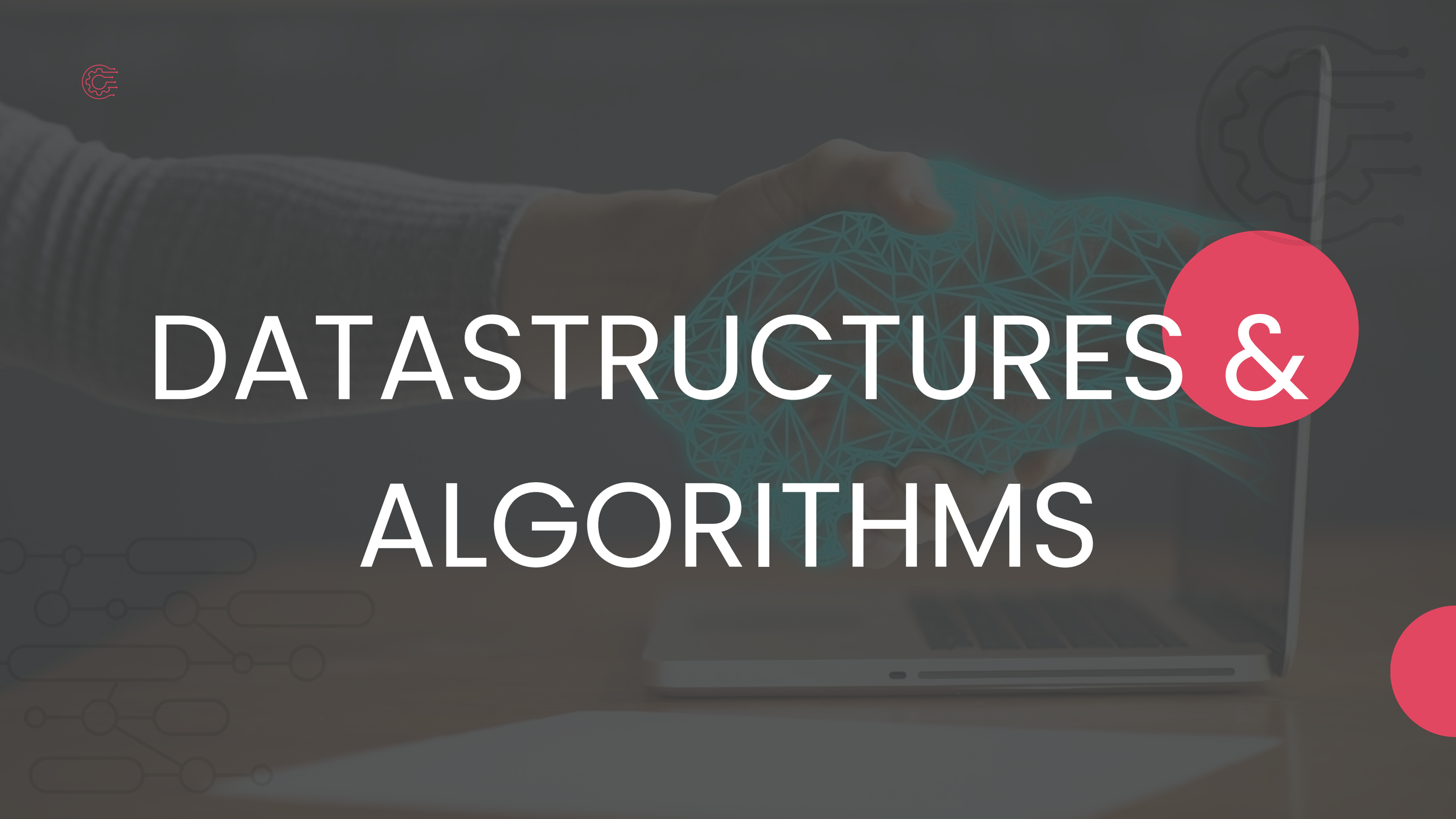


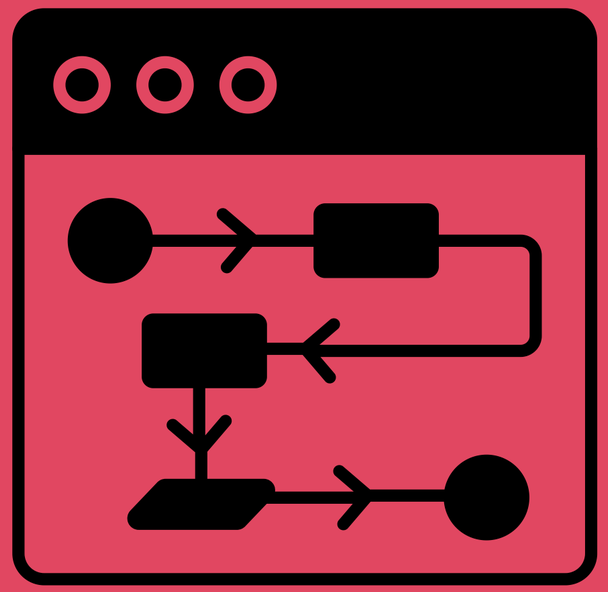
DATASTRUCTURES & ALGORITHMS

Introduction to Algorithms

Mahreeba Saleem

What is algorithm??

A list or set of instructions, used to solve problems or perform tasks



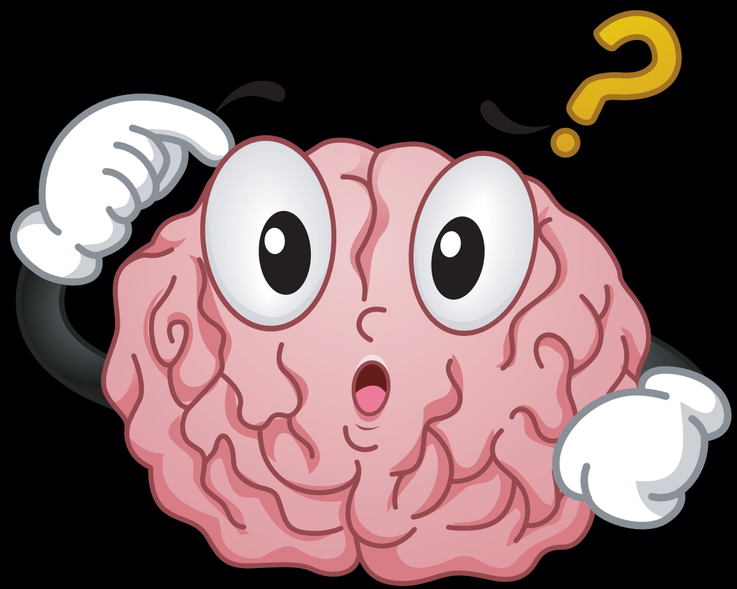
Algorithm to cross road

- Look left
- Look Right
- if no car is in close proximity
- Walk

Algorithms in computer world

Computer encoded instructions to solve problem

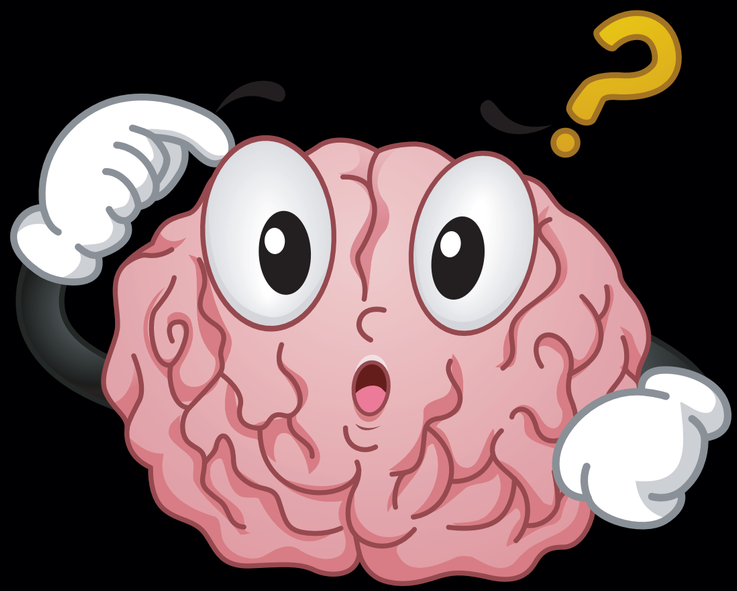
Why algorithms???



Why algorithms???

Computers are dumb machines so we developed algorithms so that we can use them to solve our problems.

How good an algorithm is??

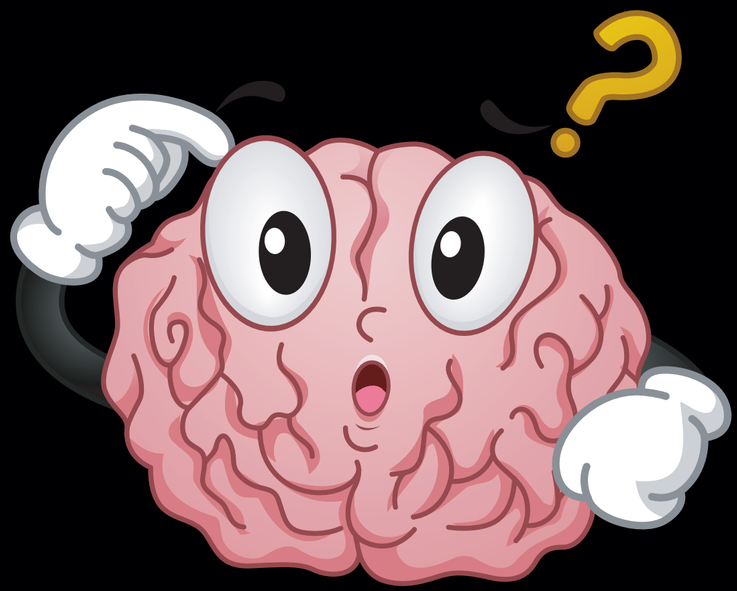


How good an algorithm is??

- How quickly it solves.
- How less memory it takes.

How good an algorithm is??

- Time Complexity
- Space Complexity



Different types of algorithms

- Searching
- Sorting
- Encryption
- Recursive

etc...

Sorting algorithms

Algorithms for reorganizing a large number of items into a specific order

3

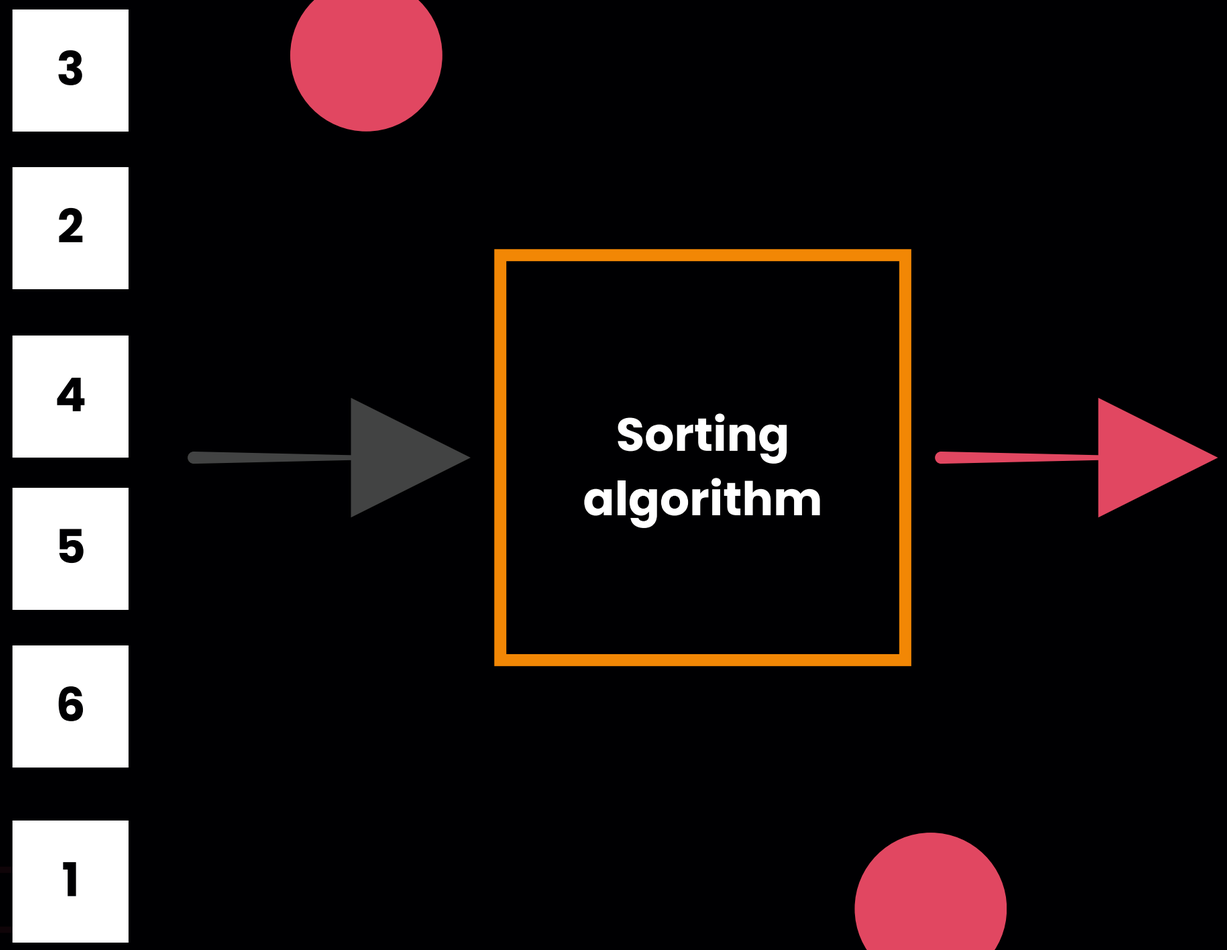
2

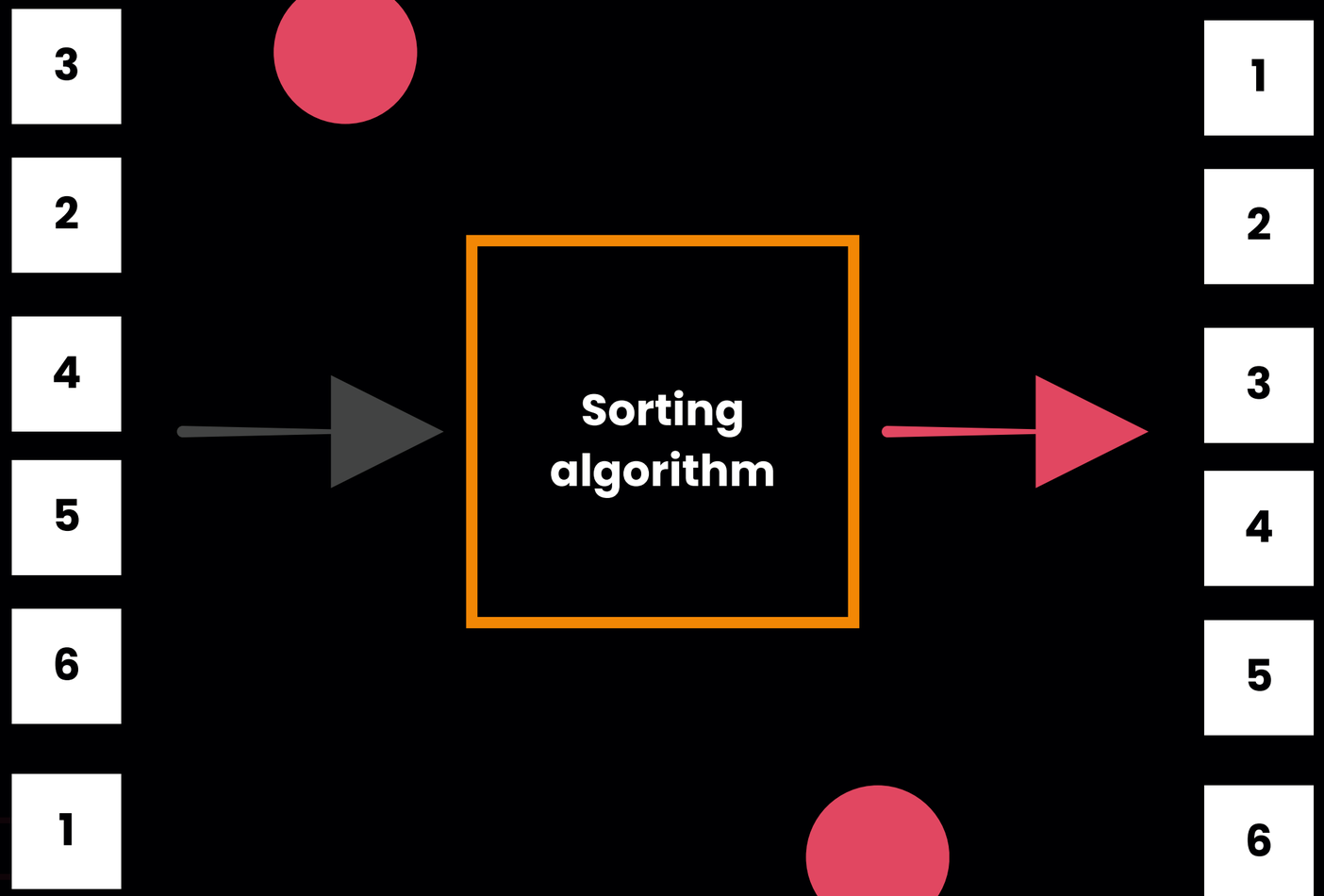
4

5

6

1


3
2
4
5
6
1
Sorting algorithm


3
2
4
5
6
1
Sorting algorithm
1
2
3
4
5
6

Sorting algorithms

- Bubble sort
- Selection sort
- Insertion sort
- Merge sort

Selection Sort

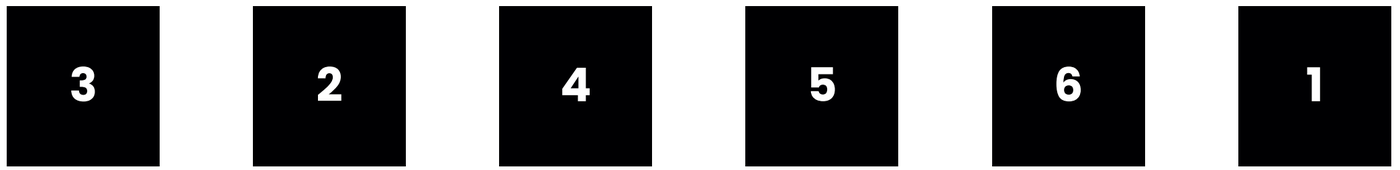
Maham Saleem

Selection Sort

- Start with the first element of the array as the "minimum" value.
- Compare this minimum value with the rest of the elements in the array.
- If you find a smaller value, update the minimum value to that element.
- Continue this process until you reach the end of the array, finding the smallest element.
- Swap the smallest element with the first element in the array.
- Repeat the above steps for the remaining unsorted portion of the array, excluding the already sorted elements.

```
#include <iostream>
#include <algorithm>
using namespace std;
void selectionSort(int arr[], int n) {
  for (int i = 0; i < n - 1; ++i) {
    int minIndex = i;
    // Find the index of the minimum element in the unsorted part of the array
    for (int j = i + 1; j < n; ++j) {
      if (arr[j] < arr[minIndex]) {
        minIndex = j;
      }
    }
    // Swap the found minimum element with the first element using swap
    swap(arr[i], arr[minIndex]);
  }
}

void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
}
```

```
int main() {
  int arr[] = {64, 25, 12, 22, 11};
  int n = sizeof(arr) / sizeof(arr[0]);

  cout << "Original array: ";
  printArray(arr, n);

  // Perform selection sort
  selectionSort(arr, n);

  cout << "Sorted array: ";
  printArray(arr, n);

  return 0;
}
```


3
2
4
5
6
1

1. Start with first element.

2. Compare with rest of array

2. Compare with rest of array

2. Compare with rest of array

2. Compare with rest of array

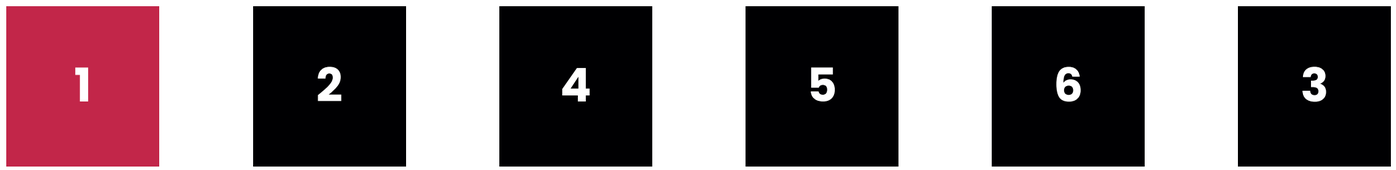
2. Compare with rest of array

2. Finding the smallest element, replace it with smallest

2. Finding the smallest element, replace it with smallest

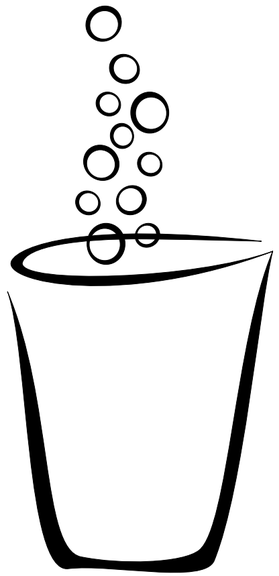
2. Finding the smallest element, replace it with smallest

Repeat same for remaining.



TIME COMPLEXITY	
BEST	O(n^2)
AVG.	O(n^2)
WORST	O(n^2)

Bubble Sort



Abdullah

Bubble algorithm

1. Start with an array of elements.
2. Iterate through the array from the first element to the second-to-last element.
3. Compare each pair of adjacent elements.
4. If the elements are in the wrong order, swap them.
5. Repeat this process for each pair of adjacent elements, moving from left to right.
6. After each iteration, the largest element will "bubble" up to its correct position at the end of the array.
7. Repeat steps 2-6 for a total number of iterations equal to the length of the array minus one.
8. The array is now sorted in ascending order.

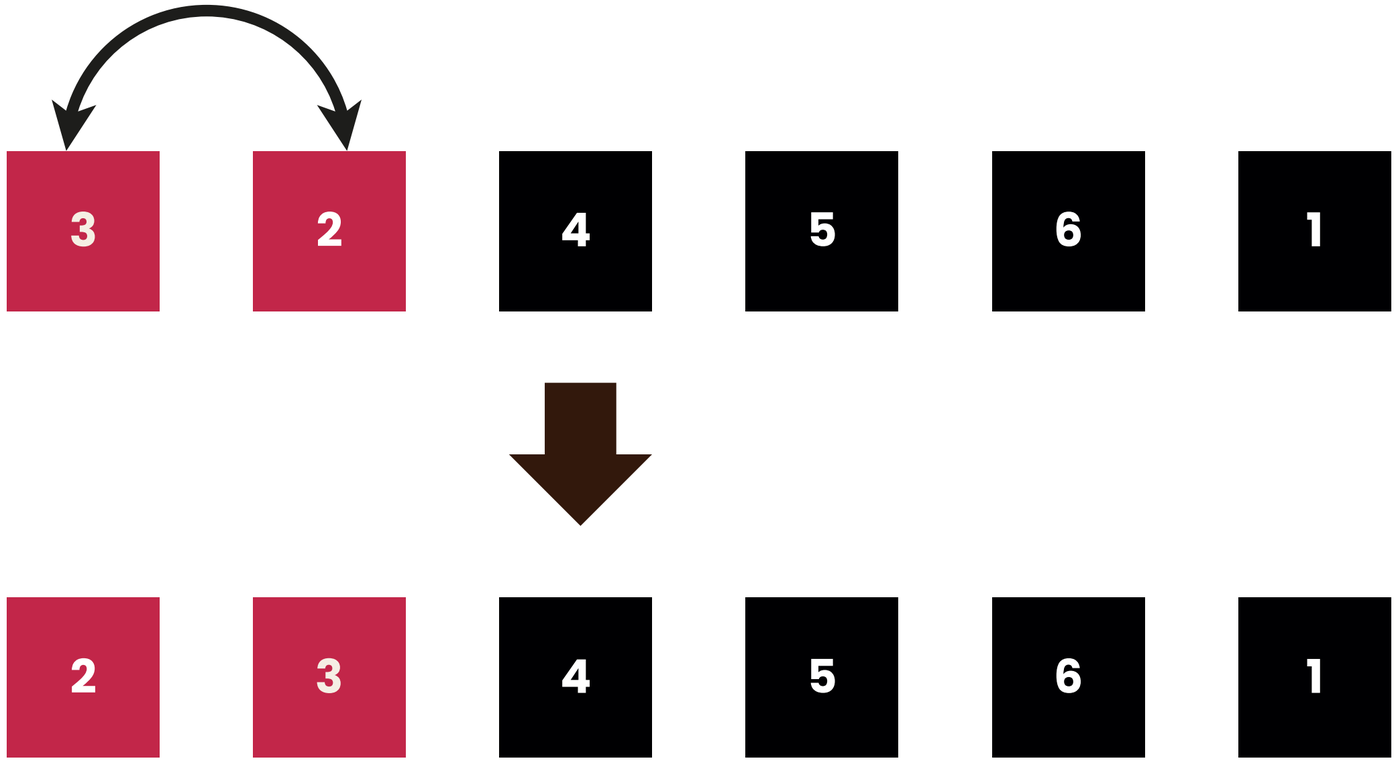
```cpp
#include<iostream>
using namespace std;
    void print(int a[], int n) //function to print array ele
    {
    int i;
    for(i = 0; i < n; i++)
    {
       cout<<a[i]<<" ";
    }
    }
 void bubble(int a[], int n) // function to implement bubble
 {
 int i, j, temp;
   for(i = 0; i < n; i++)
    {
      for(j = i+1; j < n; j++)
        {
            if(a[j] < a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }

 }
```

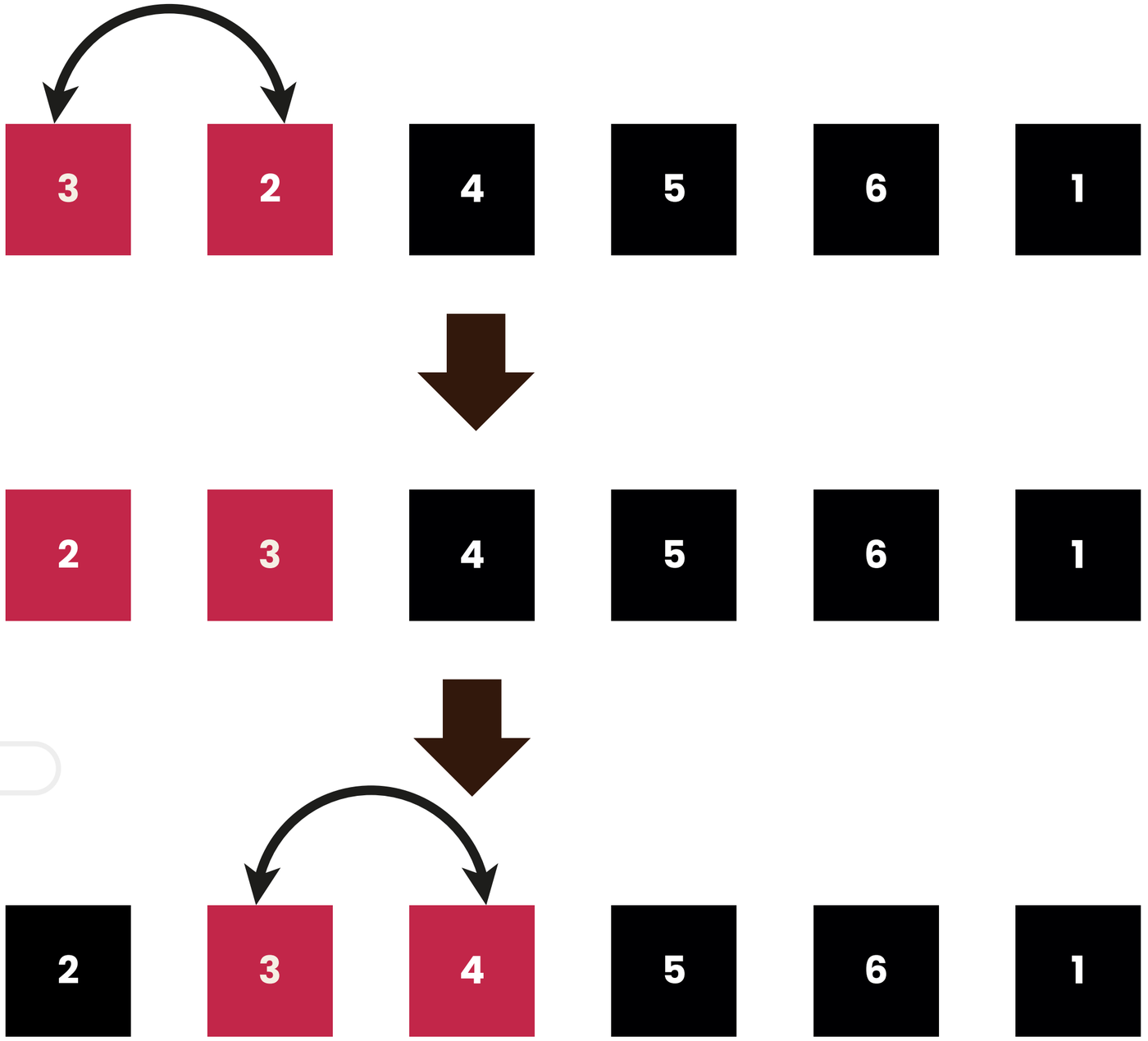
```cpp
int main()
{
    int i, j,temp;
    int a[6] = {3,2,4,5,6,1};
    int n = sizeof(a)/sizeof(a[0]);
    cout<<"Before sorting array elements are - \n";
    print(a, n);
    bubble(a, n);
    cout<<"\nAfter sorting array elements are - \n";
    print(a, n);
return 0;
}

```

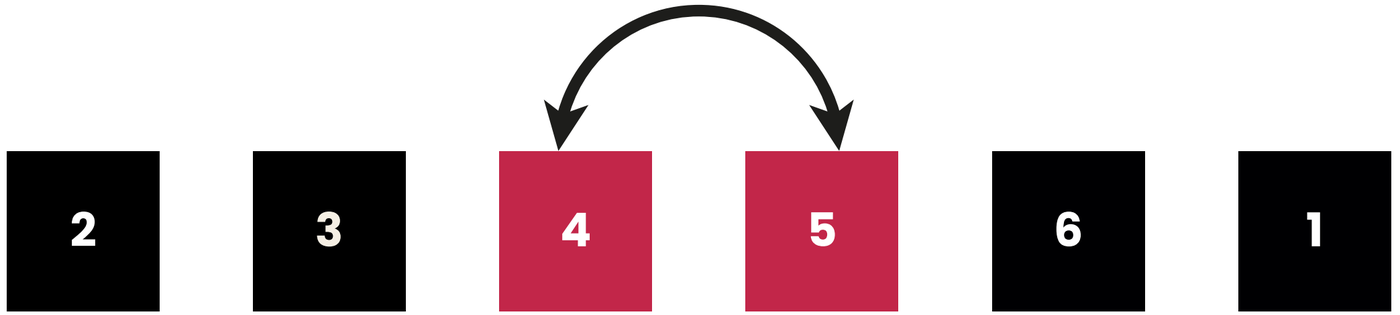
3 2 4 5 6 1

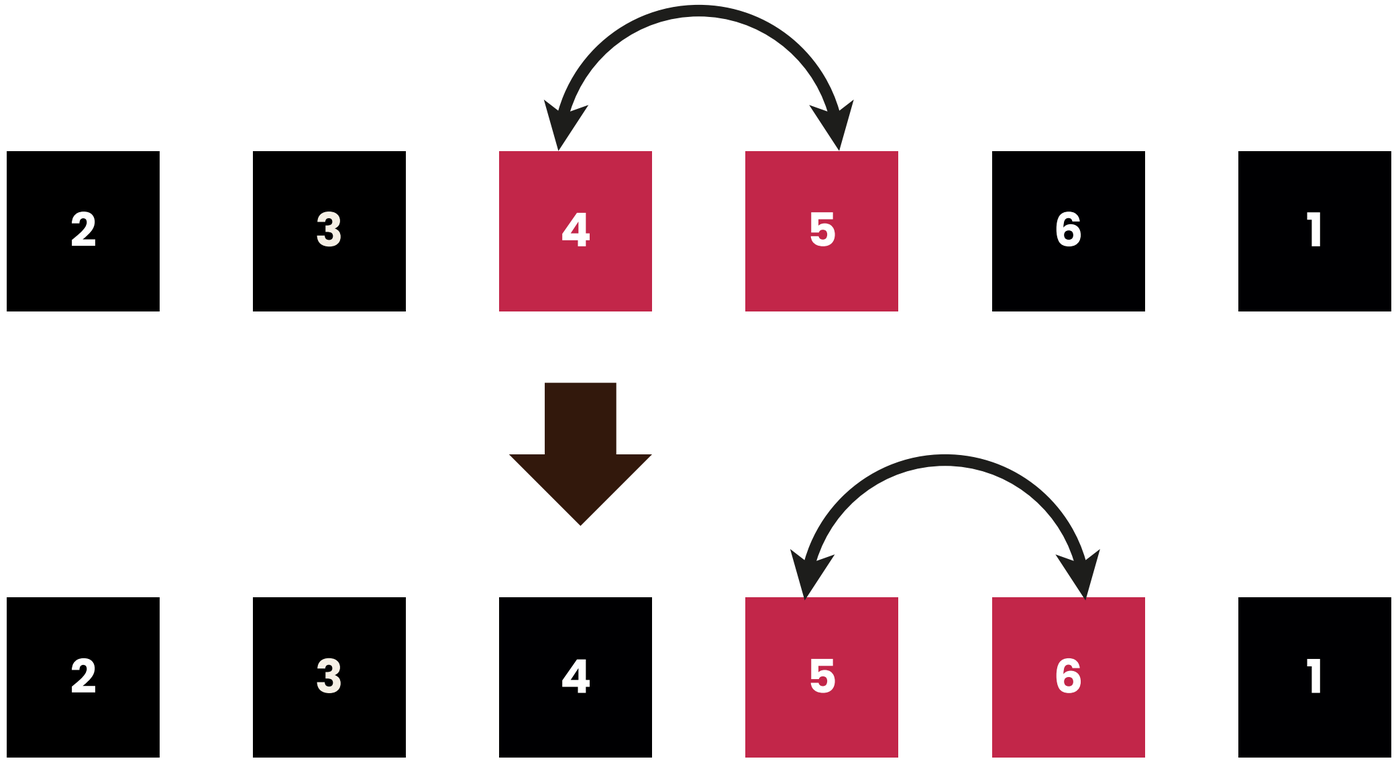
First pass

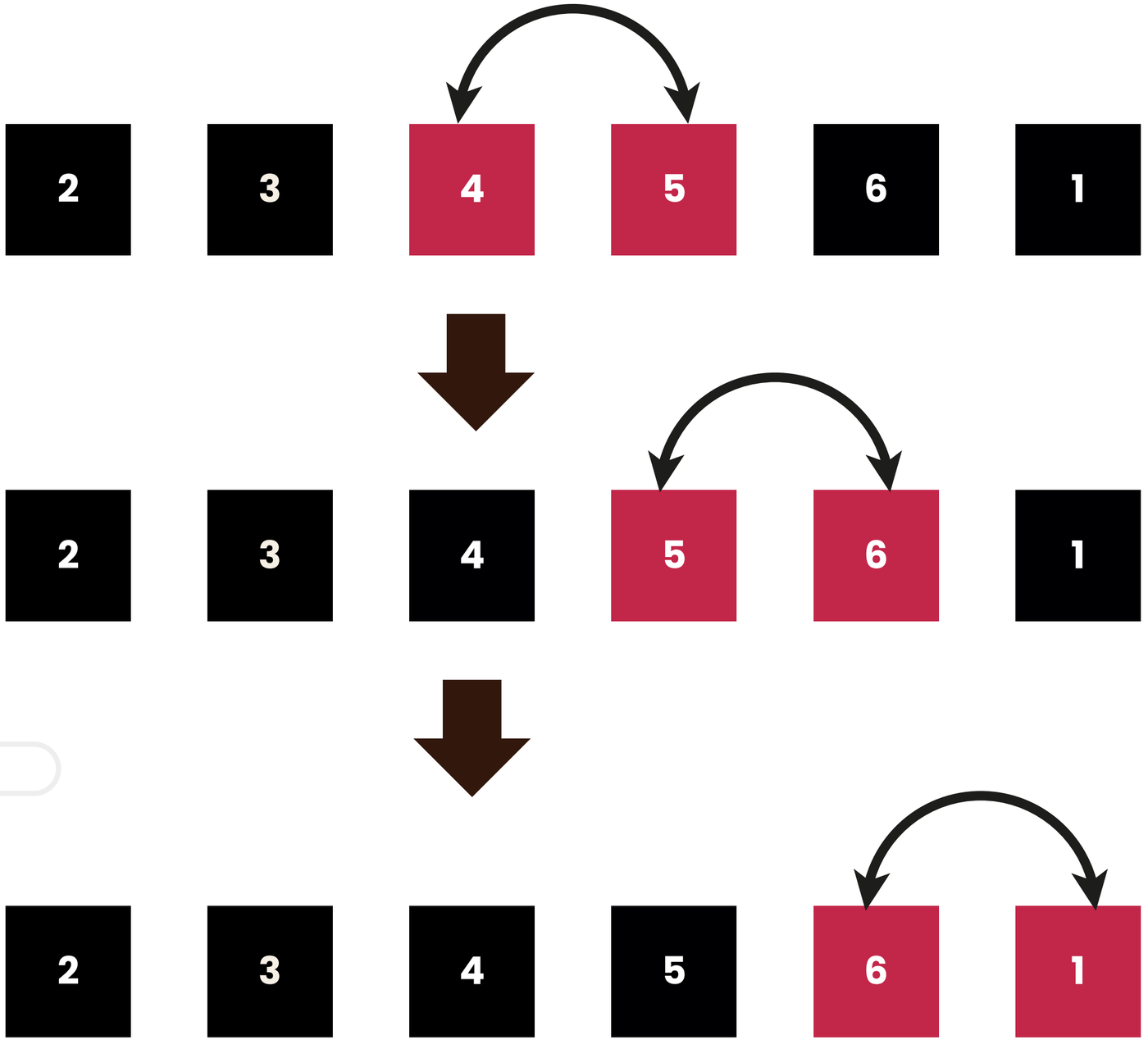

3
2
4
5
6
1

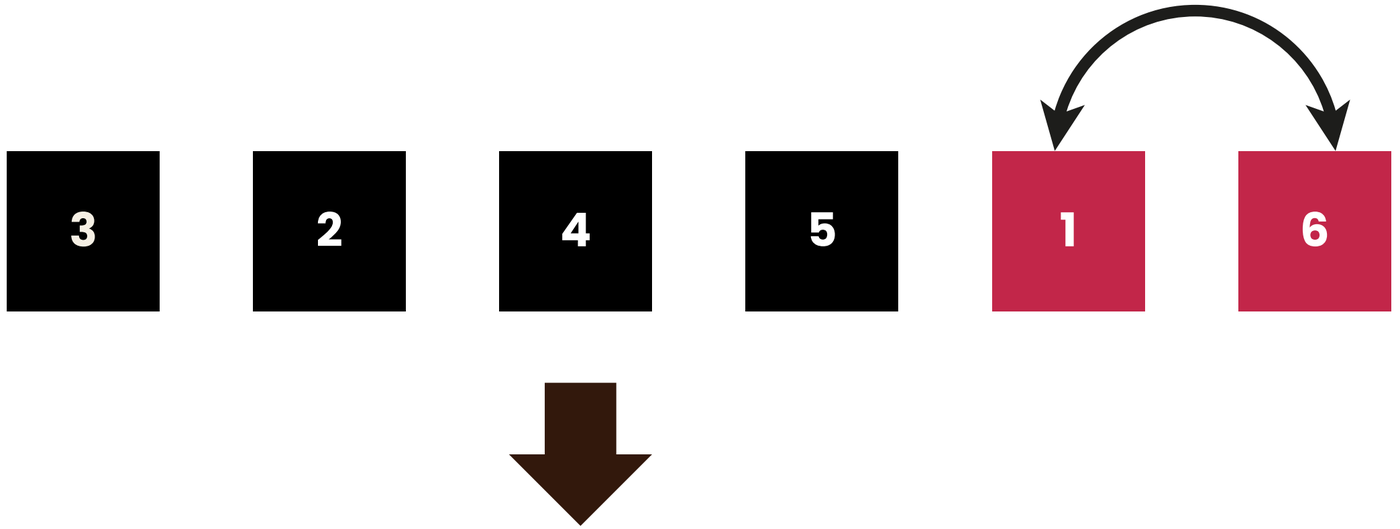

3
2
4
5
6
1
2
3
4
5
6
1


3
2
4
5
6
1
2
3
4
5
6
1
2
3
4
5
6
1


2
3
4
5
6
1


2
3
4
5
6
1
2
3
4
5
6
1

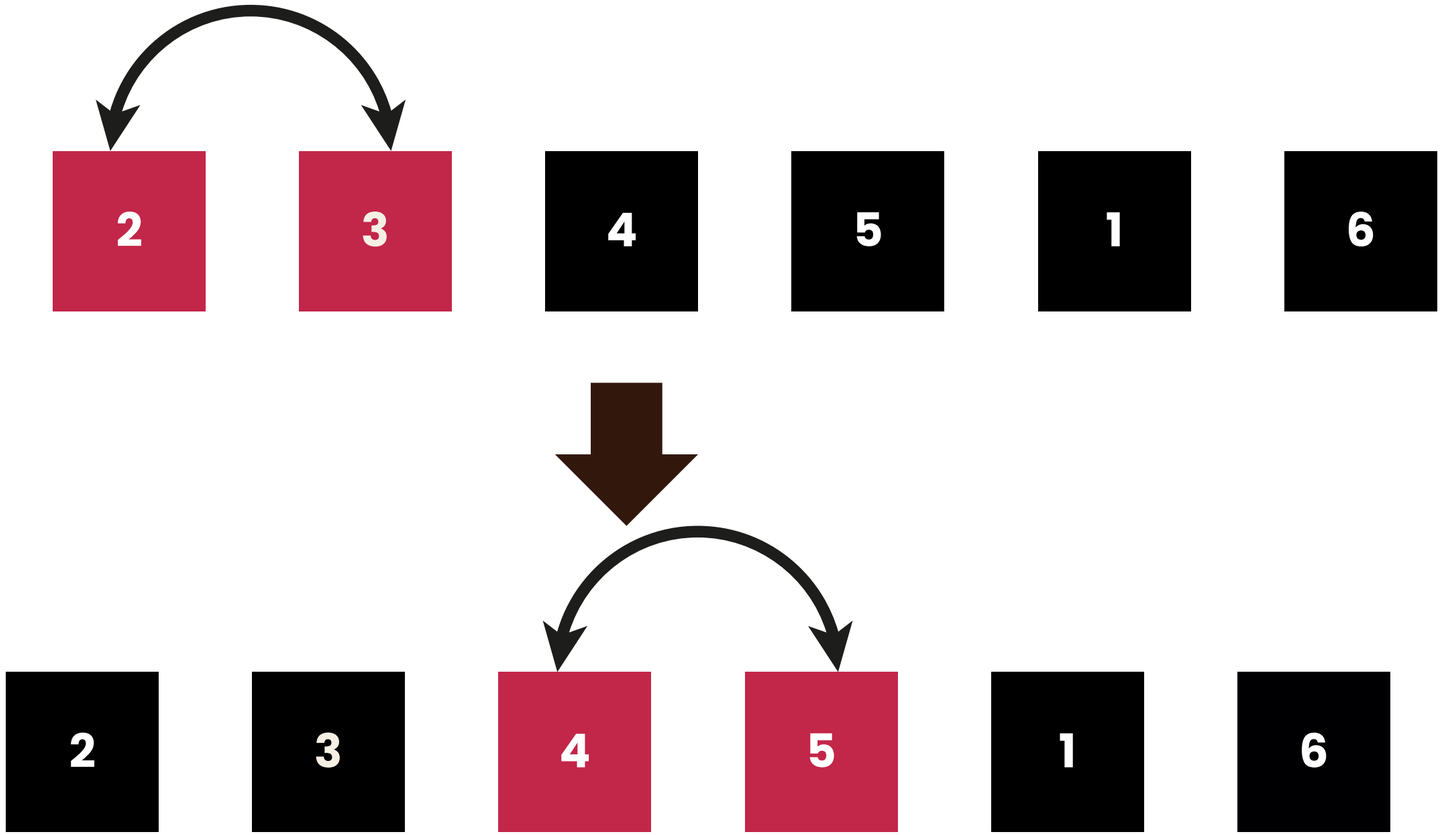

2
3
4
5
6
1
2
3
4
5
6
1
2
3
4
5
6
1

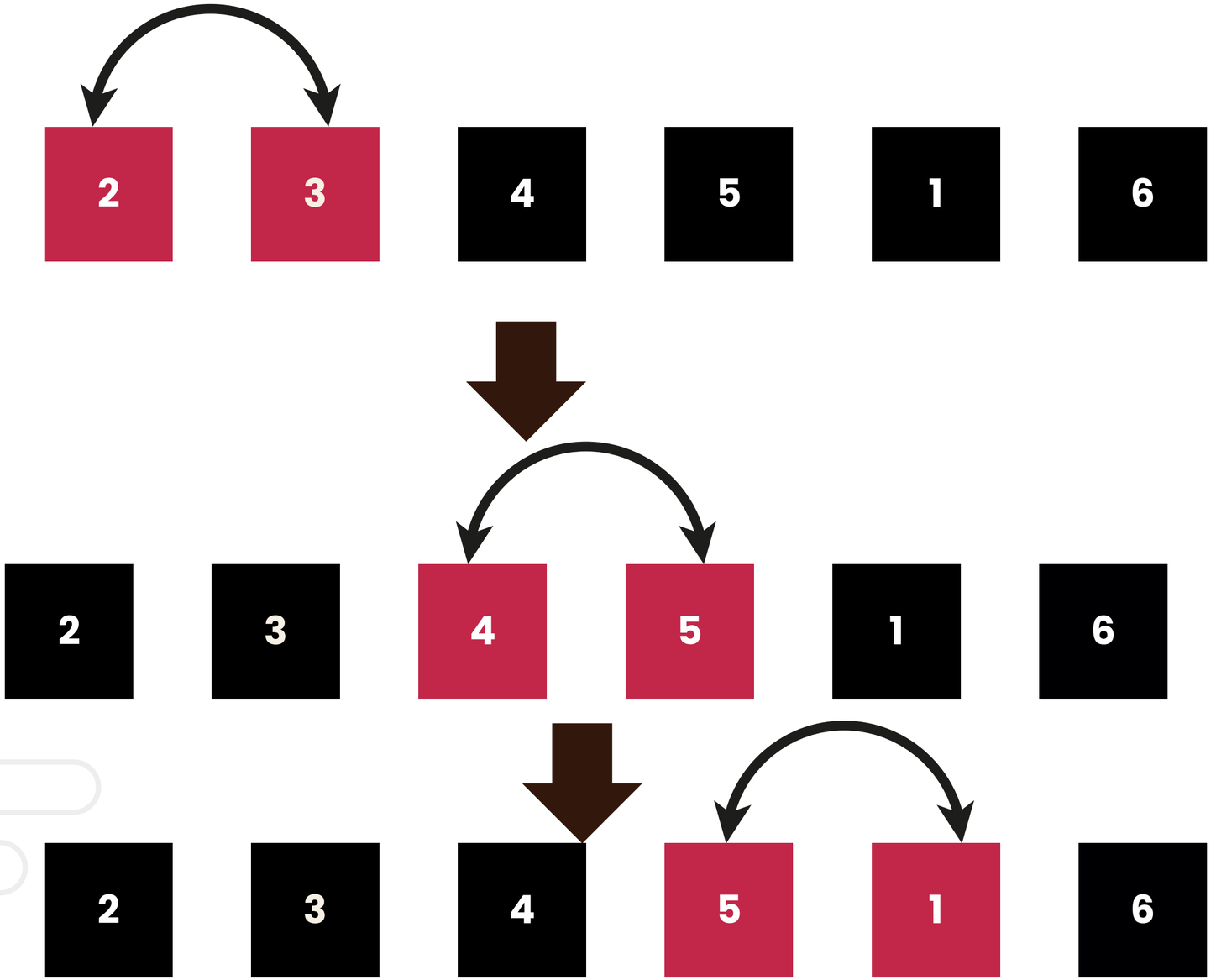


After first pass:

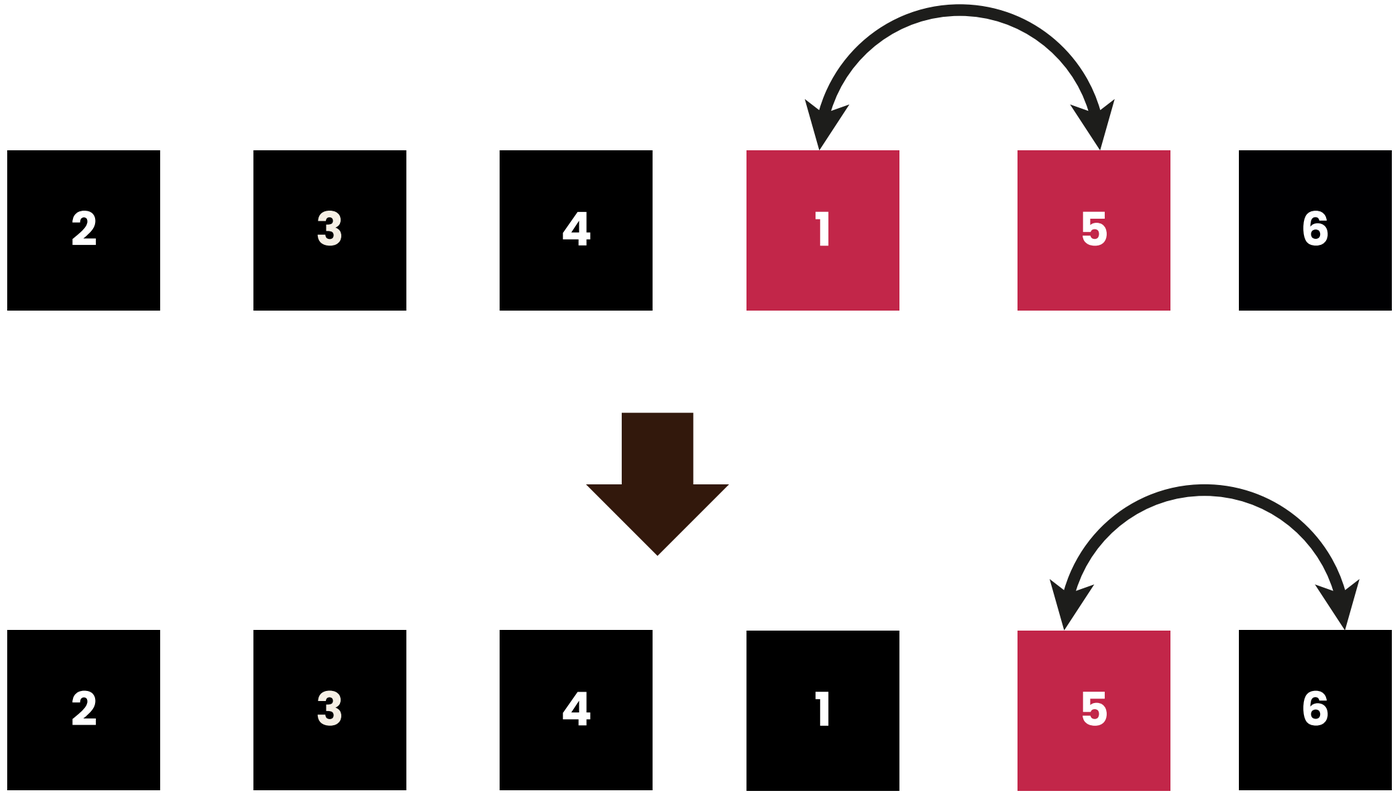
Second pass


2
3
4
5
1
6


2
3
4
5
1
6
2
3
4
5
1
6

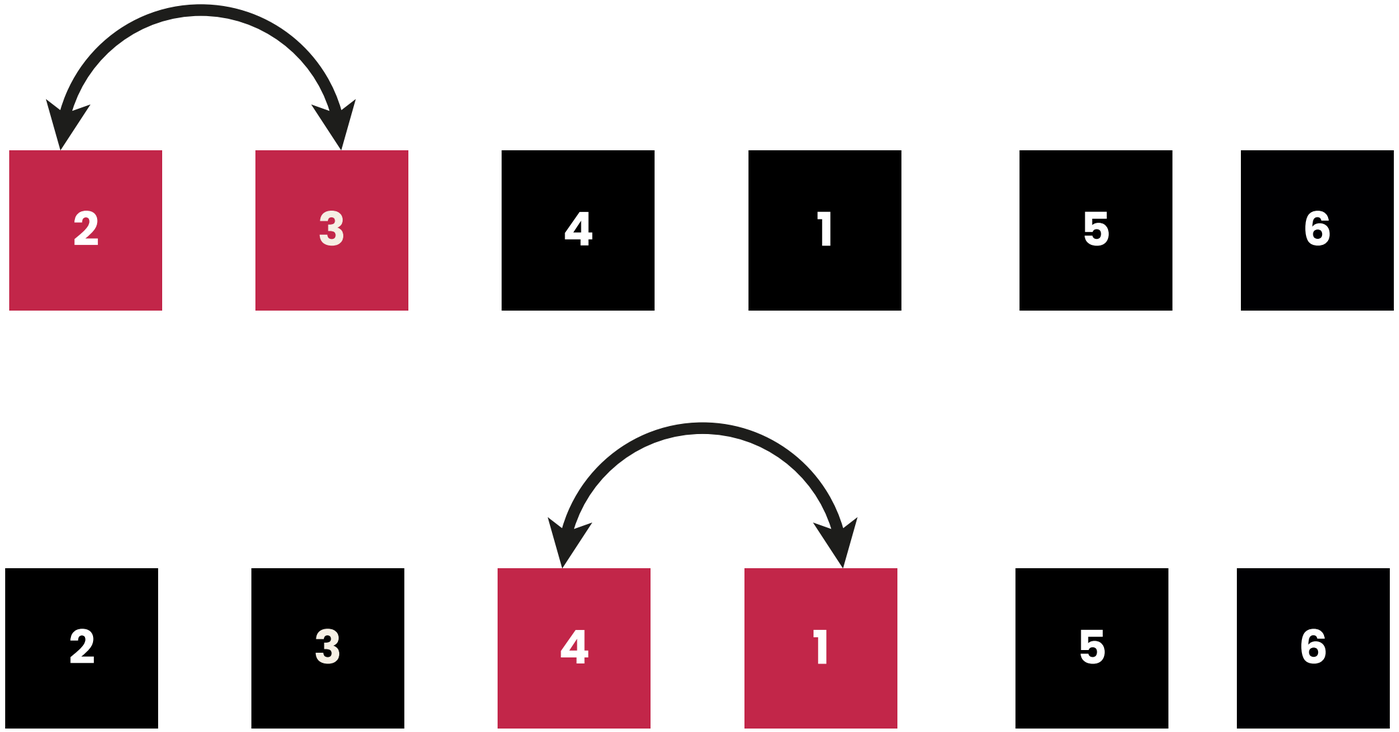

2
3
4
5
1
6
2
3
4
5
1
6
2
3
4
5
1
6

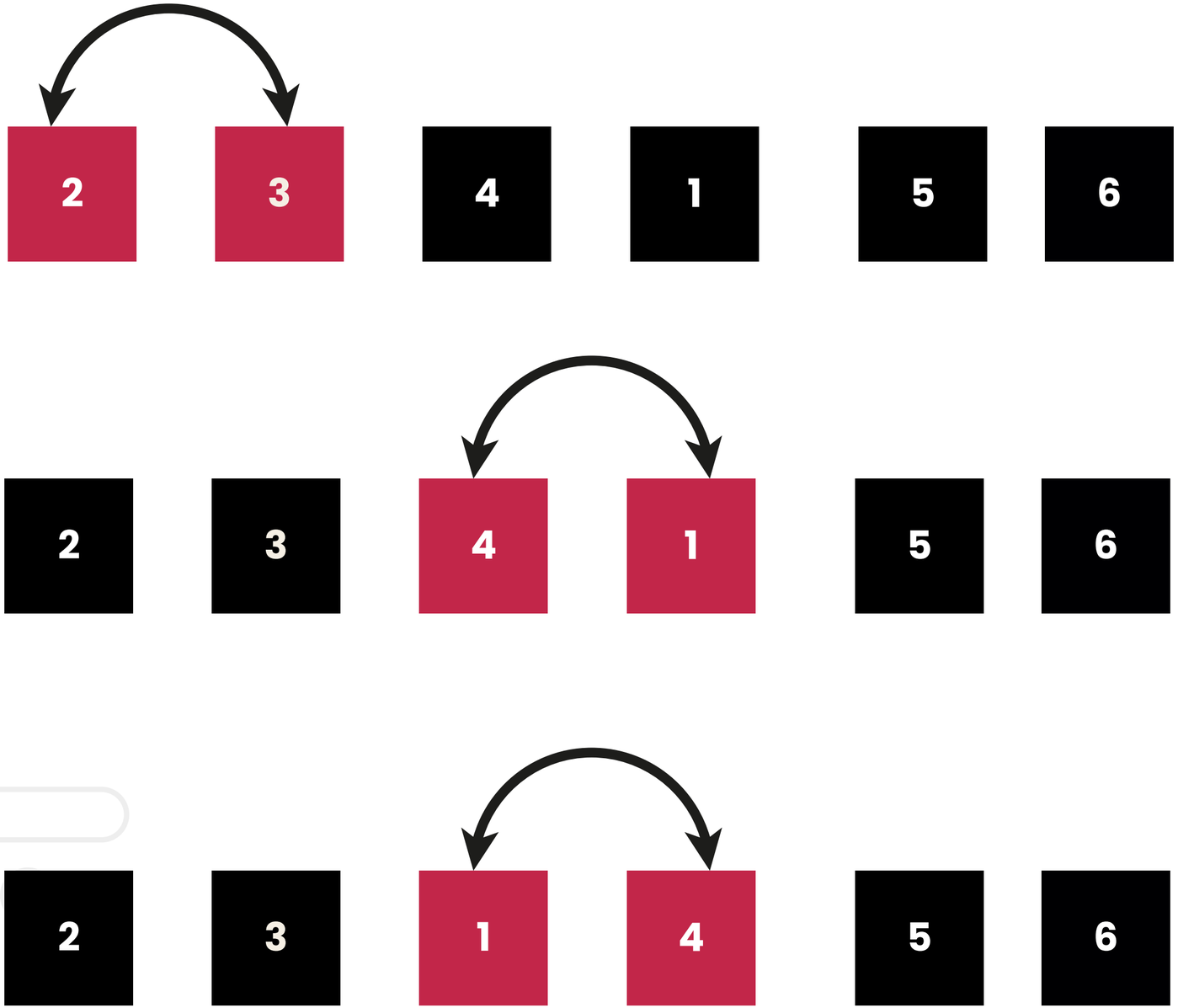

2
3
4
1
5
6

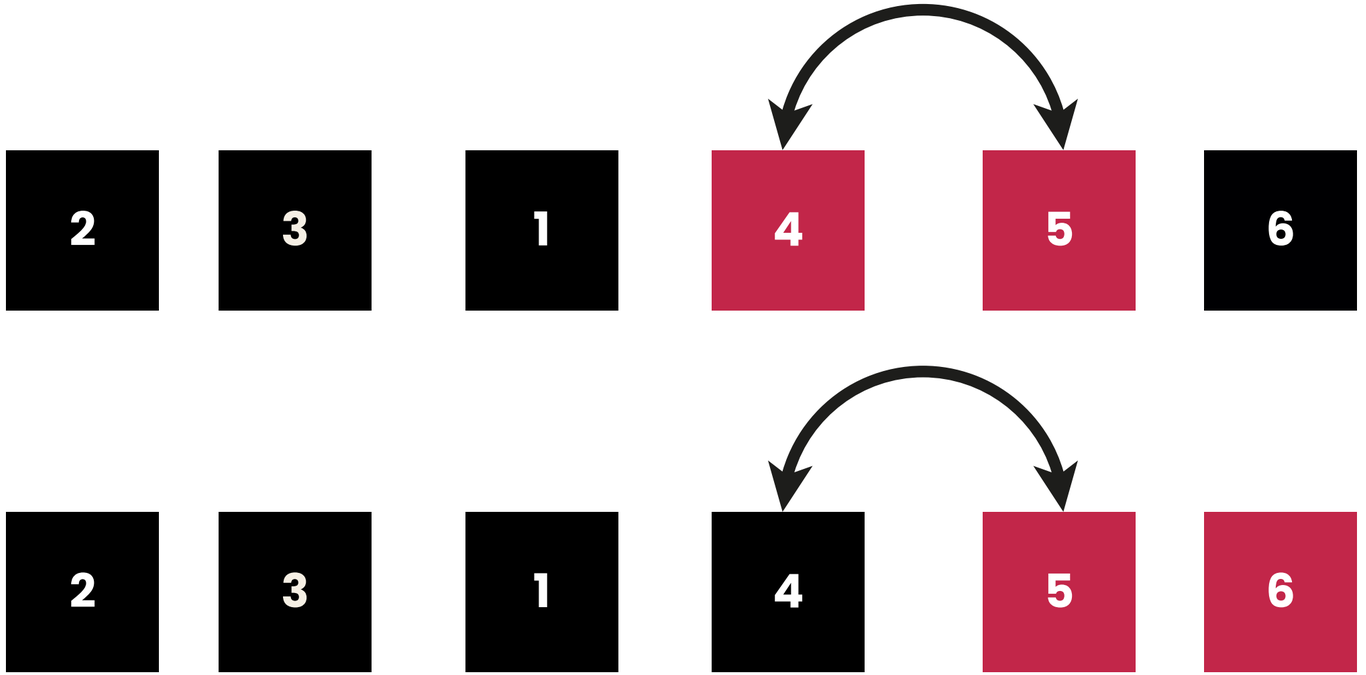

2
3
4
1
5
6
2
3
4
1
5
6

Third pass


2
3
4
1
5
6


2
3
4
1
5
6
2
3
4
1
5
6

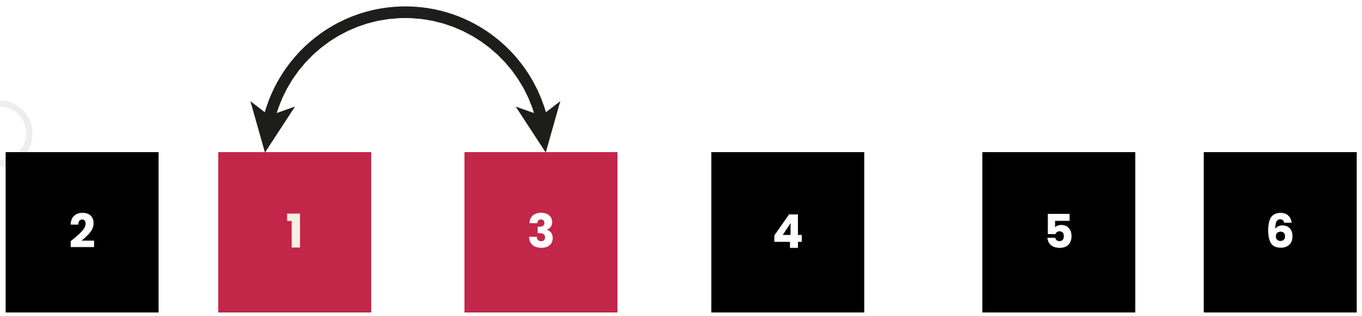

2
3
4
1
5
6
2
3
4
1
5
6
2
3
1
4
5
6

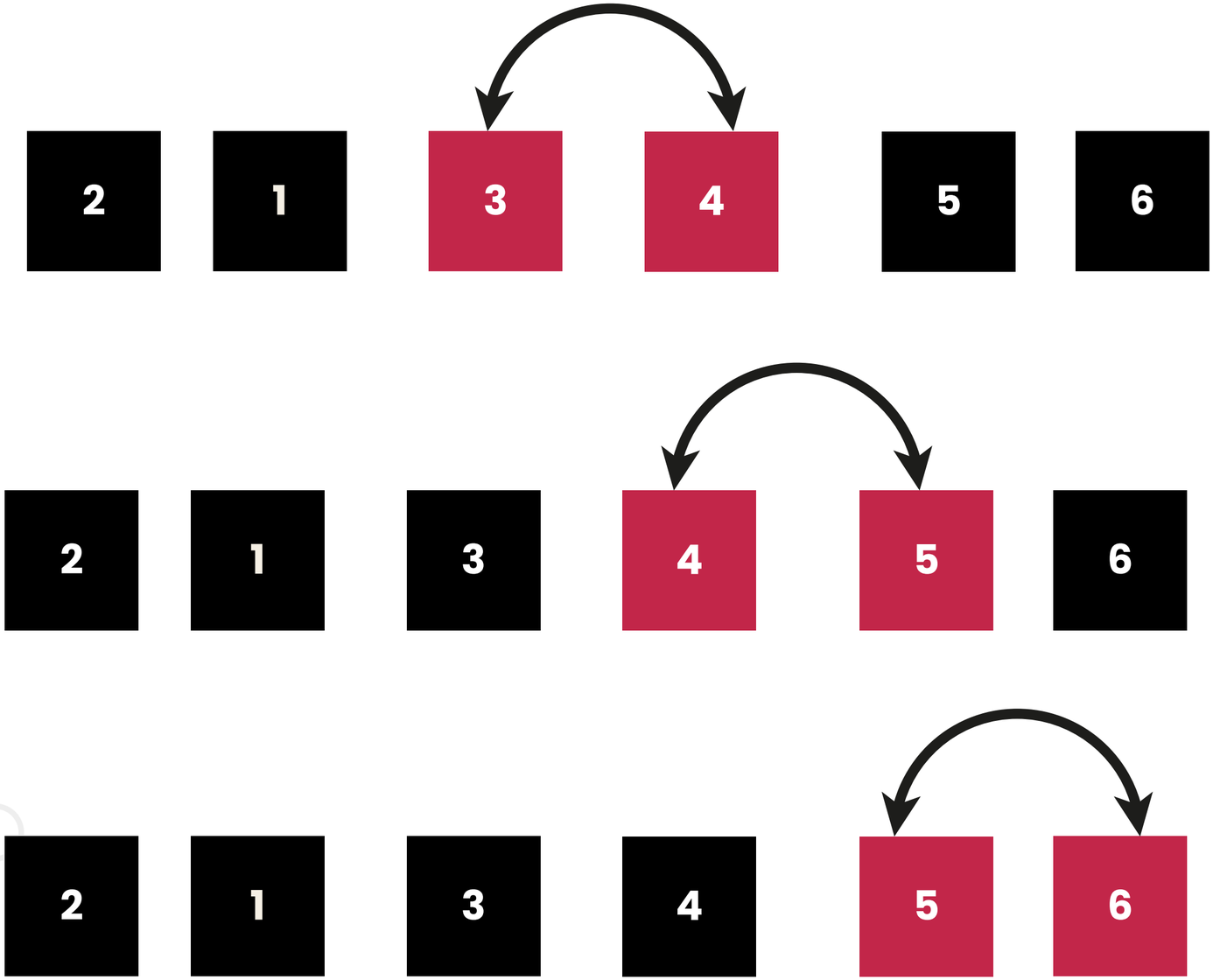

2
3
1
4
5
6
2
3
1
4
5
6

Fourth pass

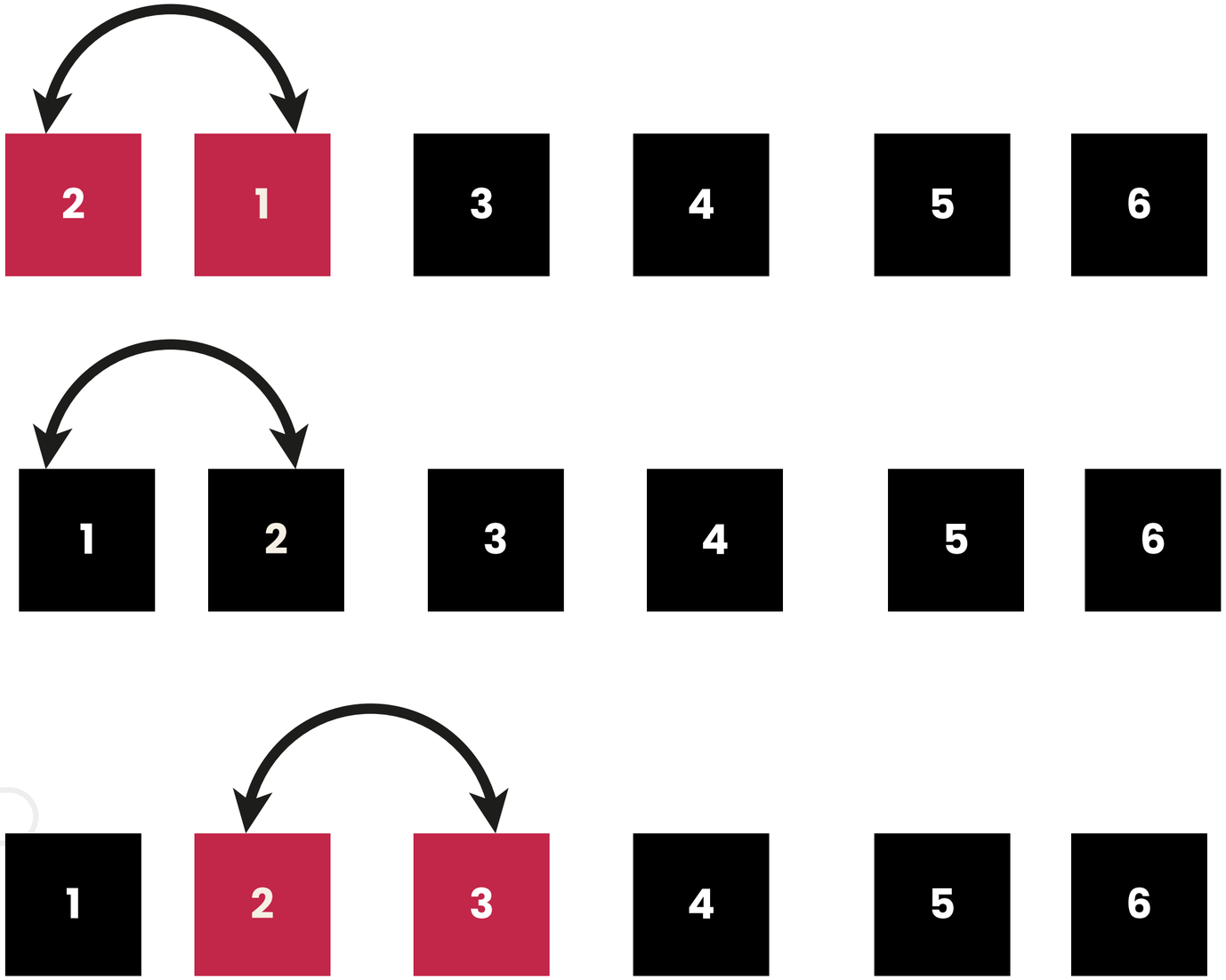

2
3
1
4
5
6

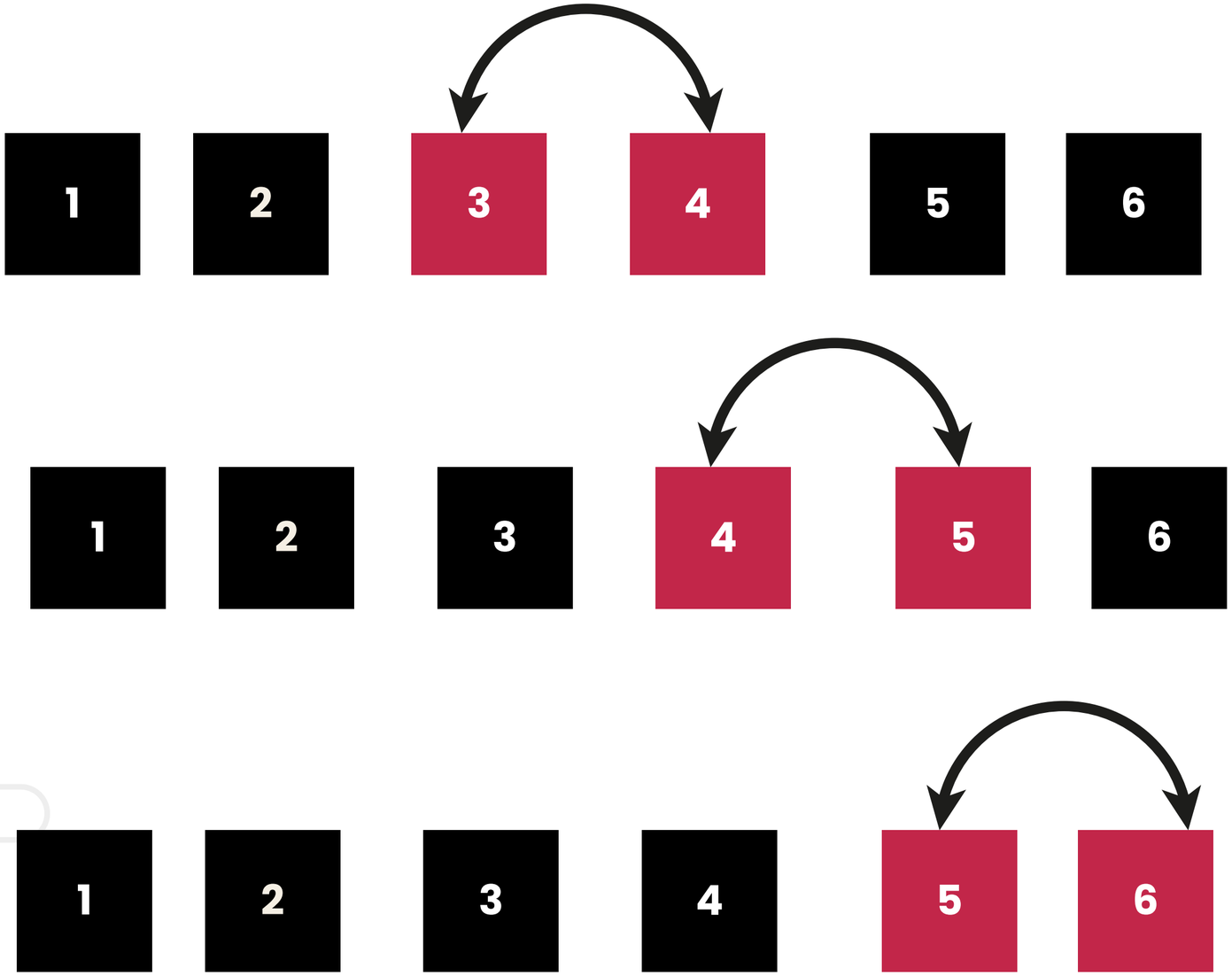

2
3
1
4
5
6


2
1
3
4
5
6


2
1
3
4
5
6
2
1
3
4
5
6
2
1
3
4
5
6

Fifth pass


2
1
3
4
5
6
1
2
3
4
5
6
1
2
3
4
5
6


1
2
3
4
5
6
1
2
3
4
5
6
1
2
3
4
5
6

TIME COMPLEXITY	
BEST	O(n)
AVG.	O(n^2)
WORST	O(n^2)

Insertion Sort

M Ahsan Naeem

Insertion sort algorithm

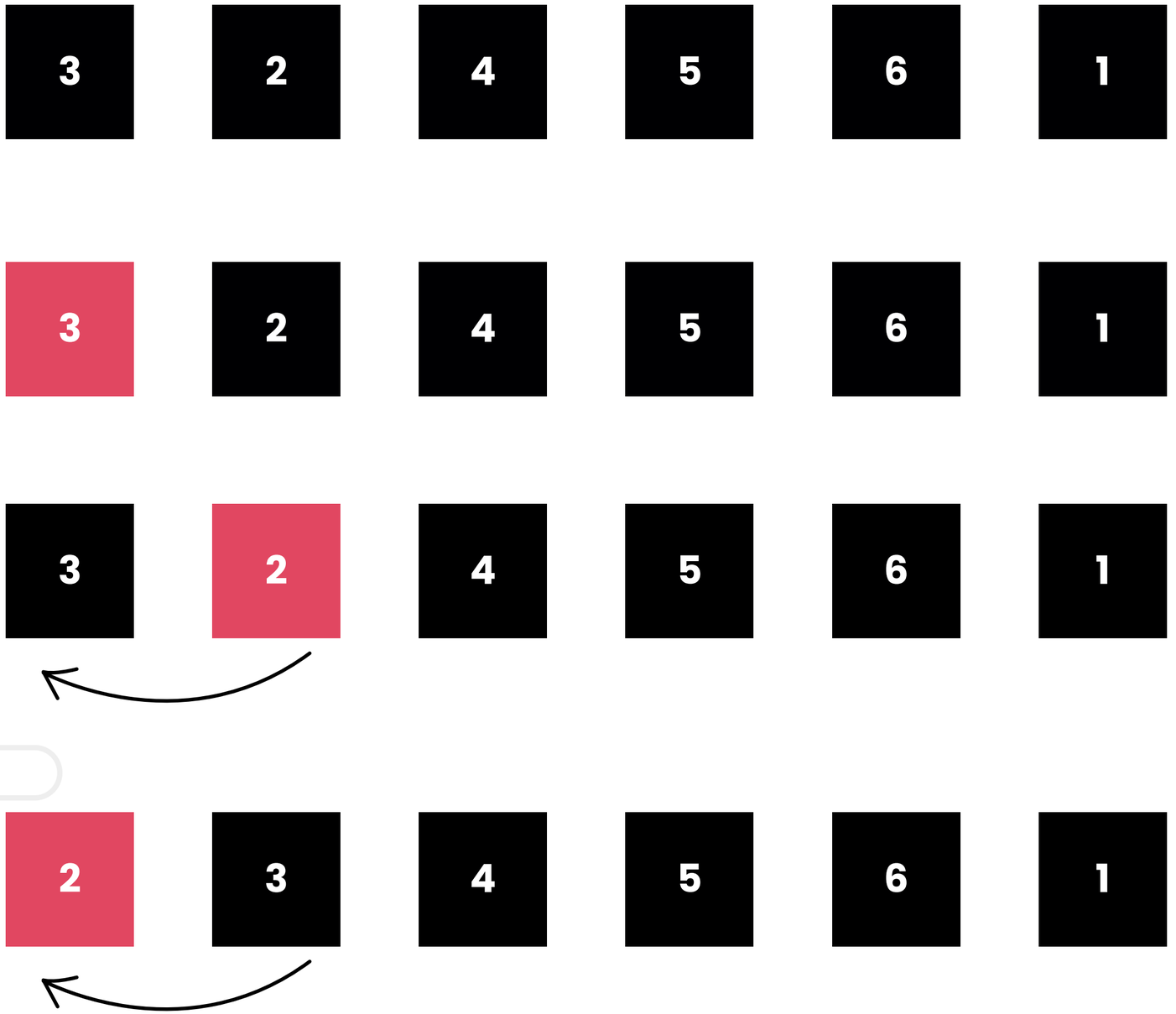
1. Start with an array of elements.
2. Iterate through the array from the second element to the last element.
3. For each element, compare it with the elements to its left.
4. If the element is smaller than the element to its left, shift the larger element to the right.
5. Repeat this process until the element is in its correct sorted position.
6. Move to the next element and repeat steps 3-5.
7. Continue iterating through the array until all elements are in their correct sorted positions.

```cpp
#include <iostream>
using namespace std;

void insert(int a[], int n){ /* function to sort an aay with insertion sort */
    int i, j, temp;
    for (i = 1; i < n; i++) {
        temp = a[i];
        j = i - 1;
        while(j>=0 && temp <= a[j])  /* Move the elements greater than temp to one position ahead from
            their current position*/
        {
            a[j+1] = a[j];
            j = j-1;
        }
        a[j+1] = temp;
    }
}
```

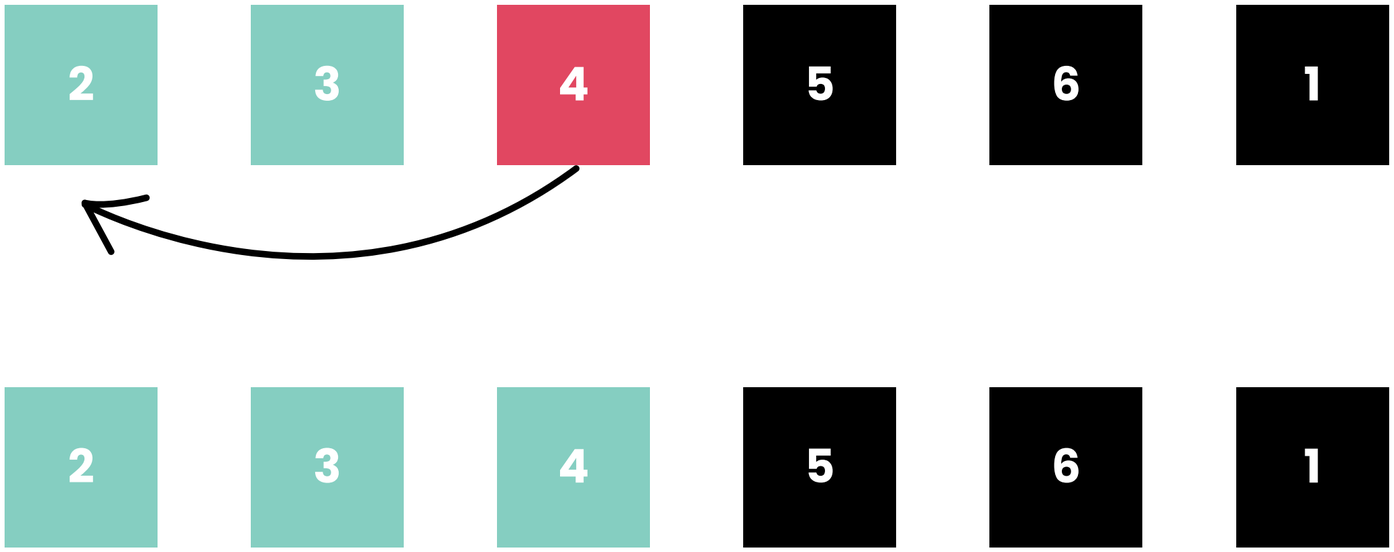
```cpp
int main()
{
    int a[] = { 3,2,4,5,6,1};
    int n = sizeof(a) / sizeof(a[0]);
    cout<<"Before sorting array elements are - "<<endl;
    printArr(a, n);
    insert(a, n);
    cout<<"\nAfter sorting array elements are - "<<endl;
    printArr(a, n);

    return 0;
}
```

```cpp
void printArr(int a[], int n){
    int i;
    for (i = 0; i < n; i++)
        cout << a[i] <<" ";
}
```

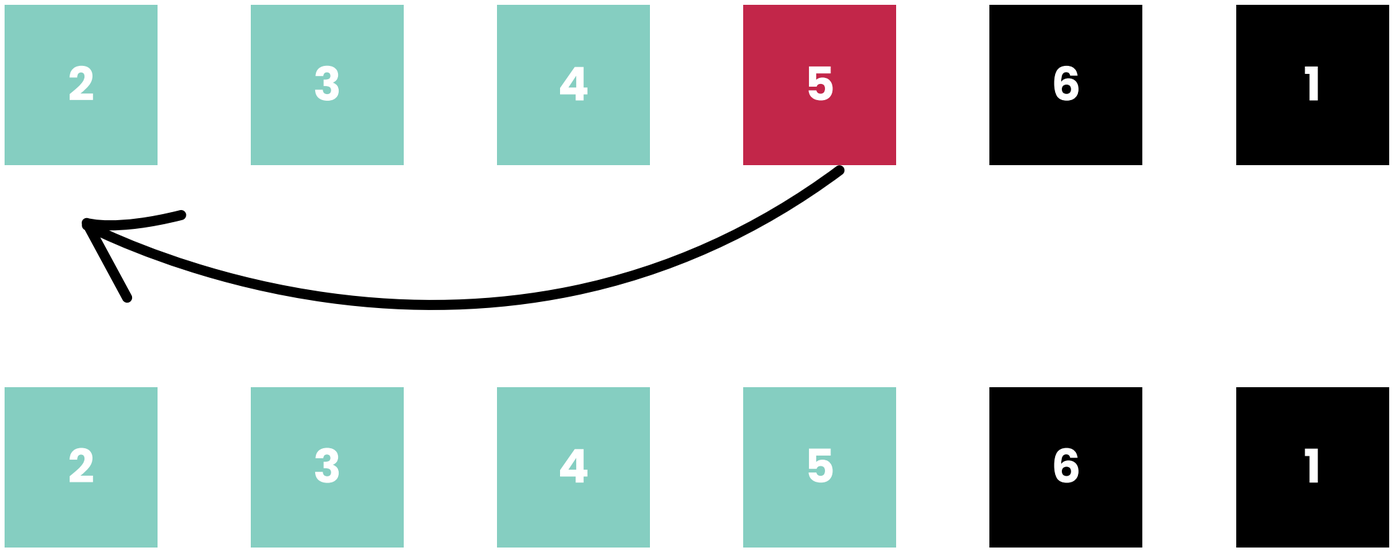
1st pass



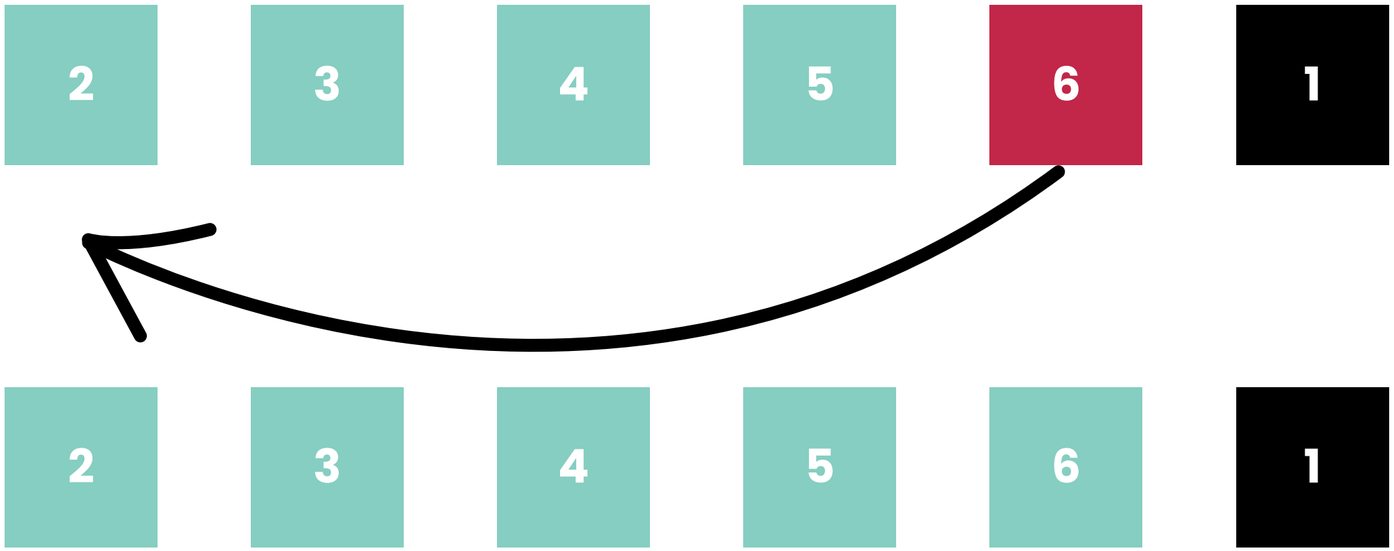
2nd pass



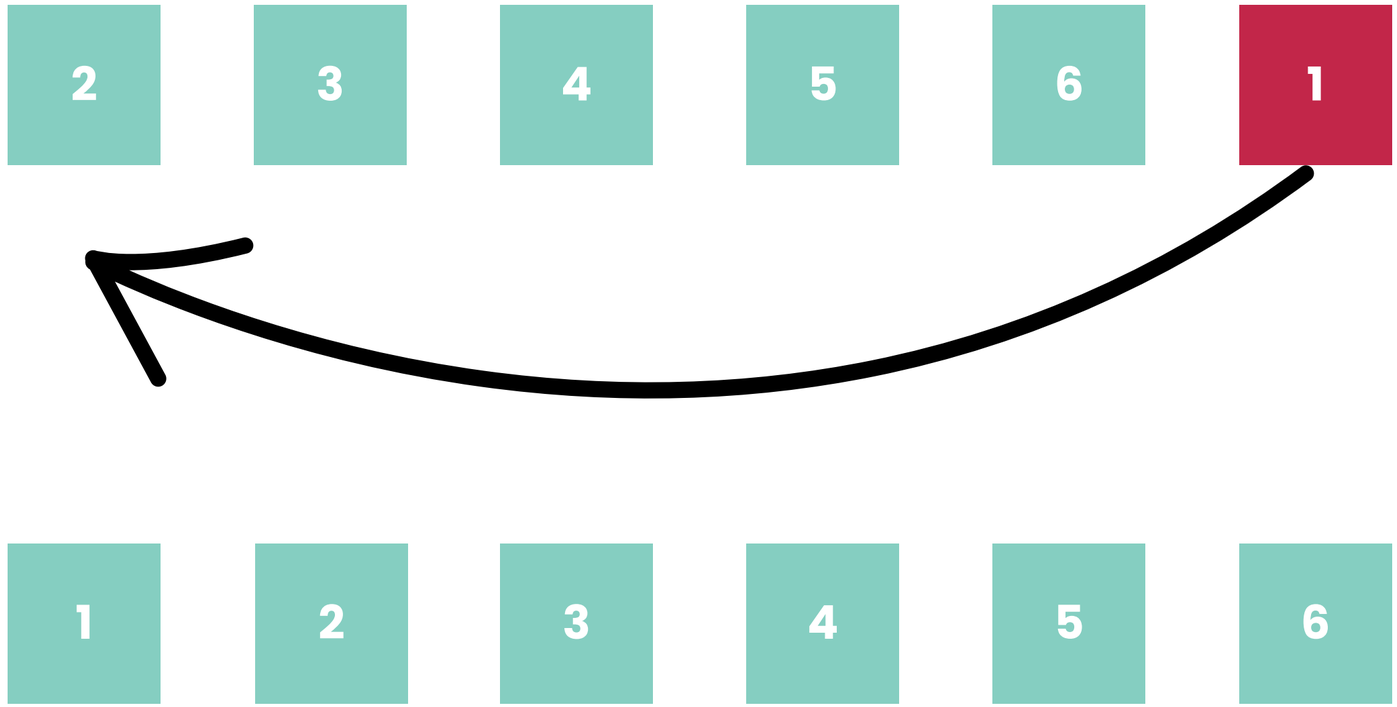
3rd pass



4th pass



5th pass



TIME COMPLEXITY	
BEST	O(n)
AVG.	O(n^2)
WORST	O(n^2)

Merge Sort

Khurram khalid

Merge sort algorithm

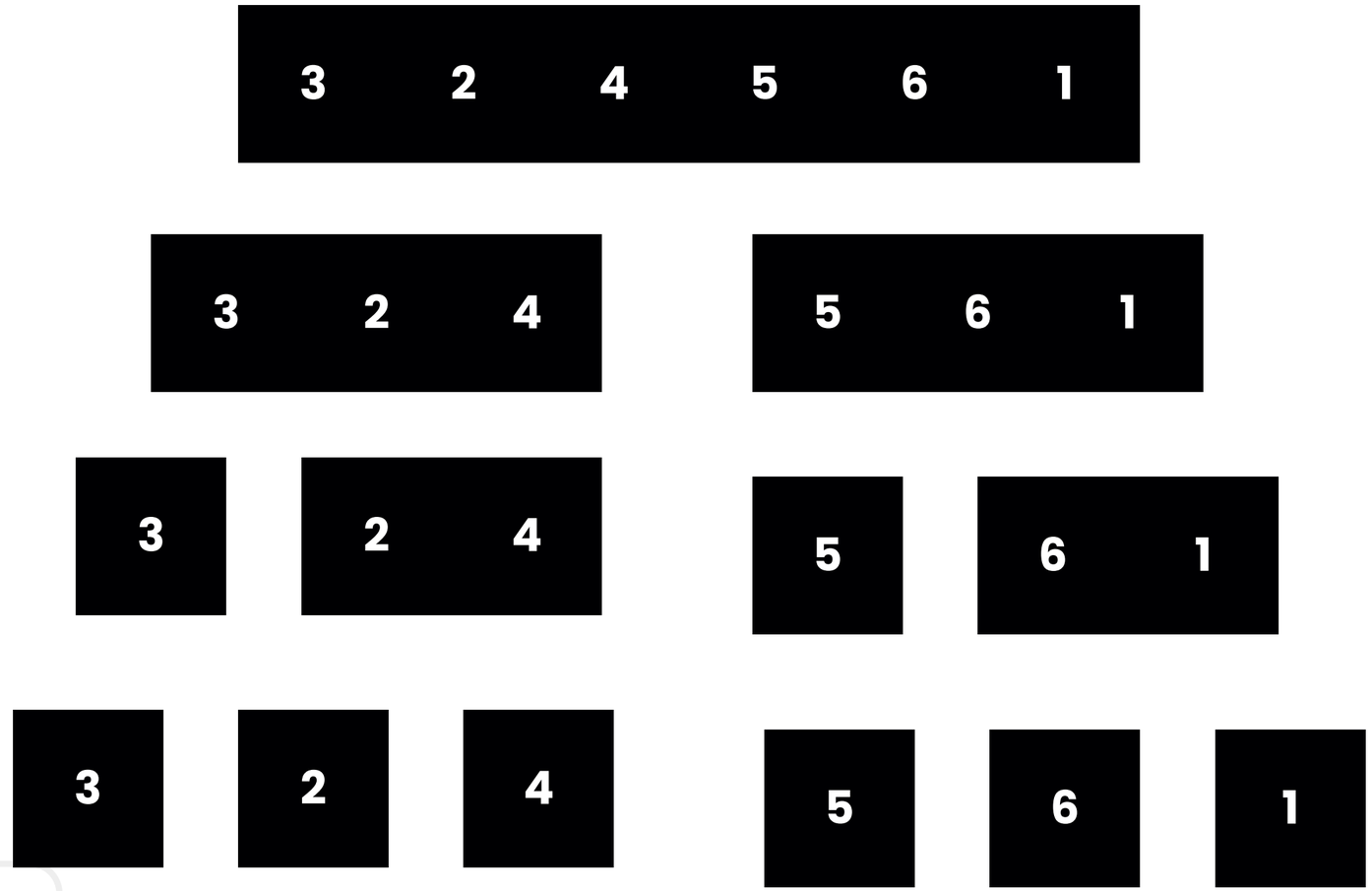
1. Divide the unsorted array into two halves.
2. Repeat the process recursively for each half until there is only one element left in each half.
3. Merge the two halves by comparing the elements and placing them in the correct order.
4. Continue merging the subarrays until the entire array is sorted.

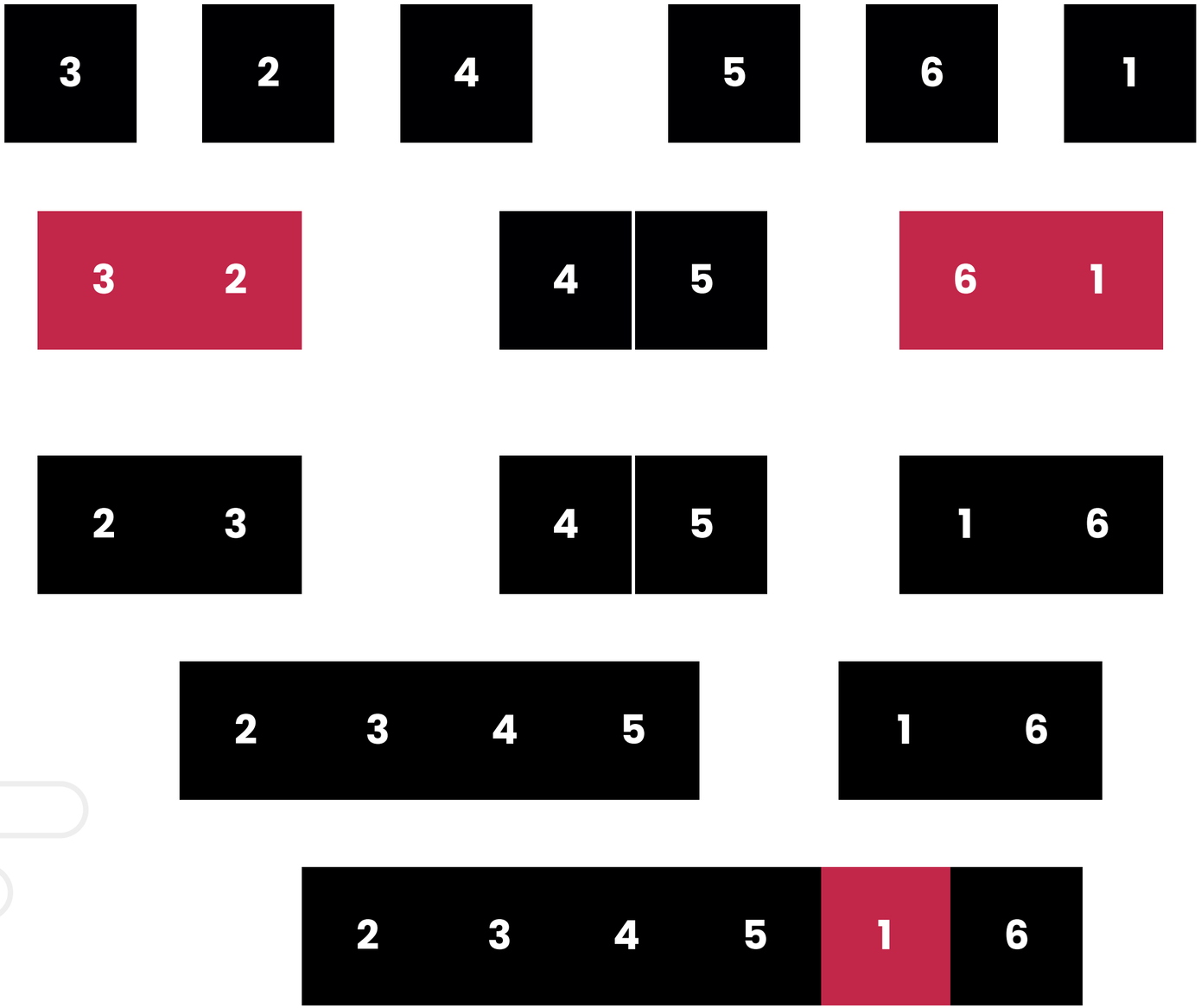
```
void merge(int A[], int mid, int low, int high)
{
   int i, j, k, B[100];
   i = low;
   j = mid + 1;
   k = low;

   while (i <= mid && j <= high)
   {
      if (A[i] < A[j])
      {
         B[k] = A[i];
         i++;
         k++;
      }
      else
      {
         B[k] = A[j];
         j++;
         k++;
      }
   }
   while (i <= mid)
   {
      B[k] = A[i];
      k++;
      i++;
   }
   while (j <= high)
   {
      B[k] = A[j];
      k++;
      j++;
   }
   for (int i = low; i <= high; i++)
   {
      A[i] = B[i];
   }
}
```

```
void mergeSort(int A[], int low, int high)
{
        int mid;
  if(low<high){
        mid = (low + high) /2;
        mergeSort(A, low, mid);
        mergeSort(A, mid+1, high);
        merge(A, mid, low, high);
        }
}
```

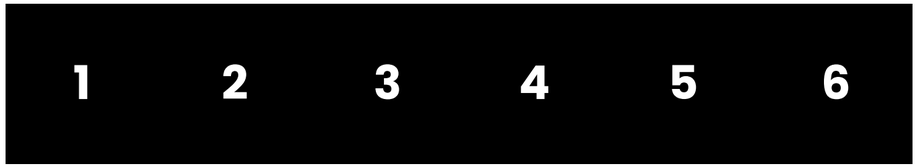
```
void printArray(int *A, int n){
        for (int i = 0; i < n; i++){
printf("%d ", A[i]);
}
printf("\n");
}
```

```
int main(){//

int A[] = {3,2,4,5,6,1};
int n = 6;
printArray(A, n);
mergeSort(A, 0, 6);
printArray(A, n);
return 0;
}
```


3 2 4 5 6 1
3 2 4
5 6 1
3
2 4
5
6 1
3
2
4
5
6
1


3 2 4 5 6 1
3 2 4 5 6 1
2 3 4 5 1 6
2 3 4 5 1 6
2 3 4 5 1 6


2 3 4 5 1 6


1 2 3 4 5 6

TIME COMPLEXITY	
BEST	O(nlog(n))
AVG.	O(nlog(n))
WORST	O(nlog(n))


Any Questions?


THANK YOU