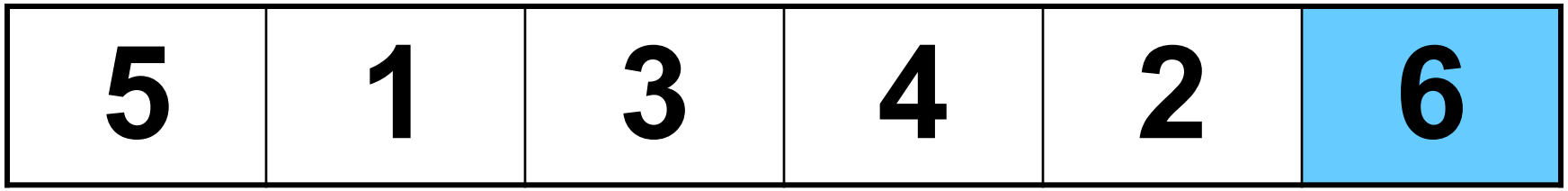


Comparison

Data Movement

Sorted

Selection Sort



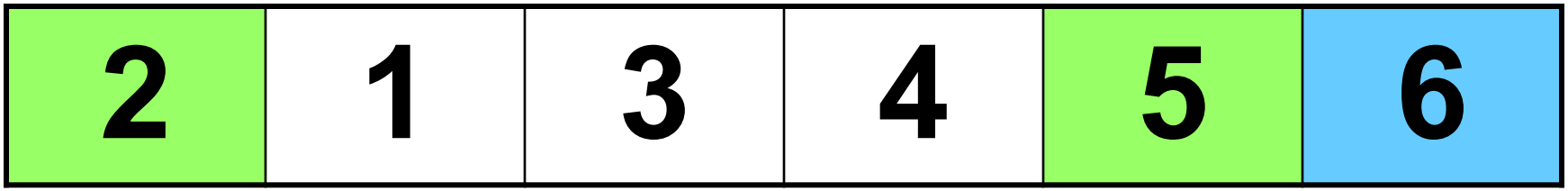
Largest

Comparison

Data Movement

Sorted

Selection Sort

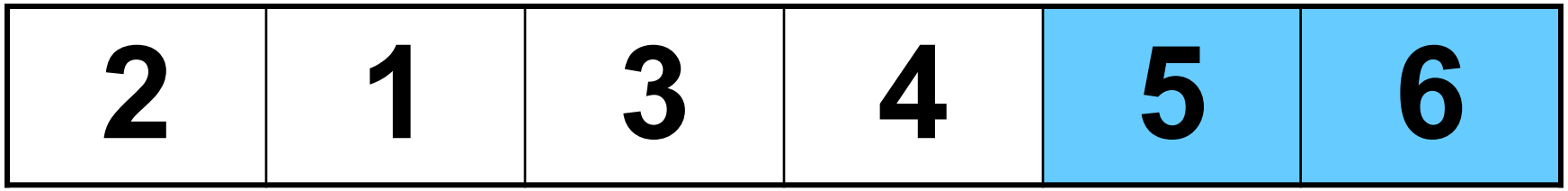


Comparison

Data Movement

Sorted

Selection Sort

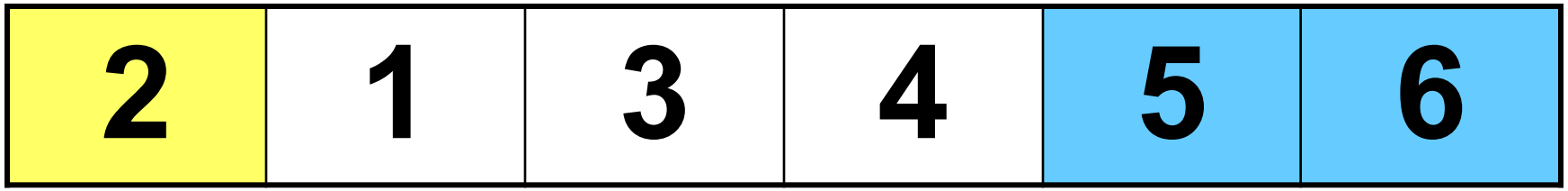


Comparison

Data Movement

Sorted

Selection Sort

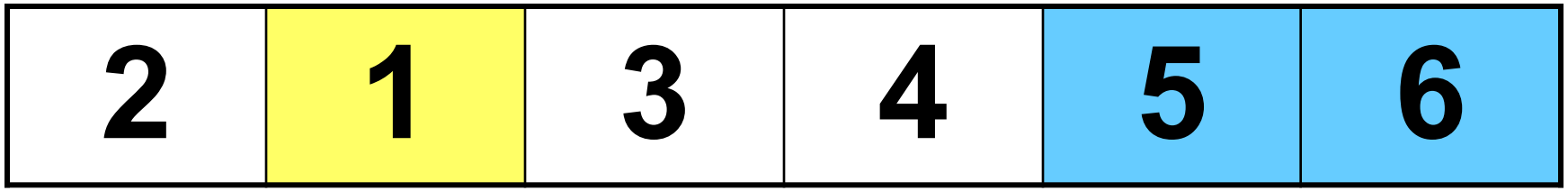


Comparison

Data Movement

Sorted

Selection Sort

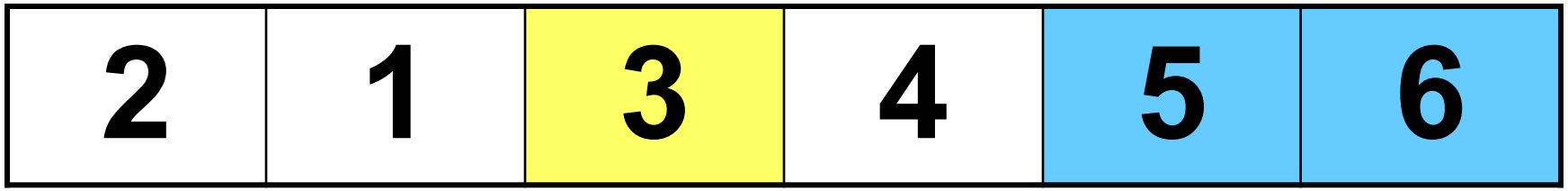


Comparison

Data Movement

Sorted

Selection Sort

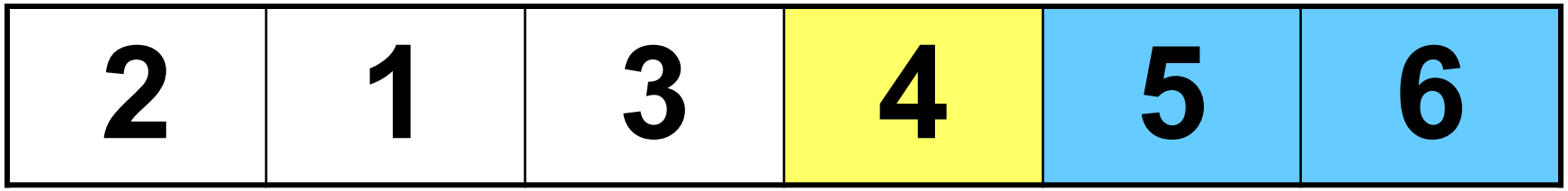


Comparison

Data Movement

Sorted

Selection Sort

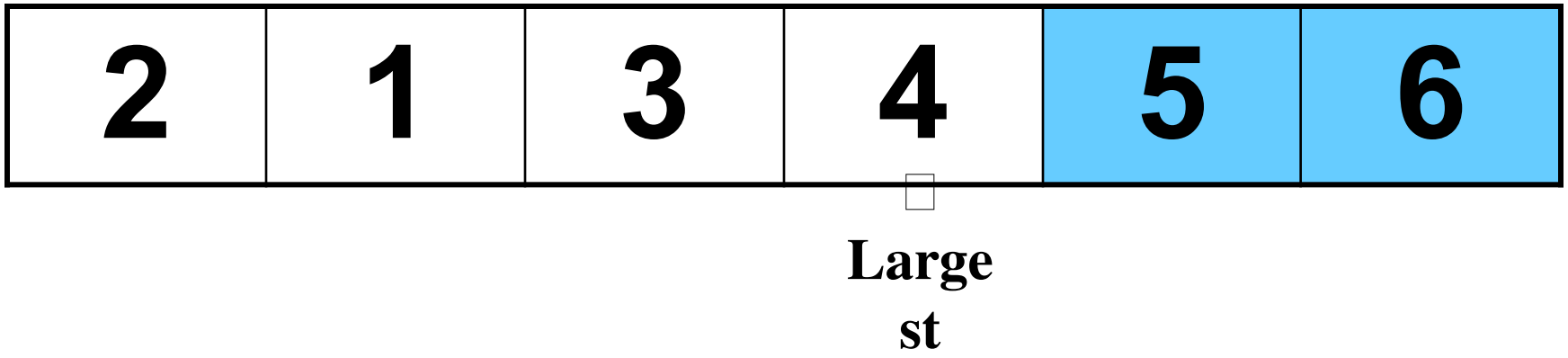


Comparison

Data Movement

Sorted

Selection Sort

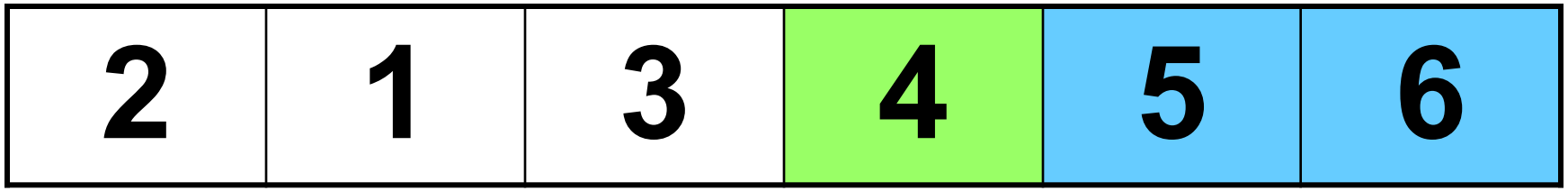


Comparison

Data Movement

Sorted

Selection Sort

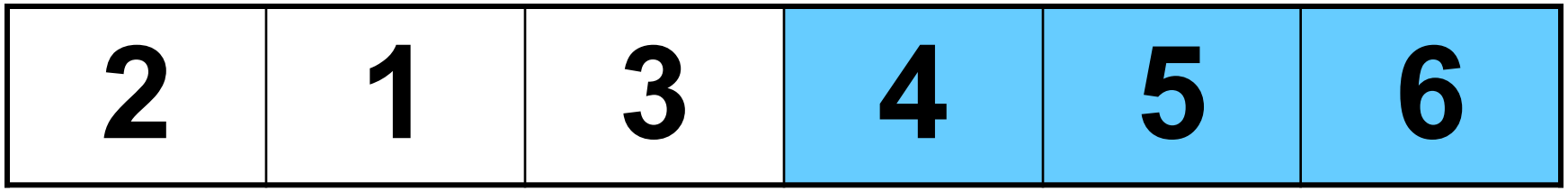


Comparison

Data Movement

Sorted

Selection Sort

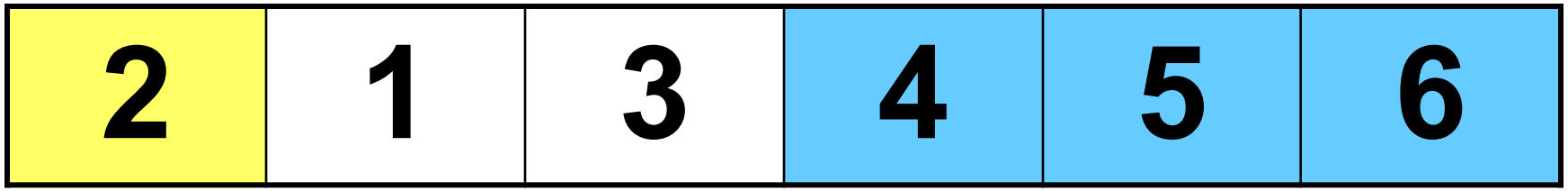


Comparison

Data Movement

Sorted

Selection Sort

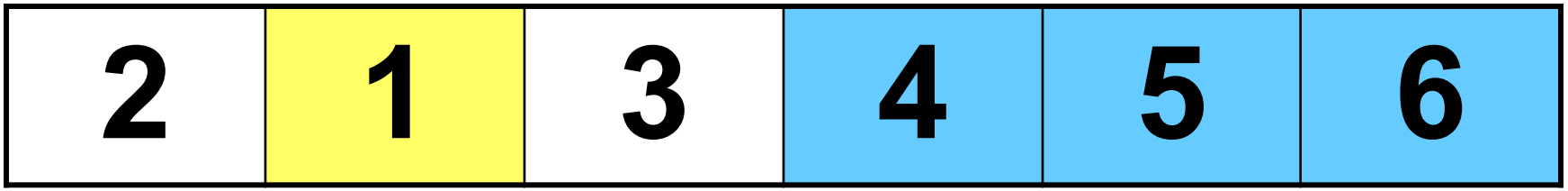


Comparison

Data Movement

Sorted

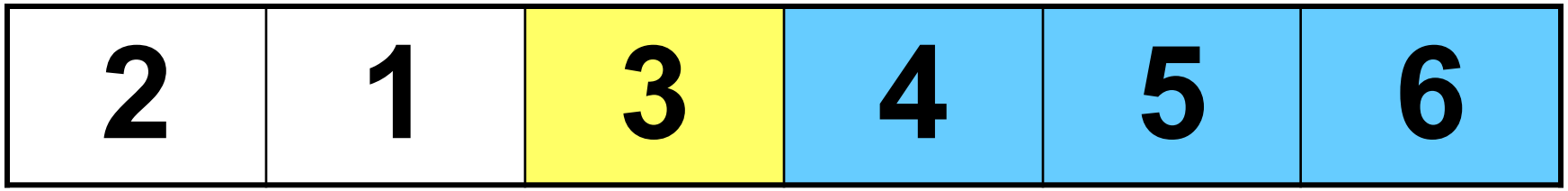
Selection Sort



Comparison

Data Movement

Selection Sort

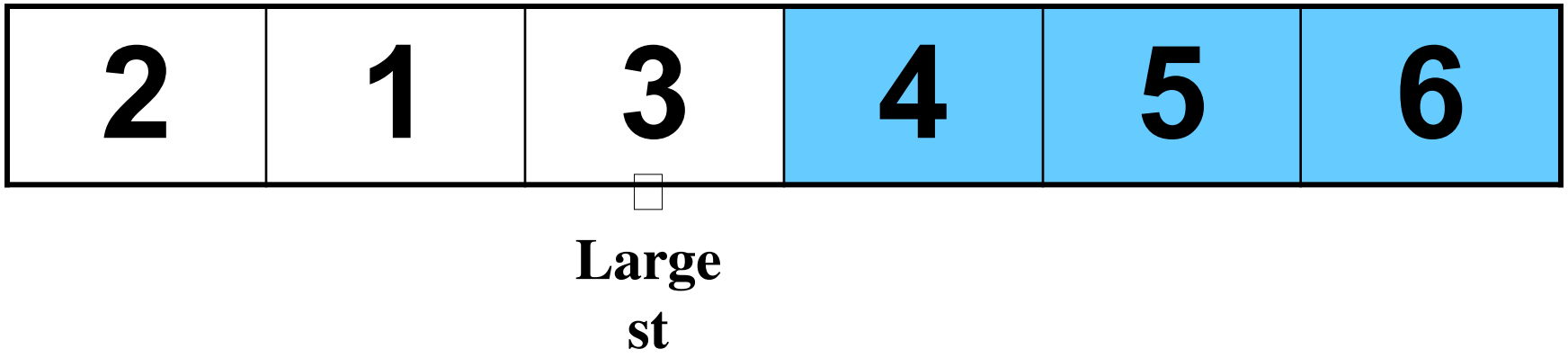


Comparison

Data Movement

Sorted

Selection Sort

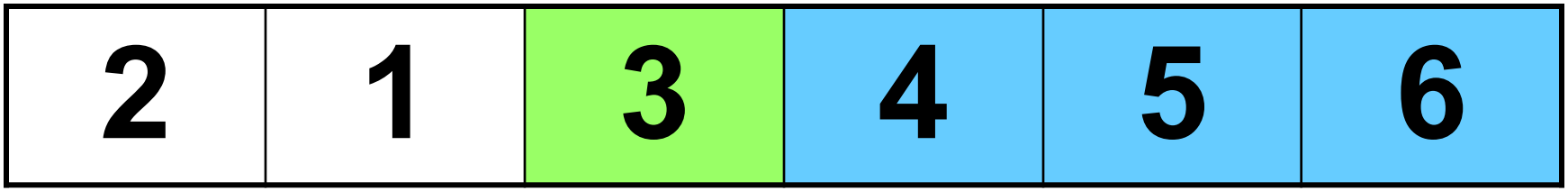


Comparison

Data Movement

Sorted

Selection Sort

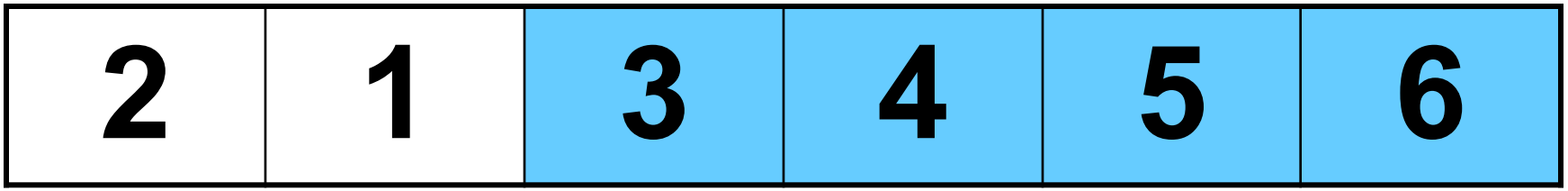


Comparison

Data Movement

Sorted

Selection Sort

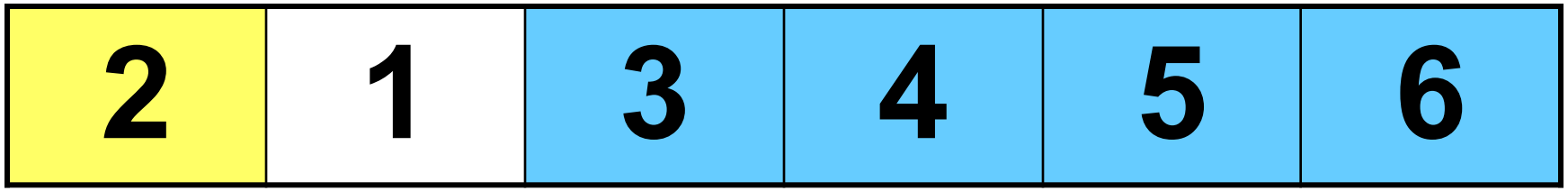


Comparison

Data Movement

Sorted

Selection Sort

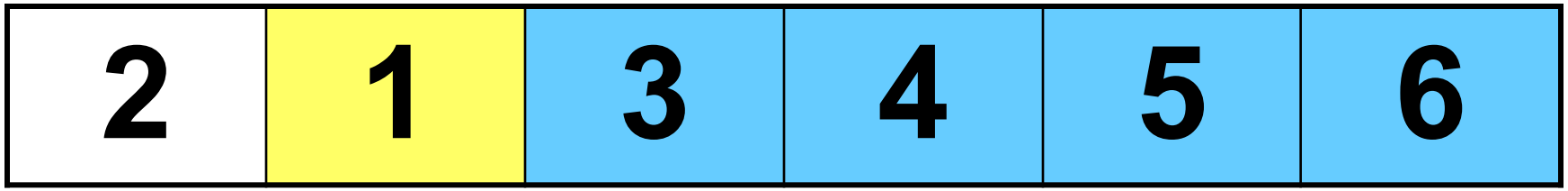


Comparison

Data Movement

Sorted

Selection Sort

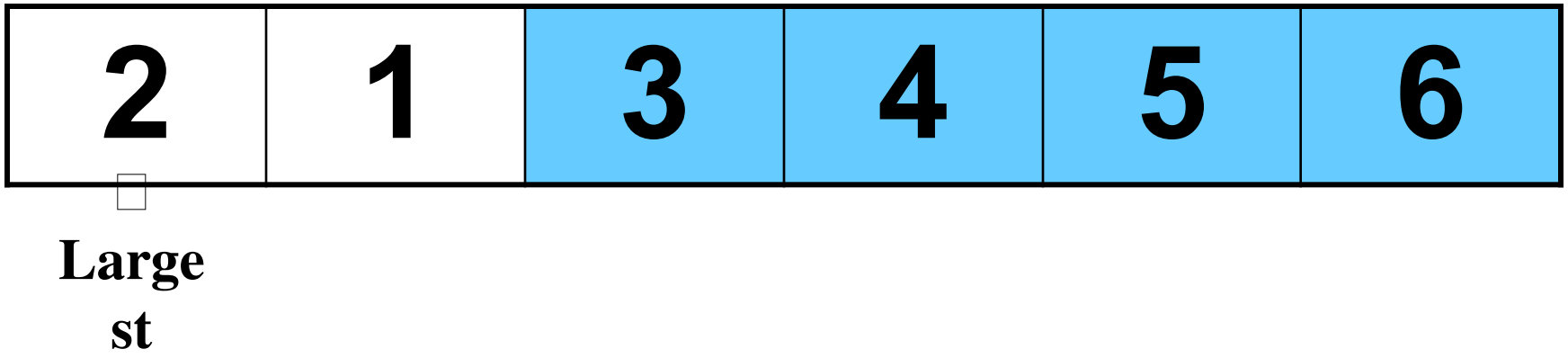


Comparison

Data Movement

Sorted

Selection Sort

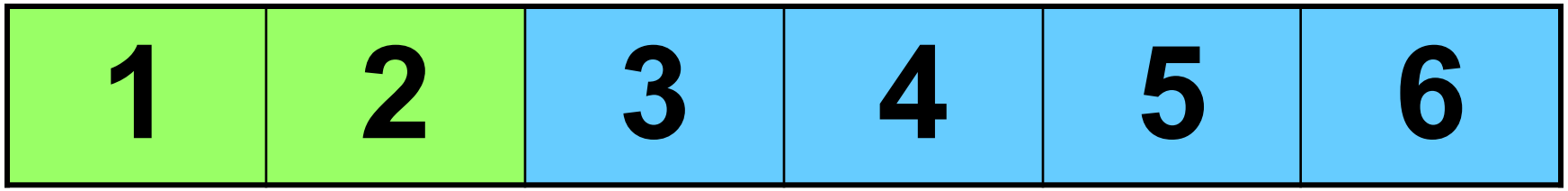


Comparison

Data Movement

Sorted

Selection Sort

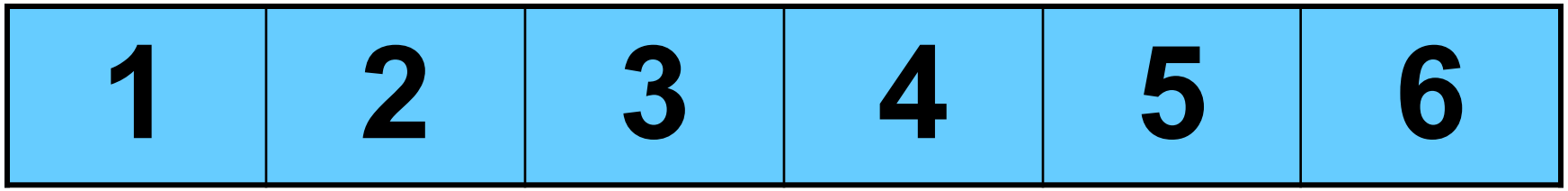


Comparison

Data Movement

Sorted

Selection Sort



DONE!

Comparison

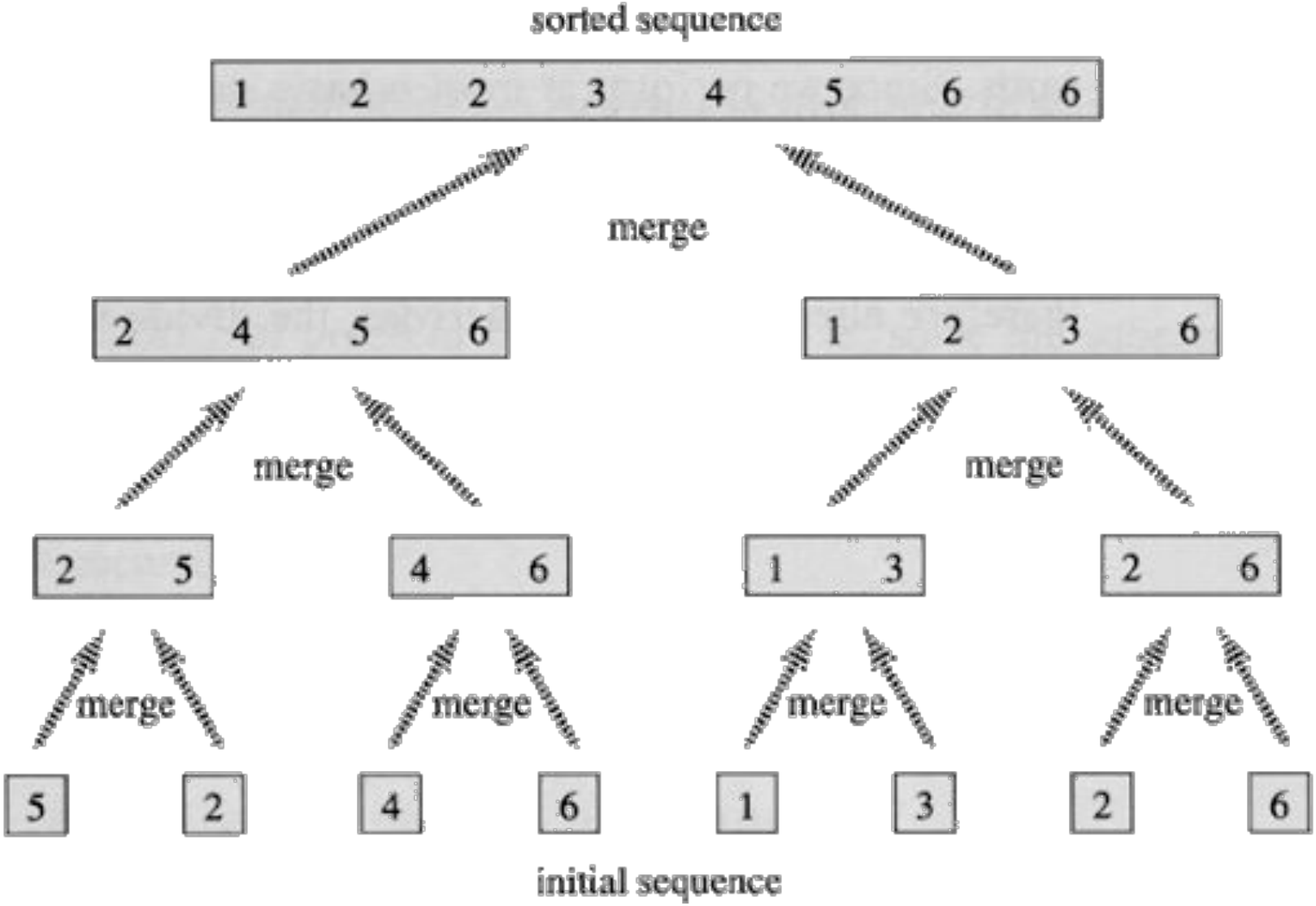
Data Movement

Sorted

Merge Sort

- Merge sort is a sorting technique based on divide and conquer technique. With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Merge Sort

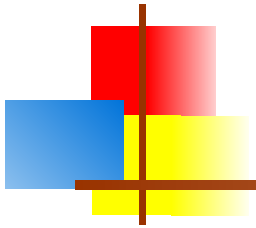

sorted sequence
1 2 2 3 4 5 6 6
merge
2 4 5 6
1 2 3 6
merge
merge
2 5
4 6
1 3
2 6
merge
merge
merge
merge
5
2
4
6
1
3
2
6
initial sequence

Merge Sort

```
MERGE(A, p, q, r)
1.  n_1 = q − p + 1
2.  n_2 = r − q
3.  let L[1..n_1 + 1] and R[1..n_2 + 1] be new arrays
4.  for i = 1 to n_1
5.      L[i] = A[p + i − 1]
6.  for j = 1 to n_2
7.      R[j] = A[q + j]
8.  L[n_1 + 1] = ∞
9.  R[n_2 + 1] = ∞
10. i = 1
11. j = 1
12. for k = p to r
13.     if L[i] ≤ R[j]
14.         A[k] = L[i]
15.         i = i + 1
16.     else A[k] = R[j]
17.         j = j + 1
```

Merge Sort

```
MERGE-SORT(A, p, r)
1.  if p < r
2.          q = ⌊(p + r)/2⌋
3.          MERGE-SORT(A, p, q)
4.          MERGE-SORT(A, q+1, r)
5.          MERGE(A, p, q, r)
```



The End