PART A:

Generic Code:

```
class Array<E> {
    private final Object[] objectarray;
    public final int length;

    public Array(int length)    {
        objectarray = new Object [length];
        this.length = length;
    }

    void set(int i, long l) {
        objectarray[i] = l;
    }

    void set2(int i, String l) {
        objectarray[i] = l;
    }
    @Override
    public String toString() {
        return Arrays.toString(objectarray);
    }

}
```

Used in main class as:

FOR STRING

```
Array<String> str_arr = new Array<>(length: 3);

for(int i=0;i<3;i++){
    str_arr.set2(i,arr.get(i));
}

System.out.println(str_arr);
```

FOR INTEGER

```
Array<Integer> int_arr1= new Array<>(length: 3);
int_arr1.set(i: 0,cgpa4.size());
int_arr1.set(i: 1, (endTime1 - startTime1));
int_arr1.set(i: 2, (endTime2 - startTime2));

System.out.println(int_arr1+" in ms");
```

Encapsulation:

Private data:

```
private int maxThreadCount;
// private int minSize;
private ArrayList<Double> cg;
private int size;
```

Getters /setters:

```
public int getmaxThreadCount(){
    return this.maxThreadCount;
}

public ArrayList<Double> getcg(){
    return this.cg;
}

public int getSize(){
    return this.size;
}
```

I/O File:

```
try{
    BufferedWriter f4 = new BufferedWriter(new
    FileWriter(fileName: "output4.txt"));

    f4.write(Double.toString(cgpa4.get(i)));

    f4.close();
}
catch (IOException e){
    System.out.println(e);
}
```

Parallelization:

Used two threads parallelly to sort array

```
Multithreading left= new Multithreading(even,
maxThreadCount: 2, even.size());
Multithreading right= new Multithreading(odd,
maxThreadCount: 2, odd.size());
//System.out.println(3);
Thread t1= new Thread(left);
//System.out.println(4);
Thread t2= new Thread(right);
//System.out.println(5);
```

Time calculation:

Time is calculated in ms and ns depending on array size

```
    long startTime1= System.currentTimeMillis();

    //System.out.println(6);
    t1.start();
    //System.out.println(7);
    t2.start();
    //System.out.println(8);

  t1.join();
  //System.out.println(9);
  t2.join();
  // System.out.println(10);
  long endTime1 = System.currentTimeMillis();
```

Odd-even Sort:

```
while (!isSorted) {
    isSorted = true;
    Double temp = 0.0;

    // Perform Bubble sort on odd indexed element
    for (int i = 1; i <= cg.size() - 2; i = i + 2) {
        if (cg.get(i) > cg.get(i+1)) {
            temp = cg.get(i);
            cg.set(i,cg.get(i+1));
            // cg.get(i) = cg.get(i+1);
            cg.set(i+1, temp);
            // cg.get(i+1) = temp;
            isSorted = false;
        }
    }

    // Perform Bubble sort on even indexed element
    for (int i = 0; i <= cg.size() - 2; i = i + 2) {
        if (cg.get(i) > cg.get(i+1)) {
            temp = cg.get(i);
            cg.set(i,cg.get(i+1));
            // cg.get(i) = cg.get(i+1);
            cg.set(i+1, temp);
            // cg.get(i+1) = temp;
            isSorted = false;
        }
    }
}
```

```
//@reference https://www.softwaretestinghelp.com/java-generic-array/
//@reference https://www.geeksforgeeks.org/java-program-for-odd-even-sort-brick-sort/
```

RUN CODE COMMANDS:

When we run part A we will see that for different size of arrays of CGPA we are sorting them using multithread and normal method and those cgpas are generated randomly using Math.random()

The output would look something like this:

```
Time is in ns

[size of arr, multithread time, normal time]
[10000, 401061600, 924944100] in ns
[1000, 284300, 13850500] in ns
[100, 424200, 472100] in ns
[10, 253200, 7300] in ns
[1, 243800, 1000] in ns
```

And data (cgpas: raw and sorted) is being stored in txt files and we see major change in timings between multithread and normal method for 10000 size of array of cgpa.