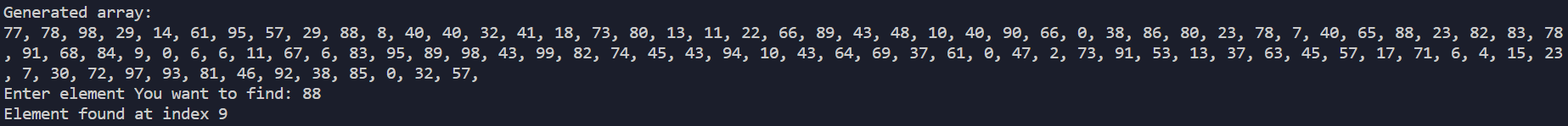
## Sastādīt programmas, kas paredzētas masīva apstrādei. Programmā paredzēt masīva aizpildi ar nejaušiem skaitļiem.

1. Uzrakstīt divas funkcijas, kas realizē masīva elementa meklēšanu: pirmo - lineāro meklēšanas algoritmu un otro - lineāro meklēšanas algoritmu ar barjeru. Salīdzināt divu algoritmu efektivitātes

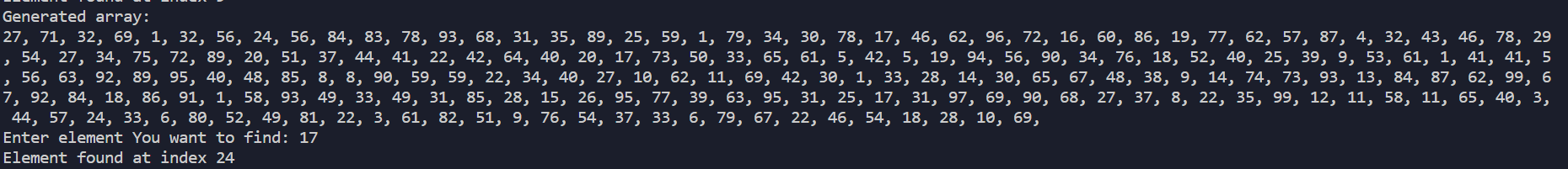
import java.util.Random;  
import java.util.Scanner;  
  
class first {  
  
 public static Scanner in = new Scanner(System.in);  
  
 public static int rand(int min, int max) {  
 Random random = new Random();  
  
 return random.nextInt(max - min) + min;  
 }  
  
 public static void print\_arr(int[] arr) {  
 String prnt = "";  
  
 for (int i = 0; i < arr.length; i++) {  
 prnt += arr[i];  
 prnt += ", ";  
 }  
  
 System.out.println(prnt);  
 }  
  
 public static int[] gen\_arr(int len) {  
 int[] arr = new int[len];  
  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = rand(0, 100);  
 }  
  
 return arr;  
 }  
  
 public static int linear\_search(int[] arr, int el) {  
 for (int i = 0; i < arr.length; i++) {  
 if (arr[i] == el) {  
 return i;  
 }  
 }  
  
 return -1;  
 }  
  
 public static int linear\_search\_barier(int[] arr, int el) {  
 int last\_index = arr.length - 1;  
 int last = arr[last\_index];  
 arr[last\_index] = el;  
  
 if (last == el) {  
 return last\_index;  
 }  
  
 int i = 0;  
  
 while (arr[i] != el & i != last\_index) {  
 i++;  
 }  
  
 if (i == last\_index) {  
 return -1;  
 }  
  
 return i;  
 }  
  
 public static void main(String[] args) {  
 int[] values = new int[] { 100, 200, 10000 };  
 long[][] times = new long[values.length][2];  
  
 for (int i = 0; i < values.length; i++) {  
 int val = values[i];  
  
 int[] arr = gen\_arr(val);  
  
 System.out.println("Generated array: ");  
 print\_arr(arr);  
  
 int el = -1;  
  
 try {  
 System.out.print("Enter element You want to find: ");  
 el = in.nextInt();  
 } catch (Exception e) {  
 System.out.print("Something went wrong");  
 in.close();  
 return;  
 }  
  
 int l\_s = -1;  
  
 long startTime = System.nanoTime();  
 l\_s = linear\_search(arr, el);  
 long endTime = System.nanoTime() - startTime;  
  
 times[i][0] = endTime;  
  
 startTime = System.nanoTime();  
 l\_s = linear\_search\_barier(arr, el);  
 endTime = System.nanoTime() - startTime;  
  
 times[i][1] = endTime;  
  
 if (l\_s < 0) {  
 System.out.println("Couldn't find the element You were looking for");  
 } else {  
 System.out.println("Element found at index " + l\_s);  
 }  
 }  
  
 System.out.println("Execution times:");  
 System.out.println("LS LSWB:");  
  
 for (long[] vals : times) {  
 String res = vals[0] < vals[1]  
 ? " < "  
 : (vals[0] == vals[1]) ? " = " : " > ";  
  
 System.out.println(vals[0] + res + vals[1]);  
 }  
 }  
}

1. reize (100 elementi)



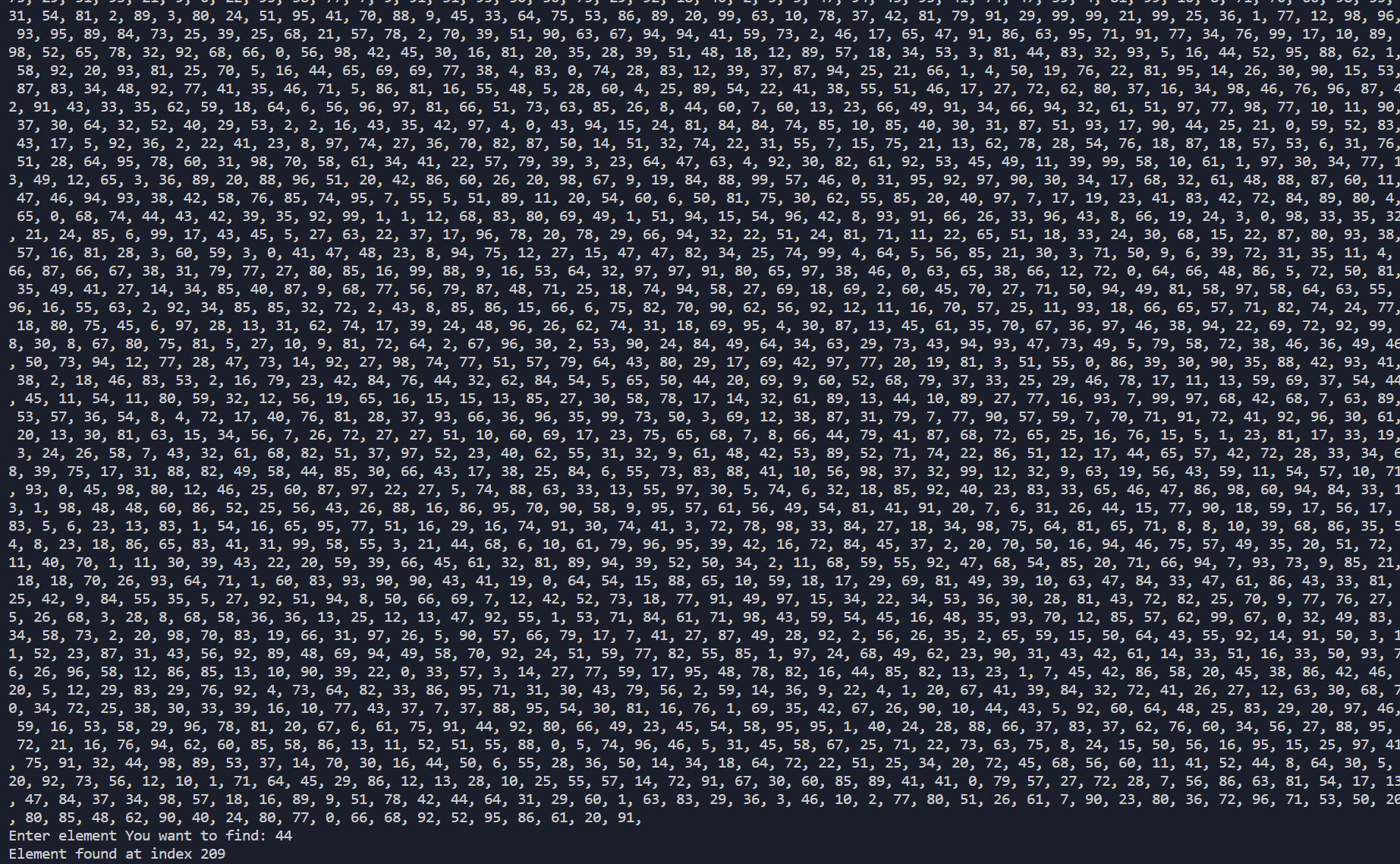
image

1. reize (200 elementi)



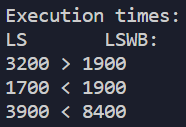
image

1. reize (1000 elementi)



image

Linear search ar barjeru bija ātraks 1. reizē, bet pārējās Linear search



image

1. Uzrakstīt funkciju, kas realizē masīva šķirošanas izvēlēs metodi (Select Sort). Noteikt algoritma efektivitāti

import java.util.Random;  
import java.util.Scanner;  
  
public class second {  
  
 public static Scanner in = new Scanner(System.in);  
  
 public static int rand(int min, int max) {  
 Random random = new Random();  
  
 return random.nextInt(max - min) + min;  
 }  
  
 public static int[] gen\_arr() {  
 int[] arr = new int[20];  
  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = rand(0, 100);  
 }  
  
 return arr;  
 }  
  
 public static void print\_arr(int[] arr) {  
 String prnt = "";  
  
 for (int i = 0; i < arr.length; i++) {  
 prnt += arr[i];  
 prnt += ", ";  
 }  
  
 System.out.println(prnt);  
 }  
  
 public static void select\_sort(int[] arr) {  
 int indexor = 0;  
  
 while(indexor < arr.length){  
 int smallest\_el = arr[indexor], smallest\_index = indexor;  
 int idx\_el = arr[indexor];  
  
 for (int i = indexor; i < arr.length; i++){  
 int el = arr[i];  
  
 if(el < smallest\_el){  
 smallest\_el = el;  
 smallest\_index = i;  
 }  
 }  
  
 arr[indexor] = smallest\_el;  
 arr[smallest\_index] = idx\_el;  
  
 indexor ++;  
 }  
  
 System.out.print("Sorted array: ");  
 print\_arr(arr);  
 }  
  
 public static void main(String[] args) {  
 int[] arr = gen\_arr();  
  
 System.out.print("Generated array: ");  
 print\_arr(arr);  
  
 select\_sort(arr);  
  
 in.close();  
 }  
}

image

image

1. Uzrakstīt funkciju, kas realizē masīva šķirošanas Hoāra metodi (Quick Sort). Noteikt algoritma efektivitāti

import java.util.Random;  
import java.util.Scanner;  
  
class third {  
  
 public static Scanner in = new Scanner(System.in);  
  
 public static int rand(int min, int max) {  
 Random random = new Random();  
  
 return random.nextInt(max - min) + min;  
 }  
  
 public static int[] gen\_arr() {  
 int[] arr = new int[20];  
  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = rand(0, 100);  
 }  
  
 return arr;  
 }  
  
 public static void print\_arr(int[] arr) {  
 String prnt = "";  
  
 for (int i = 0; i < arr.length; i++) {  
 prnt += arr[i];  
 prnt += ", ";  
 }  
  
 System.out.println(prnt);  
 }  
  
 public static int par(int[] arr, int low, int high) {  
 int pivot = arr[high];  
 int i = low - 1;  
  
 for (int j = low; j < high; j++) {  
 if (arr[j] <= pivot) {  
 i++;  
 int tmp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = tmp;  
 }  
 }  
  
 int temp = arr[i + 1];  
 arr[i + 1] = arr[high];  
 arr[high] = temp;  
  
 return i + 1;  
 }  
  
 public static void quick\_sort(int[] arr, int low, int high) {  
 if (low < high) {  
 int partition = par(arr, low, high);  
  
 quick\_sort(arr, low, partition - 1);  
 quick\_sort(arr, partition + 1, high);  
 }  
 }  
  
 public static void main(String[] args) {  
 int[] arr = gen\_arr();  
  
 System.out.print("Generated array: ");  
 print\_arr(arr);  
  
 quick\_sort(arr, 0, arr.length - 1);  
  
 System.out.print("Sorted array: ");  
 print\_arr(arr);  
 }  
}

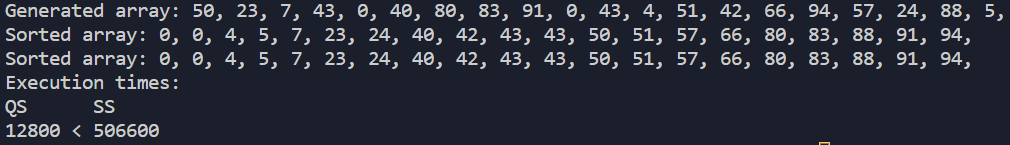
image

image

**Select Sort vs Quick Sort, kurš ātrāks?**

import java.util.Random;  
import java.util.Scanner;  
  
public class Main {  
  
 public static int rand(int min, int max) {  
 Random random = new Random();  
  
 return random.nextInt(max - min) + min;  
 }  
  
 public static int[] gen\_arr() {  
 int[] arr = new int[20];  
  
 for (int i = 0; i < arr.length; i++) {  
 arr[i] = rand(0, 100);  
 }  
  
 return arr;  
 }  
  
 public static void print\_arr(int[] arr) {  
 String prnt = "";  
  
 for (int i = 0; i < arr.length; i++) {  
 prnt += arr[i];  
 prnt += ", ";  
 }  
  
 System.out.println(prnt);  
 }  
  
 public static void select\_sort(int[] arr) {  
 int indexor = 0;  
  
 while (indexor < arr.length) {  
 int smallest\_el = arr[indexor], smallest\_index = indexor;  
 int idx\_el = arr[indexor];  
  
 for (int i = indexor; i < arr.length; i++) {  
 int el = arr[i];  
  
 if (el < smallest\_el) {  
 smallest\_el = el;  
 smallest\_index = i;  
 }  
 }  
  
 arr[indexor] = smallest\_el;  
 arr[smallest\_index] = idx\_el;  
  
 indexor++;  
 }  
  
 System.out.print("Sorted array: ");  
 print\_arr(arr);  
 }  
  
 public static int par(int[] arr, int low, int high) {  
 int pivot = arr[high];  
 int i = low - 1;  
  
 for (int j = low; j < high; j++) {  
 if (arr[j] <= pivot) {  
 i++;  
 int tmp = arr[i];  
 arr[i] = arr[j];  
 arr[j] = tmp;  
 }  
 }  
  
 int temp = arr[i + 1];  
 arr[i + 1] = arr[high];  
 arr[high] = temp;  
  
 return i + 1;  
 }  
  
 public static void quick\_sort(int[] arr, int low, int high) {  
 if (low < high) {  
 int partition = par(arr, low, high);  
  
 quick\_sort(arr, low, partition - 1);  
 quick\_sort(arr, partition + 1, high);  
 }  
 }  
  
 public static void main(String[] args) {  
 int[] arr = gen\_arr();  
 int[] initArr = arr;  
  
 System.out.print("Generated array: ");  
 print\_arr(arr);  
  
 long startTime = System.nanoTime();  
 quick\_sort(initArr, 0, arr.length - 1);  
 long endTime = System.nanoTime() - startTime;  
  
 long startTime1 = System.nanoTime();  
 select\_sort(arr);  
 long endTime1 = System.nanoTime() - startTime1;  
  
 System.out.print("Sorted array: ");  
 print\_arr(arr);  
  
 System.out.println("Execution times:");  
 System.out.println("QS SS");  
  
 System.out.println(endTime + (endTime > endTime1 ? " > " : " < ") + endTime1);  
 }  
}

**Quick sort ātrāks**



image