KAUNAS UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATICS

COMPUTER DEPARTMENT

Lygiagretaus programavimo Inžinerinis projektas

Darbą atliko:

IFF 6/8 grupės studentas
Tadas Laurinaitis

Darbą vertino:

Doc. Romas Marcinkevičius

Darbo užduoties analizė ir sprendimo metodas

Darbo užduotis: Parallel Quicksort algoritmo greičio tyrimas lyginant su sequencial Quicksort algoritmu naudojant skirtingus duomenų ir gijų kiekius.

Programinė darbo dalis buvo atlikta su Java programavimo kalba. Rikiuotų duomenų tipas – integer, kiekvienas skaičius buvo

Quicksort algoritmas yra efektyvus rikiavimo algoritmas, paremtas rikiuojamų dalykų lyginimu. Efektyviai implementavus, jis gali būti 2-3 kartus greitesnis negu pagrindiniai jo priešininkai – merge sort ir heapsort.

Visu pirma susiradau Quicksort veikimo principus ir pseudo kodą. Po to ieškojau kuom skiriasi lygiagrečiai veikiantis Quicksort nuo nuosekliai veikiančio, bei kaip iš nuosekliai veikiančio algoritmo padaryti lygiagrečiai veikiantį. Susiradęs reikiamą informaciją, bei apytiksliai žinodamas ką reikia padaryti susikūriau naują Java projektą. Tada susikūriau metodą kuris sudarytų masyvą užpildytą atsitiktiniais skaičiais bei metodą kuris rastų trijų skaičių medijaną. Po to padariau nuosekliai veikiantį Quicksort algoritmą, kad turėčiau su kuo lyginti lygiagrečiai veikiantį algoritmą. Po to padariau lygiagrečiai veikiantį Quicksort algoritmą, kurio nemažą dalį kodo radau internete. Galiausiai parašęs visas likusias programos dalis, lyginau gautus programus rezultatus naudojant įvairius duomenų kiekius ir viską surašiau į lentelę. Po to pagal šią lentelę sudariau grafikus.

Programos dalių tekstai ir jų trumpi aprašymai

```
static class MyThread extends Thread {

    private int[] array;
    private int fromIndex;
    private int toIndex;
    private int threadCount;

    public MyThread(int[] array, int fromIndex, int toIndex, int threadCount) {
        this.array = array;
        this.fromIndex = fromIndex;
        this.toIndex = toIndex;
        this.threadCount = threadCount;
    }
    @Override
    public void run() {
        parallelQS(array, fromIndex, toIndex, threadCount);
    }
    public int[] getArray() {
        return array;
    }
    public int getFromIndex() {
        return fromIndex;
    }
    public int getToIndex() {
        return toIndex;
    }
    public int getThreadCount() {
        return threadCount;
    }
}
```

Pav. Nr. 1 – Gijos klasė, kurią paleidus bus vykdomas parallel Quicksort algortimas ant priskirtos masyvo dalies

```java
//Sukuria masyva ir uzpildo ji atsistiktiniais skaiciais
public static int[] makeArray(int count, int dataRange) {
    int[] array = new int[count];
    Random rng = new Random();
    for(int i = 0; i < count; i++) {
        int temp = rng.nextInt(dataRange);
        array[i] = temp;
    }
    return array;
}
//Sukeicia du masyvo elementus vietomis
public static void swap(int[] array, int i, int j) {
    int temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
//Randa mediana pivot'ui
public static int median(int a, int b, int c) {
    int med = Math.max(Math.min(a,b), Math.min(Math.max(a,b),c));
    return med;
}
```

Pav. Nr. 2 – Pagalbiniai metodai naudoti sukurti masyvą, apkeisti masyvo elemtus vietomis ir apskaičiuoti Quicksort algoritmo pivot'ą

```java
//QuickSort algoritmas veikiantis ant pasirinkto skaiciaus giju (threadCount)
public static void parallelQS(int[] array, int fromIndex, int toIndex, int threadCount) {
    if (threadCount <= 1) {
        quickSort(array, fromIndex, toIndex);
        return;
    }
    int rangeLength = toIndex - fromIndex;
    int distance = rangeLength / 4;

    int a = array[fromIndex + distance];
    int b = array[fromIndex + (rangeLength >>> 1)];
    int c = array[toIndex - distance];

    int pivot = median(a, b, c);
    int leftPartitionLength = 0;
    int rightPartitionLength = 0;
    int index = fromIndex;

    while (index < toIndex - rightPartitionLength) {
        int current = array[index];
        if (current > pivot) {
            rightPartitionLength++;
            swap(array, toIndex - rightPartitionLength, index);
        } else if (current < pivot) {
            swap(array, fromIndex + leftPartitionLength, index);
            index++;
            leftPartitionLength++;
        } else {
            index++;
        }
    }
    MyThread thread1 = new MyThread(array, fromIndex, fromIndex + leftPartitionLength, threadCount / 2);
    MyThread thread2 = new MyThread(array, toIndex - rightPartitionLength, toIndex,threadCount - threadCount / 2);
    thread1.start();
    thread2.start();
    try {
        thread1.join();
        thread2.join();
    } catch (InterruptedException ex) {
        System.out.println("Somethings wingwong in the thved area tfir");
    }
}
```

Pav. Nr. 3 – Lygiagretus Quicksort algoritmas, veikiantis pagal nustatytą gijų skaičių

```java
//paprastas QuickSort algoritmas veikiantis nuosekliai
public static void quickSort(int[] array, int fromIndex, int toIndex) {
    while (true) {
        int rangeLength = toIndex - fromIndex;
        int distance = rangeLength / 4;
        if (rangeLength < 2) {
            return;
        }
        int a = array[fromIndex + distance];
        int b = array[fromIndex + (rangeLength >>> 1)];
        int c = array[toIndex - distance-1];

        int pivot = median(a, b, c);
        int leftPartitionLength = 0;
        int rightPartitionLength = 0;
        int index = fromIndex;

        while (index < toIndex - rightPartitionLength) {
            int current = array[index];

            if (current > pivot) {
                rightPartitionLength++;
                swap(array, toIndex - rightPartitionLength, index);
            } else if (current < pivot) {
                swap(array, fromIndex + leftPartitionLength, index);
                index++;
                leftPartitionLength++;
            } else {
                index++;
            }
        }
        if (leftPartitionLength < rightPartitionLength) {
            quickSort(array, fromIndex, fromIndex + leftPartitionLength);
            fromIndex = toIndex - rightPartitionLength;
        } else {
            quickSort(array, toIndex - rightPartitionLength, toIndex);
            toIndex = fromIndex + leftPartitionLength;
        }
    }
}
```

Pav. Nr. 4 – Nuosekliai veikiantis Quicksort algoritmas

```java
public static void main(String[] args) {
    int dataCount = (int) (10 * Math.pow(10, 6));
    int dataRange = 100000;
    int[] array = makeArray(dataCount, dataRange);
    int[] array2 = array;

    long startTime1 = System.nanoTime();
    //quickSort(array, 0, array.length);
    long endTime1 = System.nanoTime();
    long time1 = endTime1-startTime1;
    //System.out.println("Sequential QuickSort took: " +time1/1000000 +"ms to sort: " +dataCount);

    startTime1 = System.nanoTime();
    parallelQS(array2, 0, array.length, 2);
    endTime1 = System.nanoTime();
    long time2 = endTime1-startTime1;
    System.out.println("Parallel QuickSort took: " +time2/1000000 +"ms to sort: " +dataCount);

    if(Arrays.equals(array, array2)) {
        System.out.println("Arrays are equal");
    }
}
```

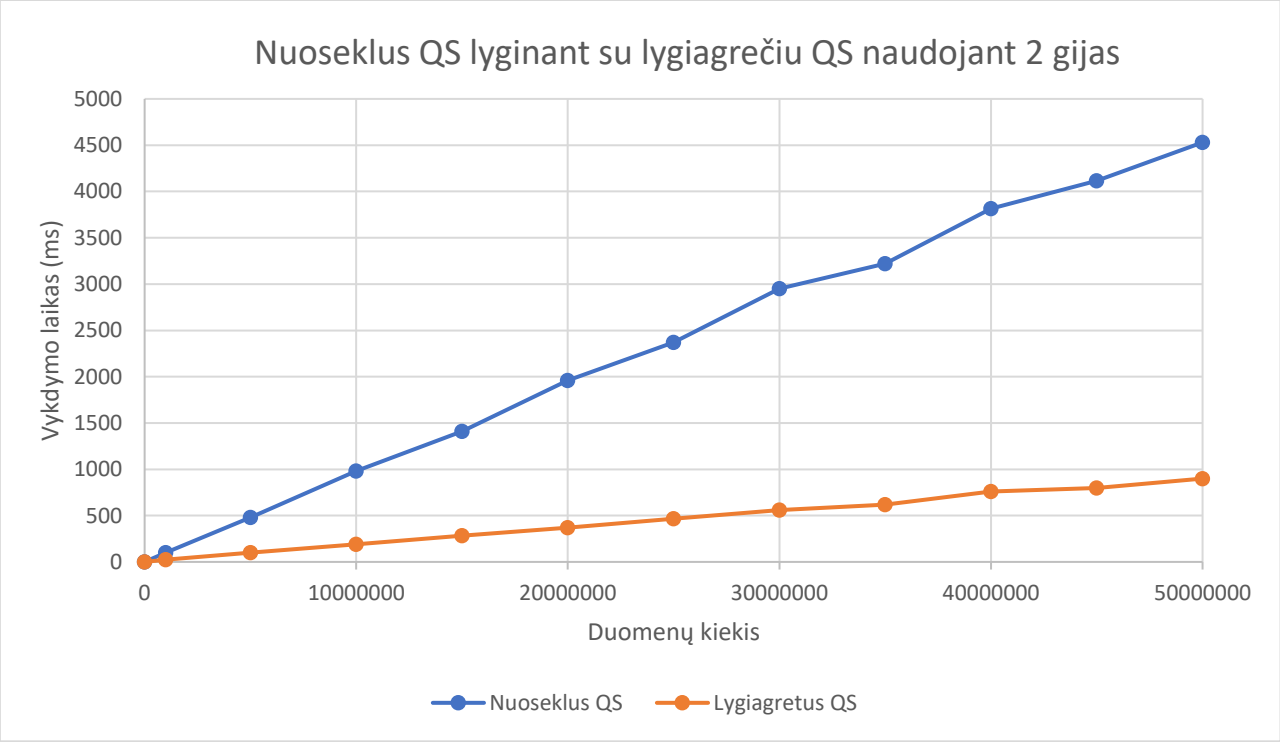
Pav. Nr. 5 – main metodas kuriame atliekami visi veiksmai

Vykdymo laiko kitimo tyrimas

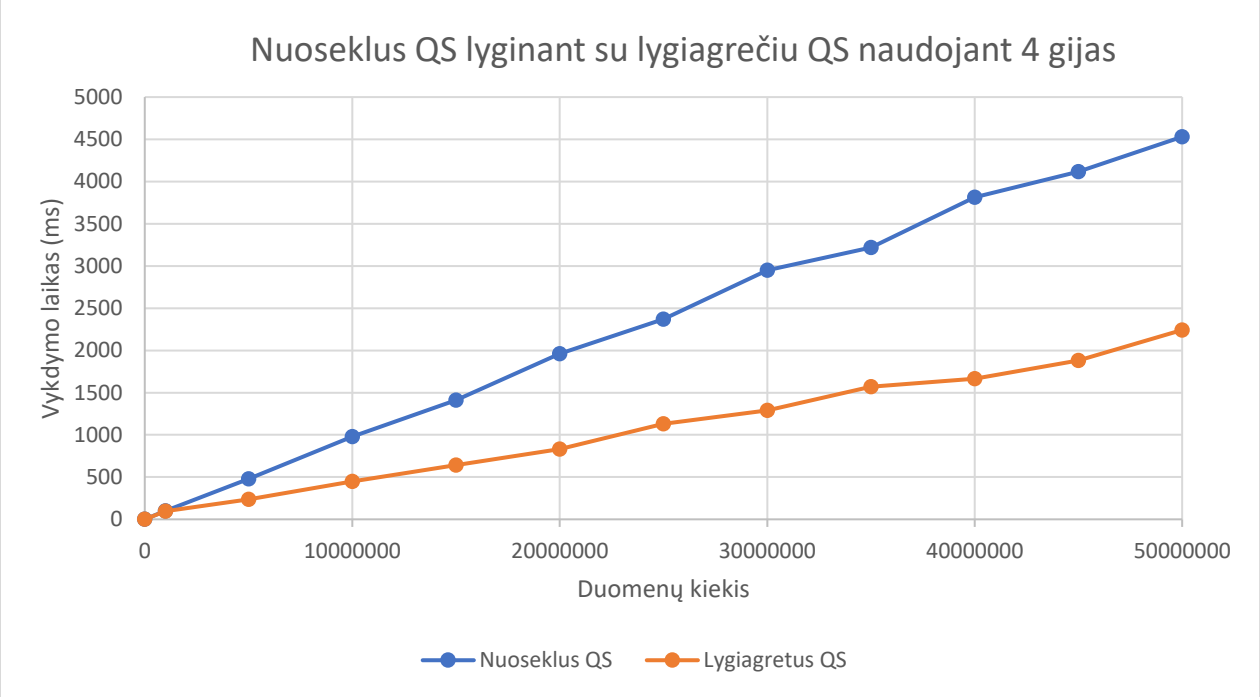
Buvo ištirtas programos vykdymo laikas naudojant duomenų kiekius nuo 1 milijono iki 300 milijonų. Svarbu paminėti, kad didesni laiko skirtumai tarp algoritmų vykdytų su skirting skaičiumi gijų galėjo atsirasti dėl skirtingos kompiuterio apkrovos vykdymo metu.

Duomenų kiekis	Laikų vidurkiai(ms)			
	Nuoseklus QS	Lygiagretus QS su 2 gijom	Lygiagretus QS su 4 gijom	Lygiagretus QS su 6 gijom
1000000	100	25	95	80
5000000	480	100	236	280
10000000	980	190	450	496
15000000	1410	284	640	680
20000000	1960	370	830	896
25000000	2370	465	1130	1080
30000000	2950	560	1290	1310
35000000	3220	620	1570	1630
40000000	3815	759	1664	1710
45000000	4115	800	1881	1950
50000000	4530	900	2242	2321
100000000	8930	1686	3930	3877
150000000	14863	2889	5915	5733
200000000	19808	3955	8735	7789
250000000	24843	5180	12454	11077
300000000	29429	5822	16394	14875

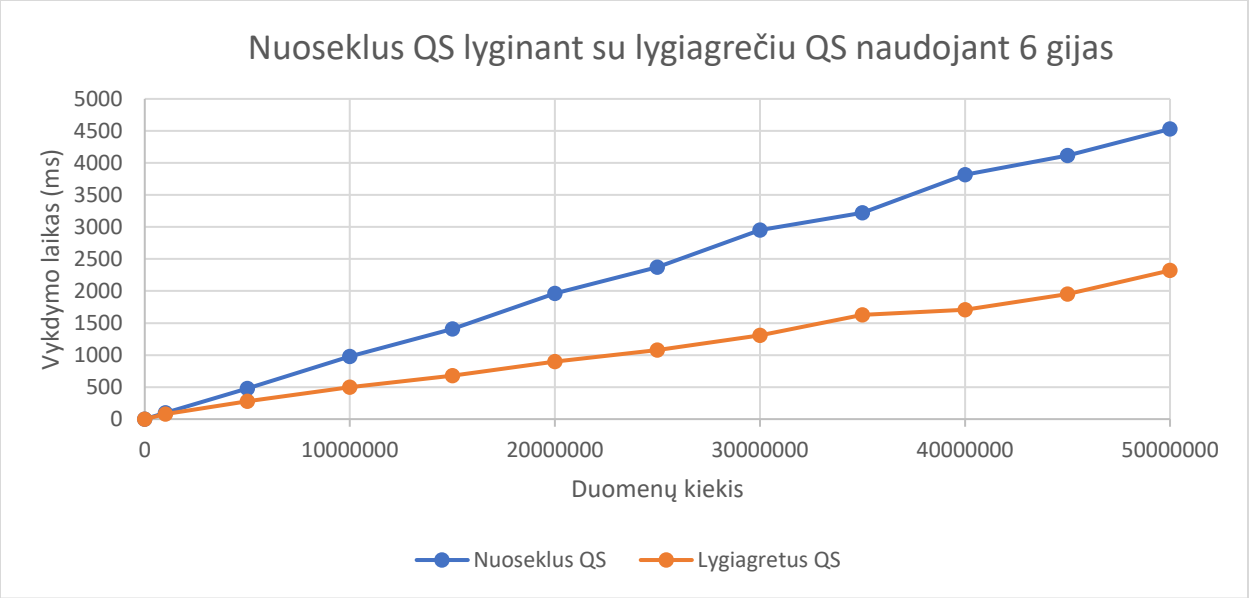
Lentelė Nr. 1 Kiekvieno algoritmo su atitinkamu skaičiumi gijų vidutinis veikimo laikas esant tam tikram skaičiui duomenų



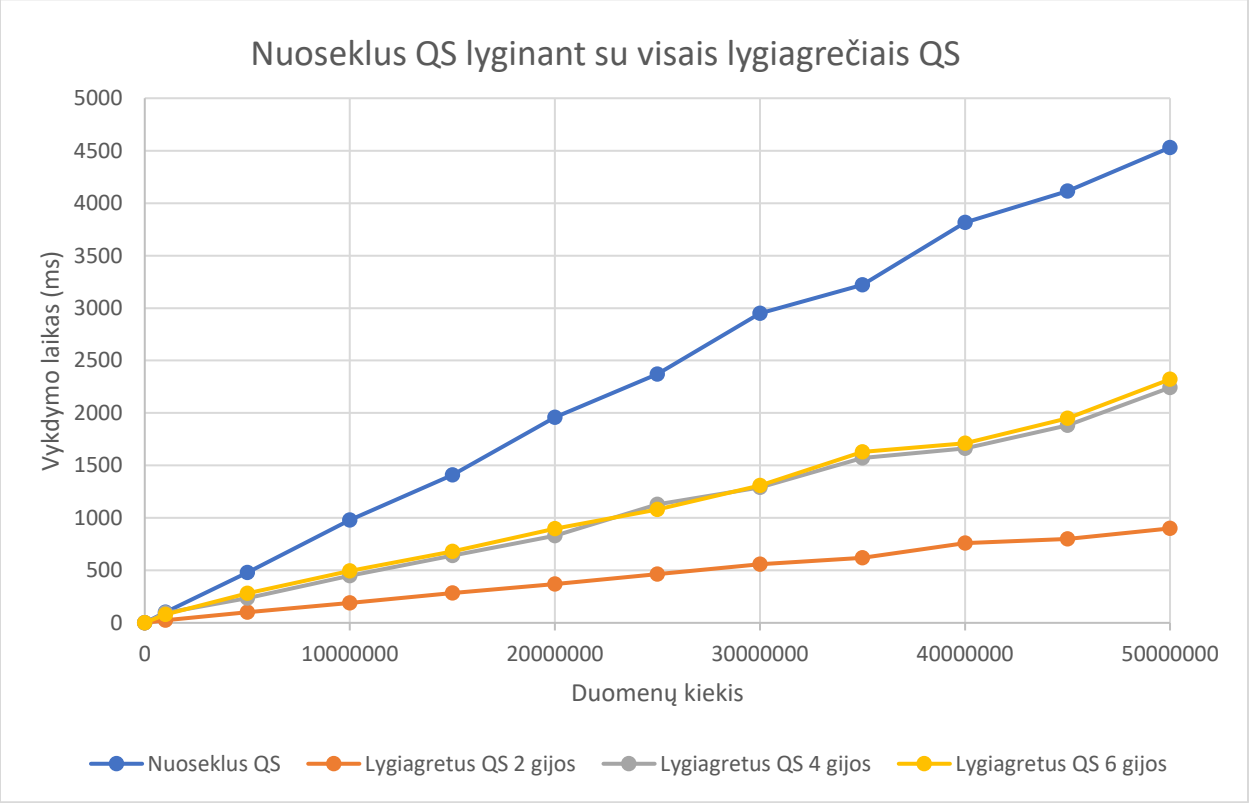
Grafikas Nr. 1 Nuoseklaus QS vykdymo laiko palyginimas su lygiagrečiu QS naudojančiu 2 gijas



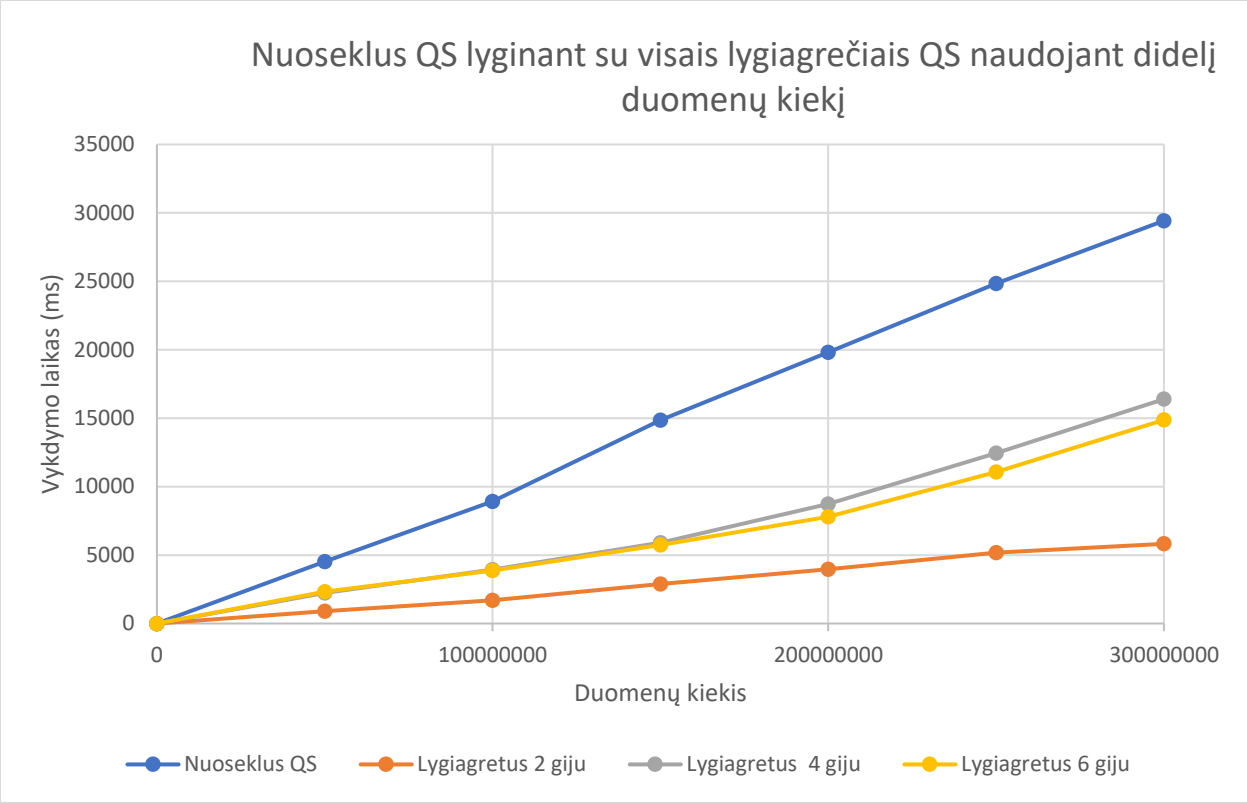
Grafikas Nr. 2 Nuoseklaus QS vykdymo laiko palyginimas su lygiagrečiu QS naudojančiu 4 gijas



Gafikas Nr. 3 Nuoseklaus QS vykdymo laiko palyginimas su lygiagrečiu QS naudojančiu 6 gijas



Grafikas Nr. 4 Nuoseklaus QS vykdymo laiko palyginimas su kiekvienu lygiagrečiu QS naudojančiu nuo 2 iki 6 gijų



Grafikas Nr. 5 Nuoseklas QS laiko palyginimas su kiekvienu lygiagrečiu QS, naudojant labai didelius duomenų kiekius (50-300milijonų)

Išvados ir literatūros sąrašas

Išvados: Programa lėčiausiai veikė nenaudojant lygiagretaus Quicksort algoritmo, o greičiausiai veikė naudojant lygiagretų Quicksort algoritmą ir 2 gijas. Veikimo laiko skirtumas, tarp lygiagretaus ir nelygiagretaus algoritmo didėjo, didinant duomenų kiekį ir buvo didžiausias naudojant 2 gijas, bei mažiausias naudojant 4 gijas. Naudojant 6 gijas laiko skirtumas buvo kiek mažesnis nei naudojant 4 gijas.

Literatūros sąrašas:

https://en.wikipedia.org/wiki/Quicksort

https://stackoverflow.com/questions/3425126/java-parallelizing-quick-sort-via-multi-threading

https://codereview.stackexchange.com/questions/121996/parallel-integer-quicksort-in-java

https://www.youtube.com/watch?v=dD4ls9cLnMk

https://www.youtube.com/watch?v=Jluv6uMwv3Q