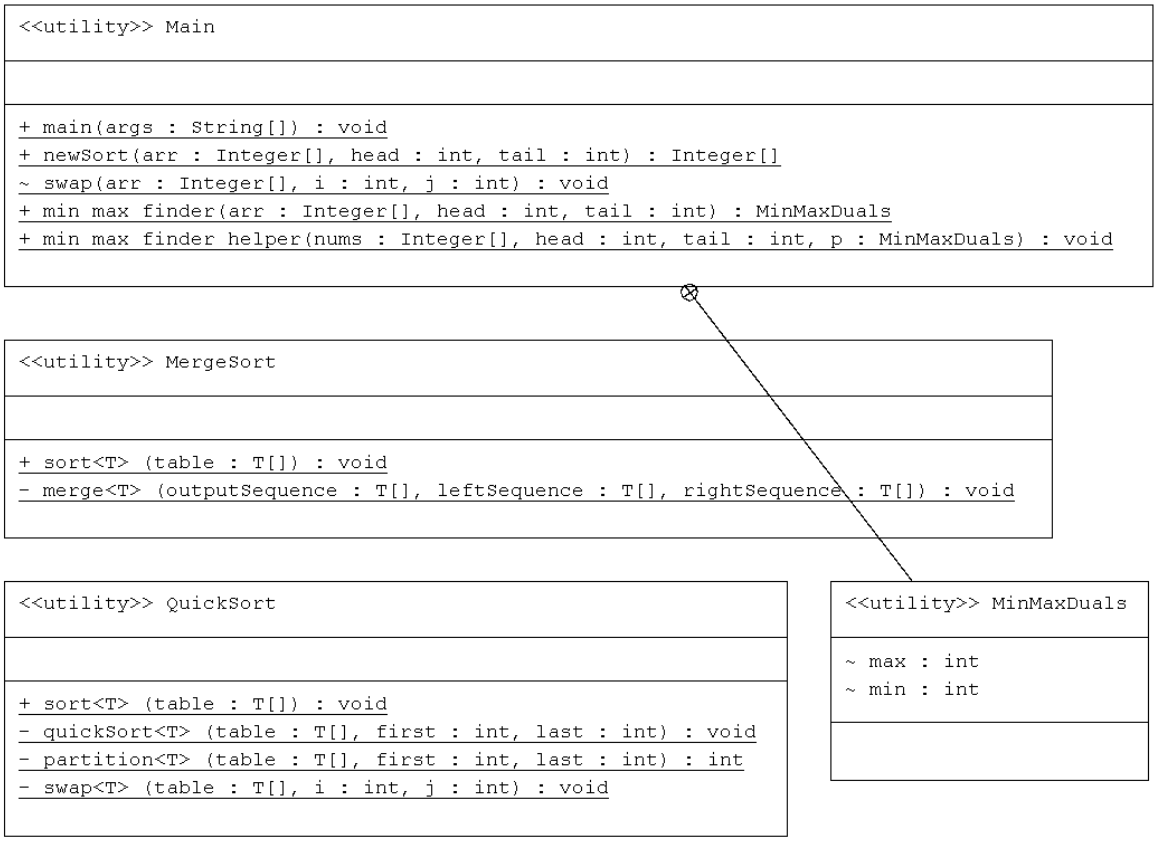


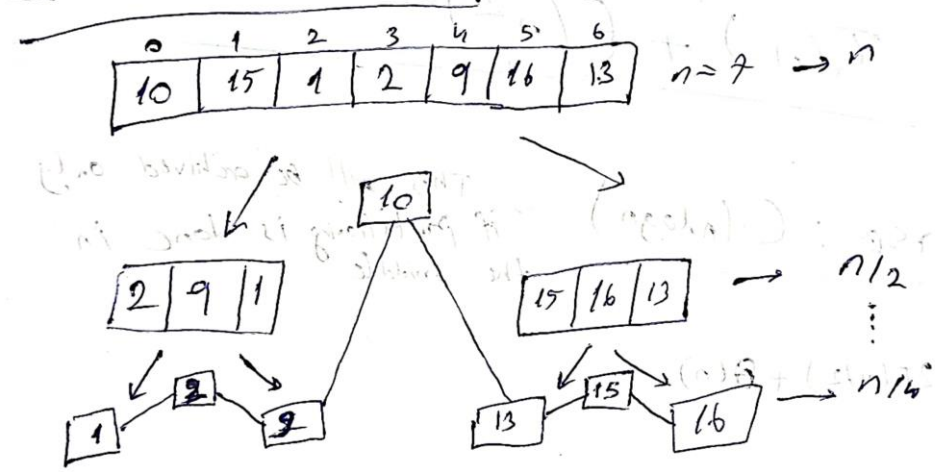

```
100luk:
mean merge sort: 0.044
mean quick sort: 0.029
mean new sort: 0.032

1000luk:
mean merge sort: 0.094
mean quick sort: 0.062
mean new sort: 0.473

10000luk:
mean merge sort: 0.5103333333333333
mean quick sort: 0.47833333333333333
mean new sort: 30.826
```

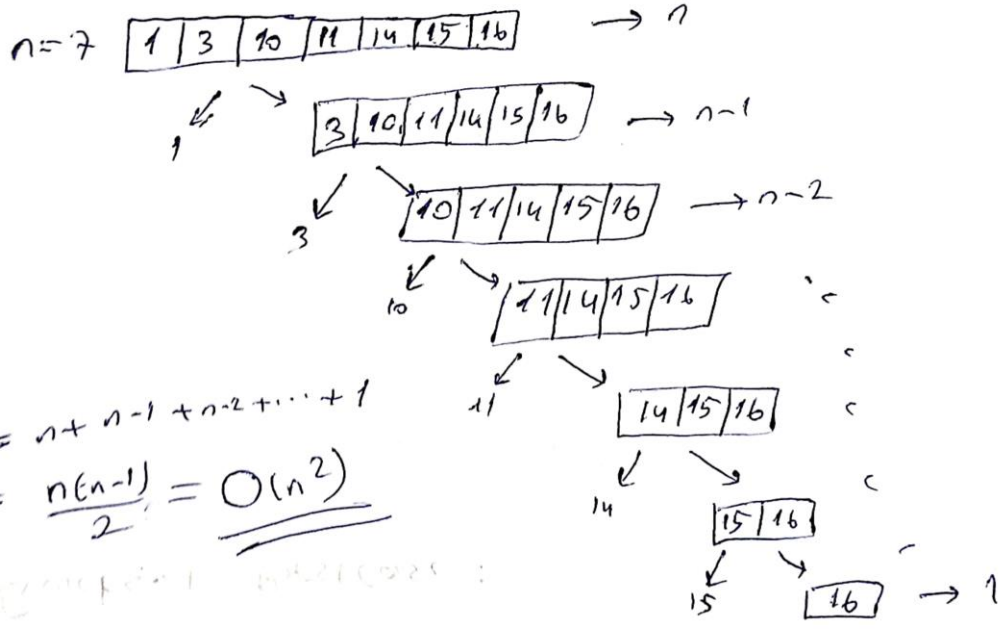
QuickSort:

QuickSort Bestcase



$$\frac{n}{2^k} = 1 \qquad n = 2^k \qquad = O(n \cdot \log n)$$

QuickSort worst case: Array is already sorted

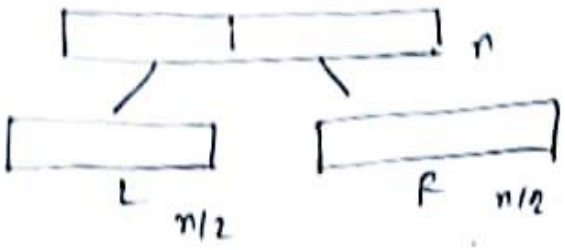


$$= n + n-1 + n-2 + \cdots + 1$$

$$= \frac{n(n-1)}{2} = O(n^2)$$

MergeSort:

Time Complexity for Merge Sort



$T(n) = 2T(n/2) + n$

$= 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$

$= 2^2 T\left(\frac{n}{4}\right) + n + n$

$= 2^2 T\left(\frac{n}{2^2}\right) + 2n$

$= 2^2\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n$

$= 2^3 T\left(\frac{n}{2^3}\right) + n + 2n$

$= 2^3 T\left(\frac{n}{2^3}\right) + 3n$

$= 2^k T\left(\frac{n}{2^k}\right) + nk$

$T(1) = O(1) = 1$

$\frac{n}{2^3} = 1$

$\frac{n}{2^k} = 1$

$n = 2^k$

$\log_2 n = k \log_2 2$

$\log_2 n = k$

$k = \log_2 n$

$= 2^{\log_2 n} T(1) + n \log_2 n$

$= 2^{\log_2 n} \cdot 1 + n \log_2 n$

$= n + n \log_2 n$

$= O(n \log_2 n)$ worst T

$O(n^2)$

newSort:

```java
 */
public static void min_max_finder_helper(Integer[] nums, int head, int tail, MinMaxDuals p) {
    // if the array contains only one element
    if (head == tail) {
        if (nums[p.max] < nums[head])
            p.max = head;
        if (nums[p.min] > nums[tail])
            p.min = tail;
        return;
    }
    // if the array contains only two elements
    if (tail - head == 1) {
        if (nums[head] < nums[tail]) {
            if (nums[p.min] > nums[head])
                p.min = head;

            if (nums[p.max] < nums[tail])
                p.max = tail;
        } else {
            if (nums[p.min] > nums[tail])
                p.min = tail;

            if (nums[p.max] < nums[head])
                p.max = head;
        }
        return;
    }
    //middle element
    int mid = (head + tail) / 2;
    //left subarray
    min_max_finder_helper(nums, head, mid - 1, p);
    //right subarray
    min_max_finder_helper(nums, mid, tail, p);
}
```

```java
     * */
    public static Integer[] newSort(Integer[] arr, int head, int tail) {
        if (head > tail)
            return arr;

        MinMaxDuals mm = new MinMaxDuals();
        mm = min_max_finder(arr, head, tail);

        swap(arr, head, mm.min);
        swap(arr, tail, mm.max);

        return newSort(arr, head + 1, tail - 1);

    }

    /**
```