5 Instruction – Searching, Multidimensional arrays

Note: exercise 5.1 has to be submitted to Peach. It contributes to your homework grade. It will not be tested on its output, since the output format is not exactly specified in this exercise.

5.1 Compare searching

Write a program that compares the following three searching algorithms for finding a key value in an integer array a. a is sorted because the second and third algorithm only work for sorted arrays.

1. Linear search – searches for the key by traversing a until the key is found.
2. Binary search – searches for the key by repeatedly halving the interval of the array where the key belongs.
3. Interpolation sort – searches for the key in the same manner as binary search, but, using the observation that if the values in the array are fairly evenly distributed, the value of the key can be used to make a better approximation to its index than the midpoint provides: use instead $low + [\frac{x - a[low]}{a[high] - a[low]}] * (high - low - 1)$, where low and $high$ are the lower and upper bound of the interval. This formula might need some adjustment, depending on your use of $high$ and low (in particular, whether they the interval does or does not include $high$ and low).

Write functions that return the index of the key if found and a negative value if not found.

Use as a measure of efficiency the number of steps (loop traversals) that an algorithm uses to find a key value.

Write a program that compares the efficiency of the three algorithms on arrays of length 10, 100, and 1000 respectively. It displays the average number of steps for each algorithm for 25 arrays of each length.

Generate random values for the arrays in a large interval. Generate the key as a random value in this interval. For a fair comparison, apply the algorithms to the same arrays.

Think about how you will obtain a sorted array. One possibility is to use the method `void Arrays.sort(int[] a)` from the package `java.util`. E.g., the command `Arrays.sort(myArray);` will sort the array `myArray` in place, so the argument will be changed into a sorted one. Another possibility is to generate the array such that it is sorted. This may give a more uniform distribution of the values and therefore be more suited for interpolation search.

The program should be well-structured in that it that contains at least methods for each of the search procedures, a method that randomly fills the array, and a method that outputs the results.

First, think about what numbers to expect.

Present key

Now generate the key by generating a random index in the array. This way, the key is always present. Does this make a difference for the execution times of the algorithms? Is there a difference between the algorithms in this respect?

5.2 Tournament

In a *double round-robin tournament*, every team plays two matches against each other team, one home match and one away match. In this exercise, you are going to build a system to record and analyze these kind of tournaments.

Input is the number of teams in the tournament or the names of the teams.

Output is a matrix with the results of all matches, the number of points achieved by each team, and the winner of the tournament.

To avoid long input sessions, the match results are randomly generated. The results are to be stored in a two-dimensional array, 0 for a loss, 1 for a draw, and 3 for a win of the home playing team.

5.3 Input

- The number of teams; or
- the names of the teams, separated by white space, and ended, after white space, by a `#`; a team name doesn't contain spaces.

If the the number of teams is provided, the names are to be taken from a built-in list, if necessary completed with generated names (e.g., "A", "B", etc.).

5.4 Output

1. The matrix with results. The home playing teams are on the vertical axis. Use three positions for the columns, separated by `|`-symbols. Display only the first three characters of the team names and add spaces at the end for names that are shorter than three characters.

   After each row comes the number of match points the team of that row has achieved.

2. The text: "Champion:" followed by the name of the team with the most points. If there are more teams that have gained the most points, the text should be "tie between:", followed by the names of these teams.

5.5 Example

Input: 4

Output:

```
     AJA|NEC|NAC|NÖC
AJA | - | 0 | 0 | 0 | 0
NEC | 3 | - | 3 | 0 | 10
NAC | 3 | 1 | - | 1 | 8
NÖC | 3 | 3 | 3 | - | 16

Champion:
NÖC
```

5.6 Extras

1. Record in the matrix the score balance, i.e., the number of goals scored by the away team subtracted from the number of goals scored by the home team. Generate for this a random number between -5 and +5. At the end of each row in the matrix, display both the number of match points and the score balance. When two teams have an equal number of match points, the total score balance will be used as tie breaker.

2. Record both the number of goals scored for and against. Use two matrices (two-dimenstional arrays) for this. Display the result as $k - l$, where k is the number of goals scored by the home team. Generate goal numbers between 0 and 5.

   Now the total amount of goals scored is a tie breaker when the goal saldo doesn't decide.