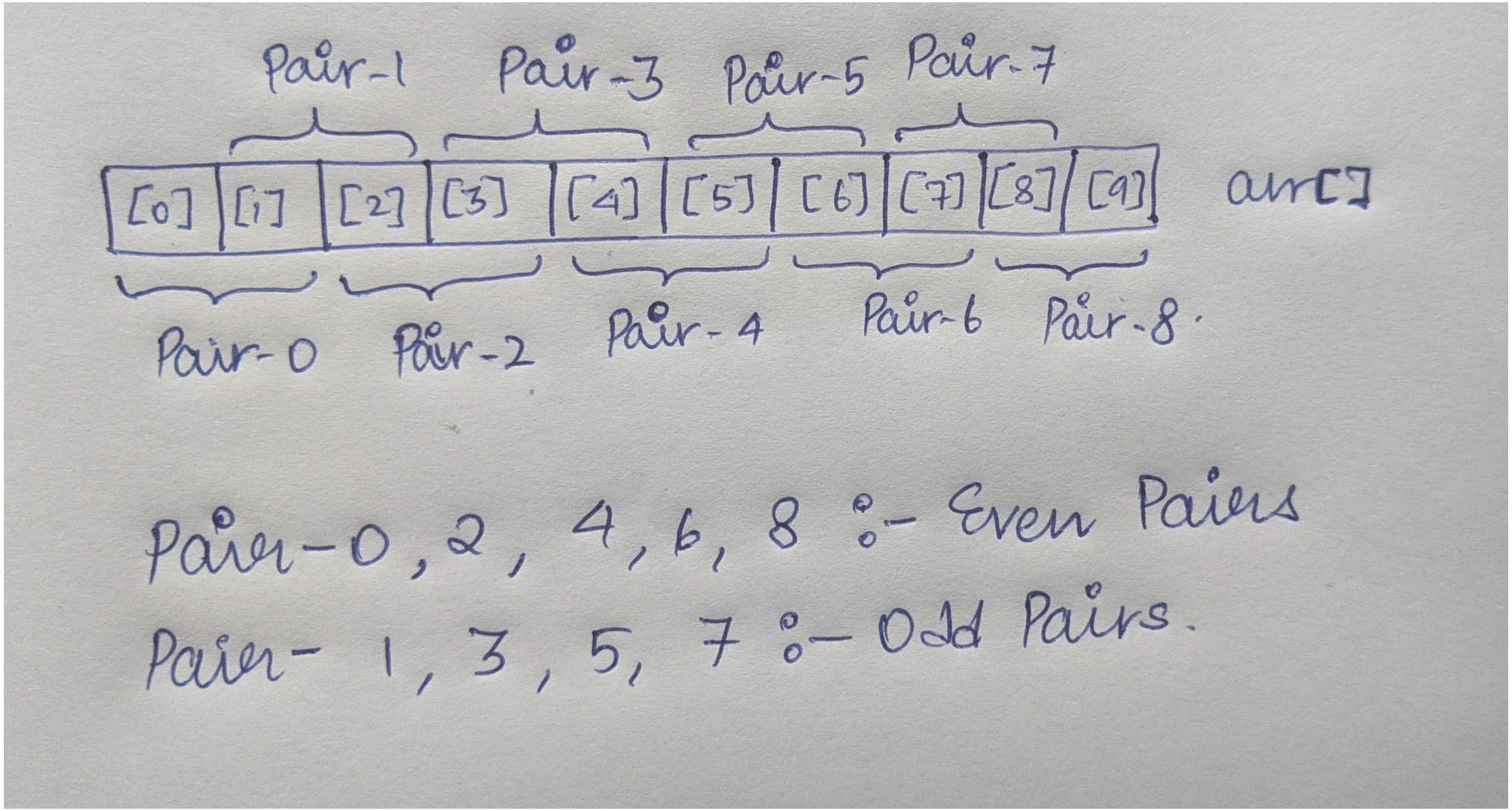
ODD-EVEN SORT

Algorithm

Explained with Example Array (Size = 10)

Even and Odd Pairs

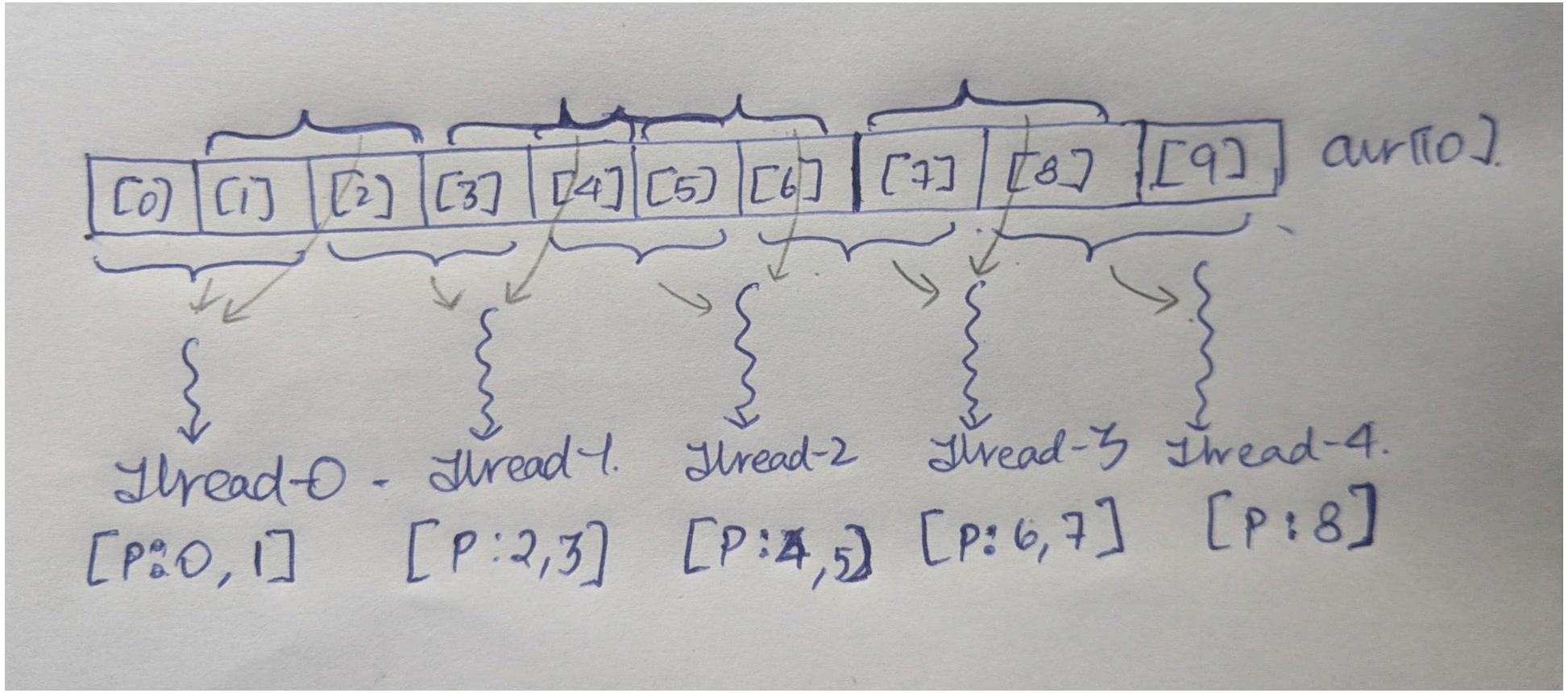


Identifying Even and Odd Pairs

- In Above example : We have Array of **10 elements from index 0 to 9**
- Mark and Identify **Pairs in the array** as shown in diagram
    - Index {0,1} : Pair-0
    - Index {1,2} : Pair-1

    and so on..
- Divide the pairs into **Even & Odd Pairs**

Assigning Threads To Different Pairs



Assigning Threads to Pairs

In above example, there are totally 10 pairs

- Thread 0 : Responsible for Pair-0, Pair-1
- Thread 1 : Responsible for Pair-2, Pair-3
- Thread 2 : Responsible for Pair-4, Pair-5
- Thread 3 : Responsible for Pair-6, Pair-7
- Thread 4 : Responsible for Pair-8, Pair-9. (Note: There is no pair-9 in our case here)

- We launch 5 threads, 1 thread for 1 pair.

Algorithm Working

- **Every Step : 5 parallel threads are launched**
- **Step 1 : All 5 threads sort their respective assigned Even-Pairs in Ascending Order Parallelly**
  - **Even Pairs : {0,2,4,6,8}**
  - **Example:** Thread-0 will sort its **assigned even-pair : pair-0**
    Thread-1 will sort its **assigned even-pair : pair-2**
    And so on for all other threads..
- **Step 2: All 5 threads sort their respective assigned Odd-Pairs in Ascending Order Parallelly**
  - **Odd Pairs: {1,3,5,7}**
  - **Example:** Thread-0 will sort its **assigned even-pair : pair-1**
    Thread-1 will sort its **assigned odd-pair : pair-3**
    An so on for all other threads….

- **The same process will be repeated 10 times (N = Array-Size = 10) :**

  5 Steps: Even-Pair Sort and 5 Steps : Odd-Pair Sort

So after 10 such steps, the array will be sorted.

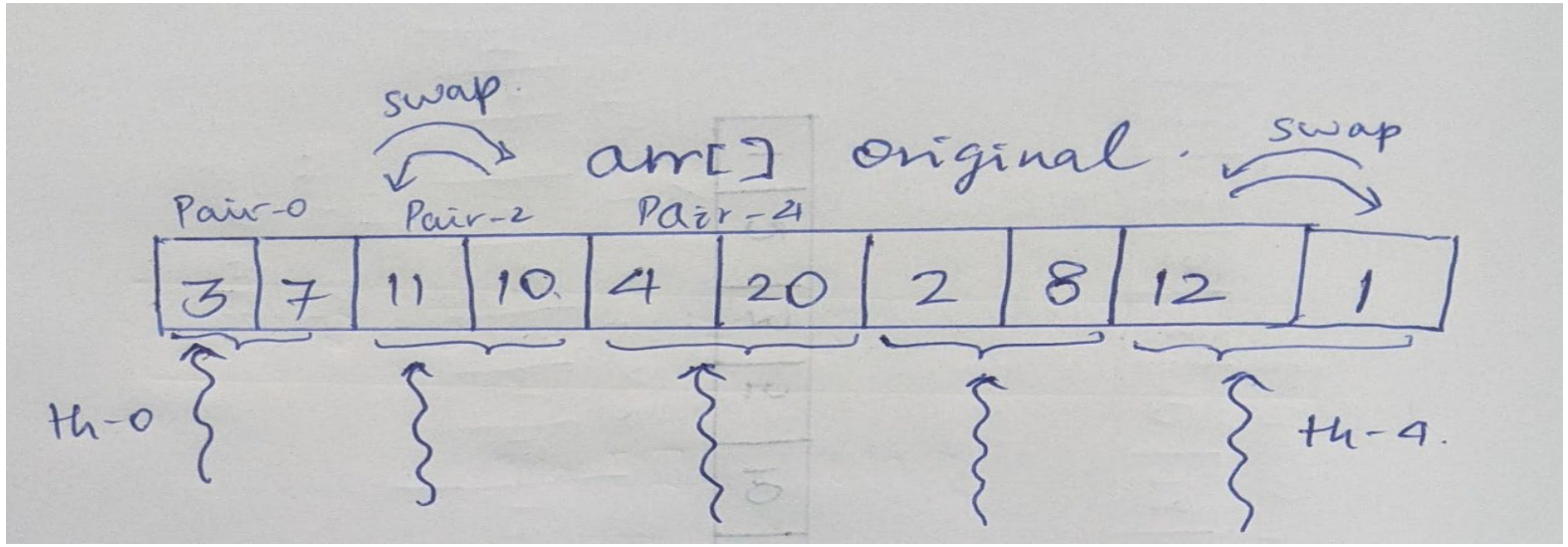
This is the odd-even sort algorithm.

ODD-EVEN ALGORITHM (ASCENDING SORT)

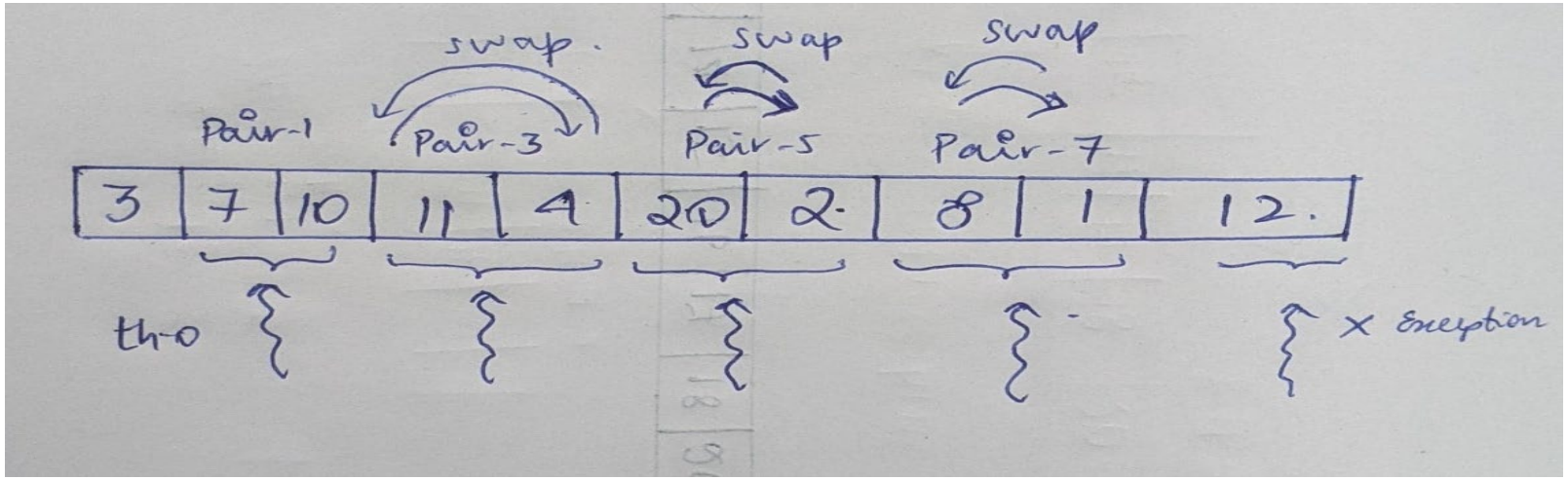
EXAMPLE RUN ON ARRAY

ARR : {3,7,11,10,4,20,2,8,12,1}

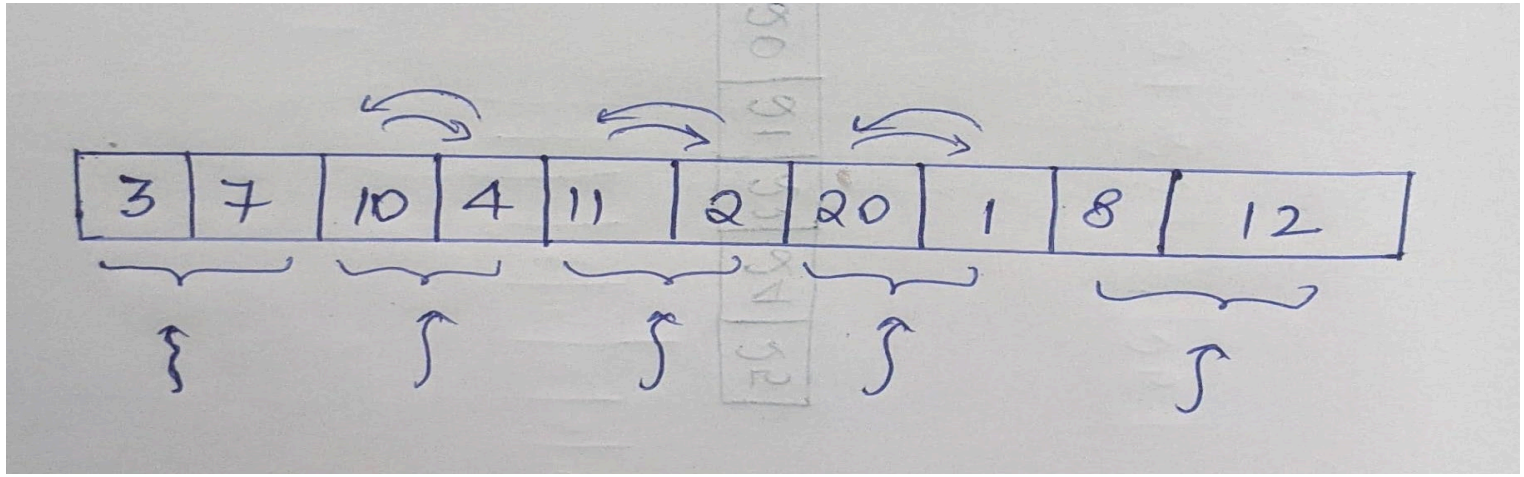
Iteration-0 (Even Pairs)



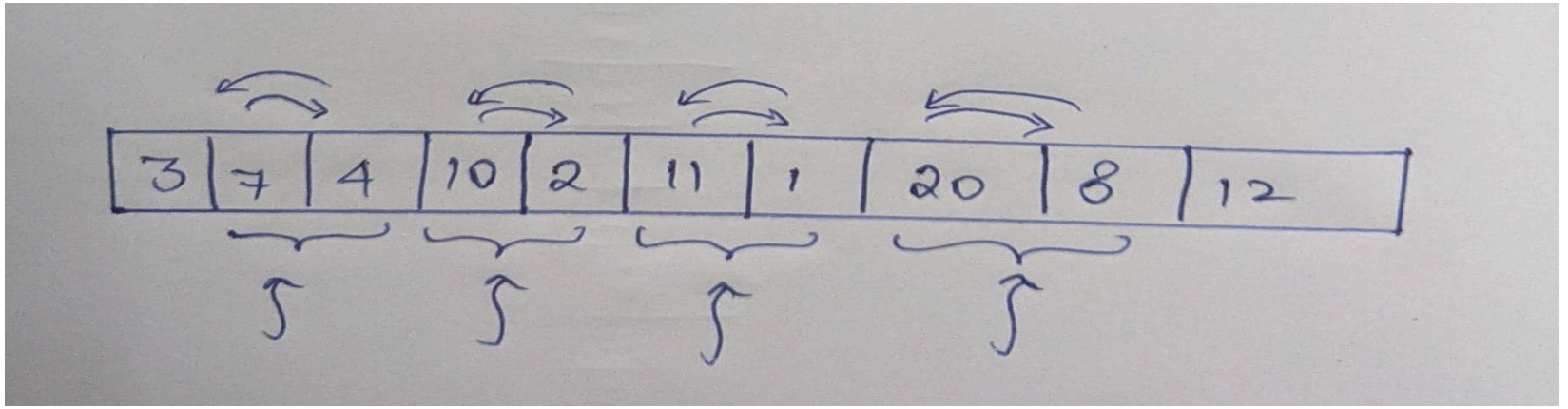
Iteration-1 (Odd Pairs)



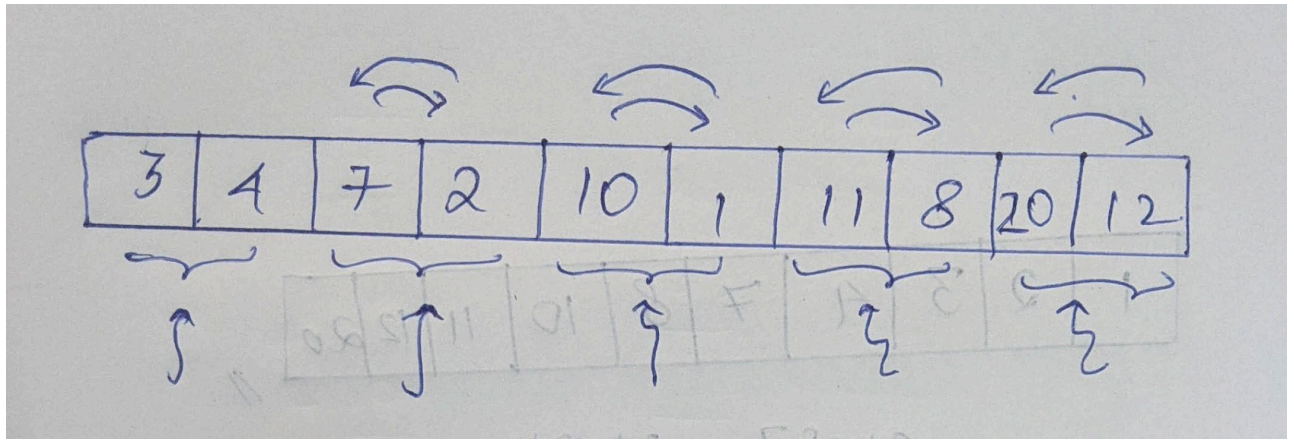
Iteration-2 (Even Pairs)



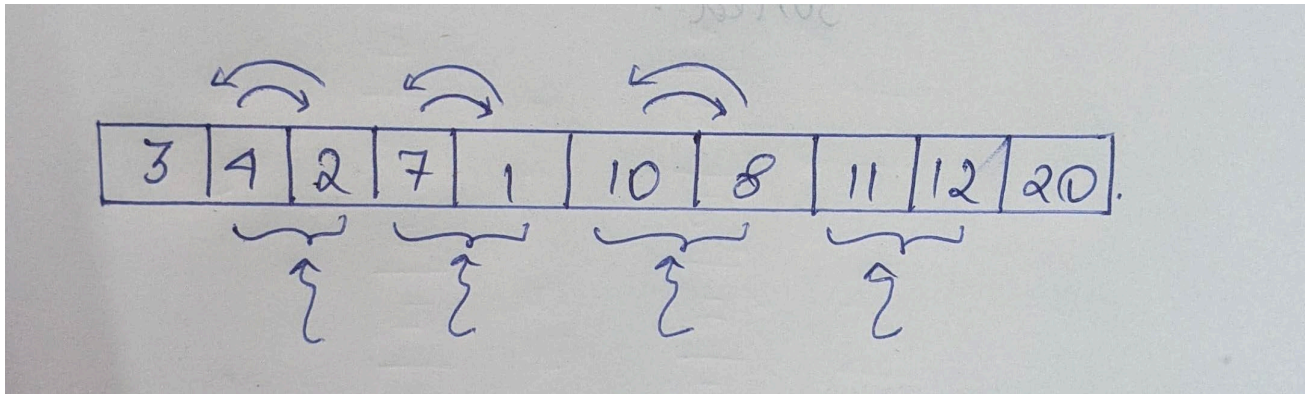
Iteration-3 (Odd Pairs)



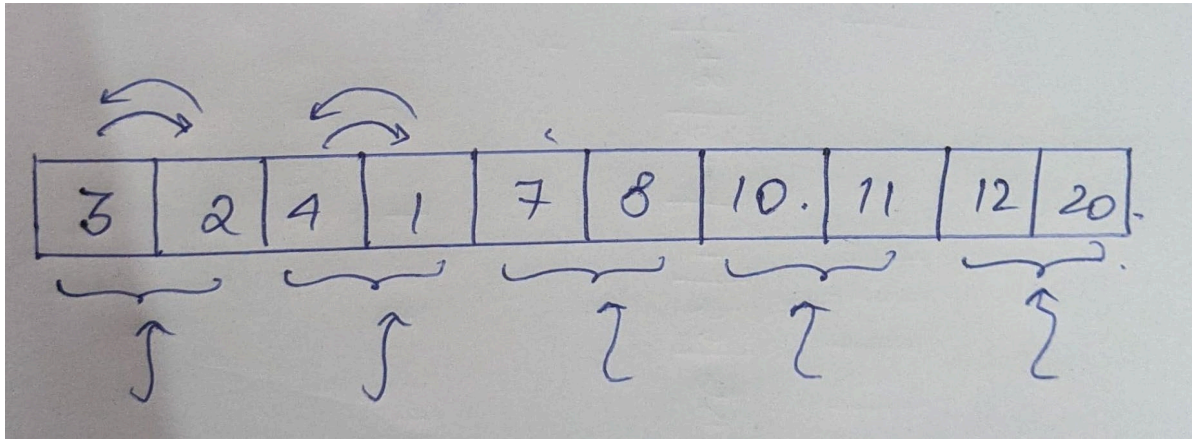
Iteration-4 (Even Pairs)



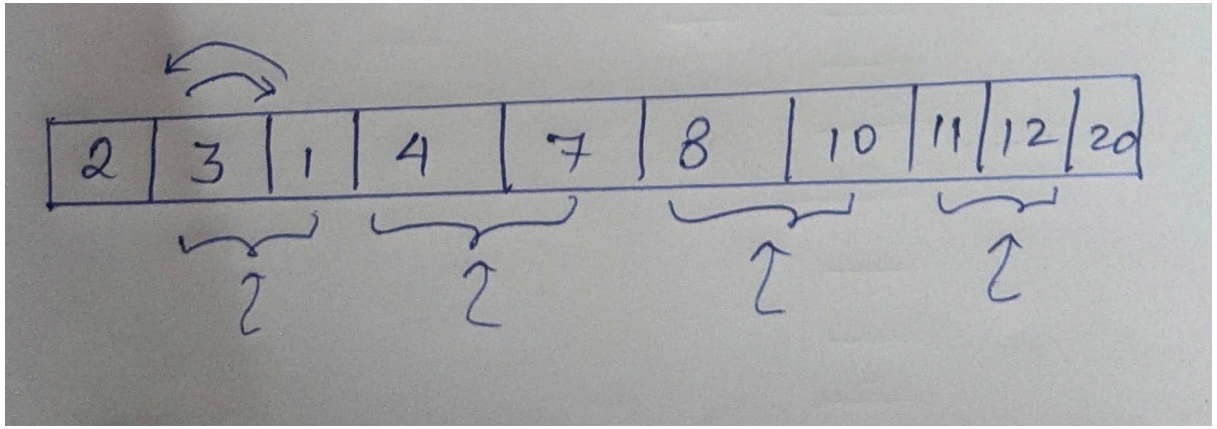
Iteration-5 (Odd Pairs)



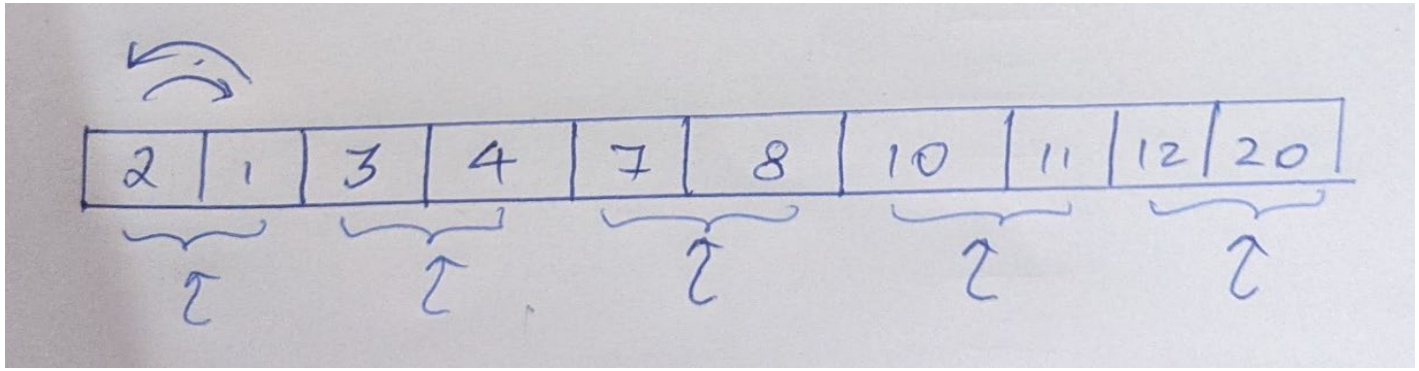
Iteration-6 (Even Pairs)



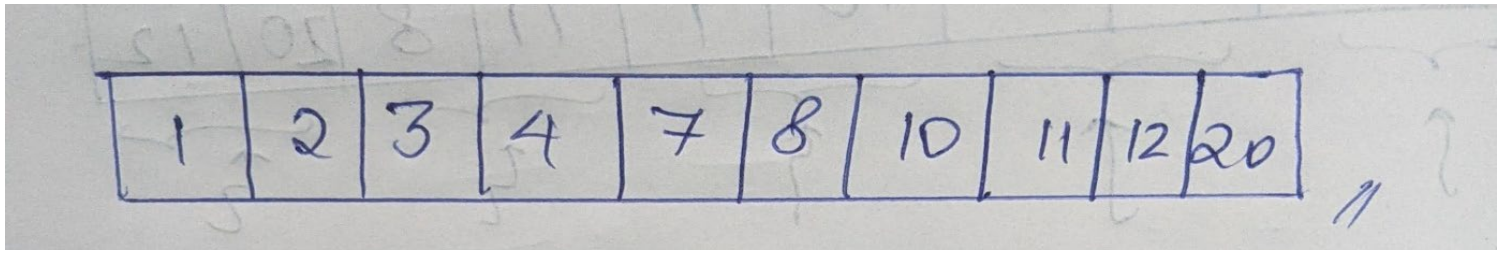
Iteration-7 (Odd Pairs)



Iteration-8 (Even Pairs)



Iteration-9 (Odd Pairs)



Array arr[] is now sorted

CUDA-CODE IMPLEMENTATION OF ODD-EVEN SORT

THE CPU-HOST CODE (main())

Create and Allocate h_arr[] (CPU Host) and d_arr[] (GPU Device)

```
int h_arr[] = {3, 7, 1, 10, 4, 20, 2, 8};

// ...


int n = sizeof(h_arr) / sizeof(h_arr[0]);     // n = size of array


//-------------------------------------------------------CREATE AND ALLOCATE d_n
int* d_n;
cudaMalloc((void**)&d_n, 1 * sizeof(int));
cudaMemcpy((void*)d_n, (void*)&n, 1 * sizeof(int), cudaMemcpyHostToDevice);

//-------------------------------------------------------------Create and Allocate d_arr[]
int* d_arr;
int size = sizeof(h_arr[0]) * n;
cudaMalloc((void**)&d_arr, size);   //Allocate memory for 5 int in d_arr  (Device global memory)
cudaMemcpy((void*)d_arr, (void*)h_arr, size, cudaMemcpyHostToDevice);   //Copy values from h_arr[] to d_arr[]
```

Make Kernel Call for n/2 threads

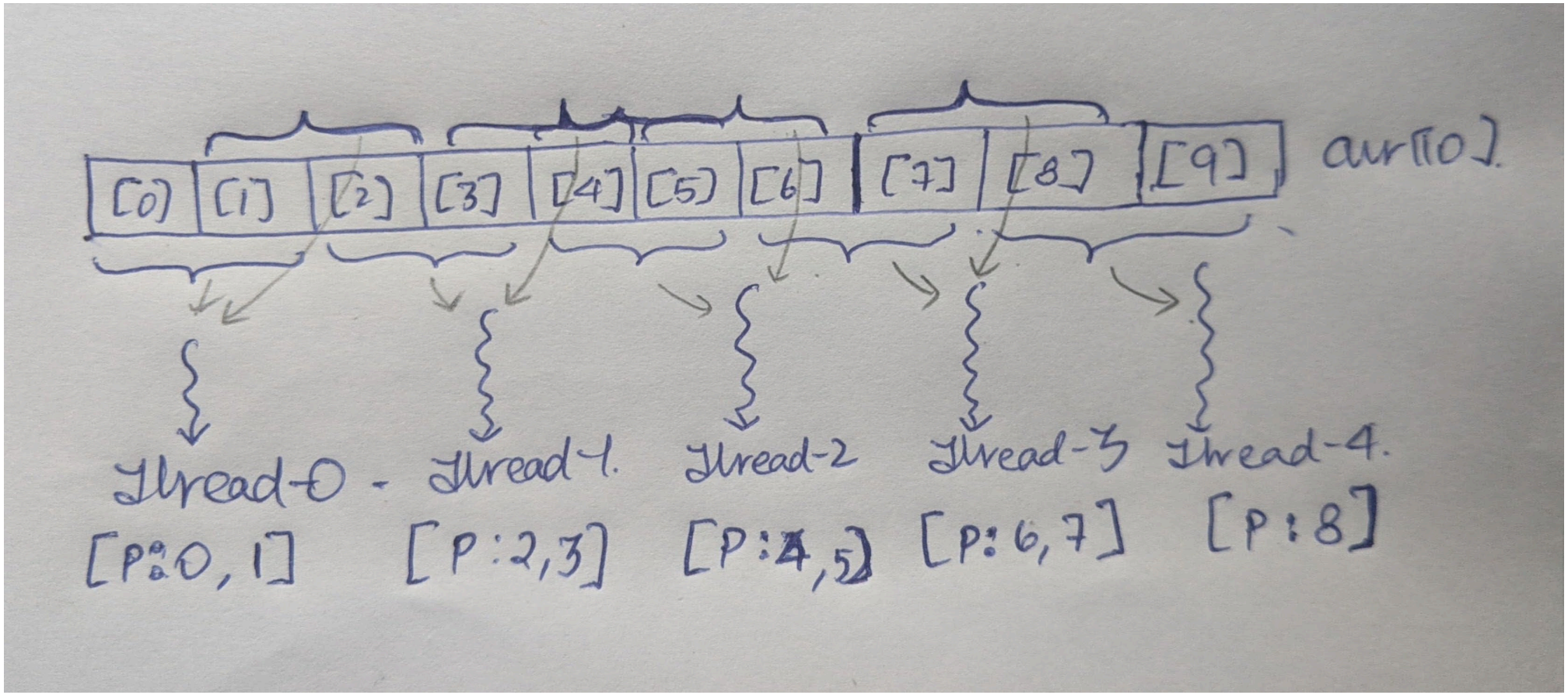
```
int k = n / 2;

/* ... */

odd_even <<< 1, k, n * sizeof(d_arr[0]) >>> (d_arr, d_n);    //The third kernel parameter is for the shared memory size in the thread

/* ... */

cudaMemcpy((void*)h_arr, (void*)d_arr, size, cudaMemcpyDeviceToHost);   //Copy values from d_arr[] to h_arr[]
```

KERNEL CODE

```
__global__ void odd_even(int* arr, int* arr_size)
{
    const int n = *arr_size;
    int idx = threadIdx.x;

    int f_even = idx * 2;
    int s_even = f_even + 1;

    int f_odd = idx * 2 + 1;
    int s_odd = f_odd + 1;

    //Copy all inputs array from global memory to shared memory
    extern __shared__ float sh_arr[];
    sh_arr[f_even] = arr[f_even];
    sh_arr[s_even] = arr[s_even];
    __syncthreads();
```

- **Calculate indexes of even pair and odd pair** that would be handled by the current thread
- **Copy all elements from input arr[] to shared memory array sh_arr[].** (By Copying the Even pair elements of all n/2 threads.)
- **Barrier** all threads to ensure that they are copied

Assigning Threads To Different Pairs



Calculating Even and Odd Pair Element Indexes of Each Thread

- **Kernel code will be run by all 5 threads parallelly**
- **(f_even , s_even) : Will hold the first index and second index of even pair of current thread**
  - *f_even =idx*2*
    s_even= f_even +1

  - **Example:** *For thread-0 (idx=0), Its even pair consists of elements at indexes (0,1) :*
    - **f_even = 0*2 = 0**
    - **s_even = 0+1= 1**

- **(f_odd , s_odd) : Will hold the first index and second index of odd pair of current thread**
  - *f_odd = idx *2+1*
    s_odd = f_odd +1

  - **Example:** *For Thread-0 (idx=0), Its odd-pair consists of elements at indexes {1, 2}.*
    - **f_odd = 0*2+1= 1**
    - **S_odd = 1+1 = 2**

This same process will be done for all threads to calculate their respective even and odd pair addresses parallelly

n Steps : In Each step 5 Parallel pairs are sorted

```
for (int i = 0; i < n/2; i++)
{

    //--------------------------Sort Even Pairs-----------------------------------------------------

    //If the pair is in ascending order already
    if (sh_arr[s_even] >= sh_arr[f_even])
    {
        //Ignore
    }


    //If the pair is NOT in ascending order already
    else
    {
        //Swap both of them
        int temp = sh_arr[s_even];
        sh_arr[s_even] = sh_arr[f_even];
        sh_arr[f_even] = temp;

    }


    __syncthreads();    //BARRIER
```

Sort Even Pairs: Code

- Each thread will sort its even pairs.
- Here, each thread checks if its even pair is sorted and if not, it sorts the pair
- “ ***if (sh_arr[s_even] >= sh_arr[f_even])*** “
  - If the second element in pair is greater than first element, we ignore . (**Because pair is already sorted**)
  - **If the pair is not in ascending order** : we **swap** both the elements of the pair so that they are in sorted order.

```
    //--------------------------Sort Odd Pairs-----------------------------------------------------

    if (s_odd >= n) continue;    //Case of final OVERFLOW odd pair

    //If the pair is in ascending order already
    if (sh_arr[s_odd] >= sh_arr[f_odd])
    {
        //Ignore
    }


    //If the pair is NOT in ascending order already
    else
    {
        //Swap both of them
        int temp = sh_arr[s_odd];
        sh_arr[s_odd] = sh_arr[f_odd];
        sh_arr[f_odd] = temp;

    }

    __syncthreads();  //BARRIER

}
```

Sort Odd Pairs: Code

- Code working is similar to that of Even Pair sort

STEP COMPLEXITY AND WORK COMPLEXITY

- Totally n/2 steps are performed for array of size-n
- In each step, n/2 work is done parallelly
- Hence:

Step Complexity : O(n)
Work Complexity : $O(n^2)$

END