Q1.1

We can use Mergesort

① sort the left half of array (recursively)

② sort the right half of array (recurrively)

③ Merge the solutions.

Because we use recursive idea to solve this question,

we can $T(n) = aT\left(\frac{n}{b}\right) + cn^k$

we can use Master Theorem solves this.

So we have $T(n) = 2T\left(\frac{n}{2}\right) + c \cdot n$

if we apply master Theorems, we'll see that our case is one where $a = b^k$, $2 = 2^1$,

Our complexity is $O(n \log n)$, and it has been proven that an array can't be sorted faster than $O(n \log n)$.

SO, it's lower bound $\Omega(n \log n)$.

```java
public static void mergeSort(int[] array, int left, int right) {
    if (right <= left) return;
    int mid = (left+right)/2;
    mergeSort(array, left, mid);
    mergeSort(array, mid+1, right);
    merge(array, left, mid, right);
}

```

```java
void merge(int[] array, int left, int mid, int right) {
   // calculating lengths
   int lengthLeft = mid - left + 1;
   int lengthRight = right - mid;
   int leftArray[] = new int [lengthLeft];
   int rightArray[] = new int [lengthRight];
   for (int i = 0; i < lengthLeft; i++)
       leftArray[i] = array[left+i];
   for (int i = 0; i < lengthRight; i++)
       rightArray[i] = array[mid+i+1];
   int leftIndex = 0;
   int rightIndex = 0;
   for (int i = left; i < right + 1; i++) {
       if (leftIndex < lengthLeft && rightIndex < lengthRight) {
           if (leftArray[leftIndex] < rightArray[rightIndex]) {
               array[i] = leftArray[leftIndex];
               leftIndex++;
           }
           else {
               array[i] = rightArray[rightIndex];
               rightIndex++;
           }
       }
       else if (leftIndex < lengthLeft) {
           array[i] = leftArray[leftIndex];
           leftIndex++;
       }
       else if (rightIndex < lengthRight) {
           array[i] = rightArray[rightIndex];
           rightIndex++;
       }
   }
}
```