# Searching with data structures

Gerben Aalvanger
Student-number: 3987051
Utrecht University

Erik Visser
Student-number: 3470806
Utrecht University

William Kos
Student-number: 3933083
Utrecht University

Sam van der Wal
Student-number: 3962652
Utrecht University

13 april 2015
Teacher: Peter de Waal

## 1 Introduction

Searching values and storing them in memory is a key concept in computer science. For optimal speed and complexity, data is stored in a lot of different data structures, for example a tree or a skip-list. Every data structure has its own advantages, some perform better on insertion, while other perform better on finding a specific value. In this document we will describe how we will compare data structures and actually compare them. We will start by proposing a research question and some sub-questions. Secondly we specify the problem we want to research and the scope of our project. After that we explain our experiments in terms of criteria, test data and scenarios. And last we will show in different ways our test results and compare them to each other and end with a conclusion.

## 2 Research description

In this research the question "Which data structure has the shortest duration on given actions" has been put central. We have used 4 data structures to get the answer to this question. To get the best insight in this question with 4 data structures we had chosen to take data structures which differ a lot from each other, namely:

- Lists
- Balanced trees
- Hash tables
- Min-max heaps

The list and hash table are both straight forward data structures, but there are multiple implementations possible for balanced trees and min-max heaps. For the balanced tree we had chosen to use the AVL tree because it is similar to the red-black tree, but known to be faster on lookups.

The min-max heap implementation we have used in our experiments is the interval heap. Our initial idea was to use the original min-max heap implementation but after implementing this algorithm it turned out that it did not work according to what was described in the paper we found. Therefore the decision was made to use the interval heap since the complexity for the individual actions are the same as with the min-max heap. All the actions used in our research will be described in the next section.

Part of the research was performing statistics on the results gained from the experiments. The focus laid at the average and the standard deviation of the performed actions on special cases. Because the same tests were performed multiple times we have chosen for both the average and the standard deviation to gain a good insight in the overall results retrieved from the experiments.

## 3 Experiment description

We will separate the experiment in several section:

### 3.1 Build

In the build we have "build" the different datastructures. For the build operation we have build from 15 different unsorted datasets, this is divided in datasets of size $10.000, 100.00$ and $1.000.000$, each size occurring 5 times. For each of these 15 options we have also tested the sorted and reverse sorted set. Each dataset is tested 30 times. This means that we have conducted all tests with 1350 distinct datasets.

### 3.2 Search

We have searched in the pre-build datastructures of section 3.1. The amount of searches is half the amount of elements in the datastructure. All the values of the searches are determined random and are guaranteed to be in the datastructure. Like in the build section we have performed all search-tests 30 times.

We wanted to know if finding the minimum or maximum through the getMin (see section 3.5) and getMax (see section 3.6) methods of the datastructures are faster then searching for the minimum or maximum value with this search method. This test is conducted 1000 times on the datasets described in section 3.1

### 3.3 Insert

We have constructed new datastructures from scratch by repeatedly inserting new elements. Like in section 3.1 we have tested the insert actions for 15 sets of elements (set sizes: $10.000, 100.000, 1.000.000$) and each of these tests have been performed 30 times.

### 3.4 Delete

We have performed delete operations on the pre-build datastructures of section 3.1. The amount of deletes is half the amount of elements in the datastructure. All the values of the deletes are determined random and are guaranteed to be in the datastructure. Like in the build section we have performed all delete-tests 30 times.

### 3.5 getMin

We performed getMin operations on the pre-build datastructures of section 3.1. The amount of getMin operations in a test is 1000 since it returns always the same value. GetMin returns the lowest value in the datastructures. Like in the build section we performed all getMin-tests 30 times.

### 3.6 getMax

We performed getMax operations on the pre-build datastructures of section 3.1. The amount of getMax operations in a test is 1000 since it returns always the same value. GetMax returns the highest value in the datastructures. Like in the build section we performed all getMax-tests 30 times.

### 3.7 extractMin

We performed extractMin operations on the pre-build datastructures of section 3.1. The amount of extractMin operations in a test is half the amount of the elements in the datastructure. ExtractMin returns the lowest value in the datastructures and then deletes this entry. Like in the build section we performed all extractMin-tests 30 times.

### 3.8 extractMax

We performed extractMax operations on the pre-build datastructures of section 3.1. The amount of extractMax operations in a test is half the amount of the elements in the datastructure. ExtractMax returns the highest value in the datastructures and then deletes this entry. Like in the build section we performed all extractMax-tests 30 times.

## 4 Results

Below, we present the results of our experiments. We start in (4.1) by presenting the results per datastructure and dataset. The values presented in the cells are in the format *mean ±standard deviation*. In the tables, we focus on the differences between datastructures, therefore, we have only selected results for random datasets. In (4.2) we also focus on the differences between datasets. This differences are presented in the last two bar charts. For each bar chart, on the horizontal axis, we give the different datasets and datastructures. The height of the bars represent the amount of time (in milliseconds), one action takes. The first two graphs represent the differences between searching for a value, being the min/max value and explicitly asking for the min/max value. In (4.3) we explain our hypotheses, method of testing and results of our statistical tests.

## 4.1 Tables

.

**BUILD** (random)

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 4,2 ±0,4 | 3 ±0 | 2 ±0 | 1 ±0 |
| **100.000** | 52,4 ±0,5 | 36,4 ±0,5 | 18,2 ±4 | 29,2 ±1,2 |
| **1.000.000** | 663, 6 ±48,5 | 1032,8 ±31,4 | 177 ±2,3 | 610,4 ±9,2 |

**SEARCH** (random)

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 6 ±0 | 1 ±0 | 1118,4 ±4,5 | 1 ±0 |
| **100.000** | 75,8 ±3,4 | 12,8 ±0,7 | 115007,4±3112,6 | 6,8 ±1,8 |
| **1.000.000** | 991 ±40,3 | 211,4 ±11,9 | timeout | 106,6 ±5,8 |

**INSERT** (random)

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 6 ±0 | 1 ±0 | 1118,4 ±4,5 | 1 ±0 |
| **100.000** | 75,8 ±3,4 | 12,8 ±0,7 | 115007,4±3112,6 | 6,8 ±1,8 |
| **1.000.000** | 991 ±40,3 | 211,4 ±11,9 | timeout | 106,6 ±5,8 |

**DELETE** (random)

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 16 ±0 | 1,2 ±0,4 | not tested | 1 ±0 |
| **100.000** | 2022,2 ±6,1 | 21,2 ±5,4 | not tested | 9,2 ±2,9 |
| **1.000.000** | 510015,8 ±95899,4 | 480 ±37,9 | not tested | 128 ±2,1 |

**GETMIN**

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 1 ±0 | 1 ±0 | 1 ±0 | 3230,8 ±11,1 |
| **100.000** | 1 ±0 | 1 ±0 | 1 ±0 | 37349 ±67,6 |
| **1.000.000** | 1 ±0 | 1,2±0,4 | 1 ±0 | 874572,4±3270,8 |

**GETMAX**

| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 1 ±0 | 1 ±0 | 1 ±0 | 3174 ±7,9 |
| **100.000** | 1 ±0 | 1 ±0 | 1 ±0 | 46396,6 ±6468,1 |
| **1.000.000** | 1 ±0 | 1,2 ±0,4 | 1 ±0 | 905490,6 ±20239,3 |

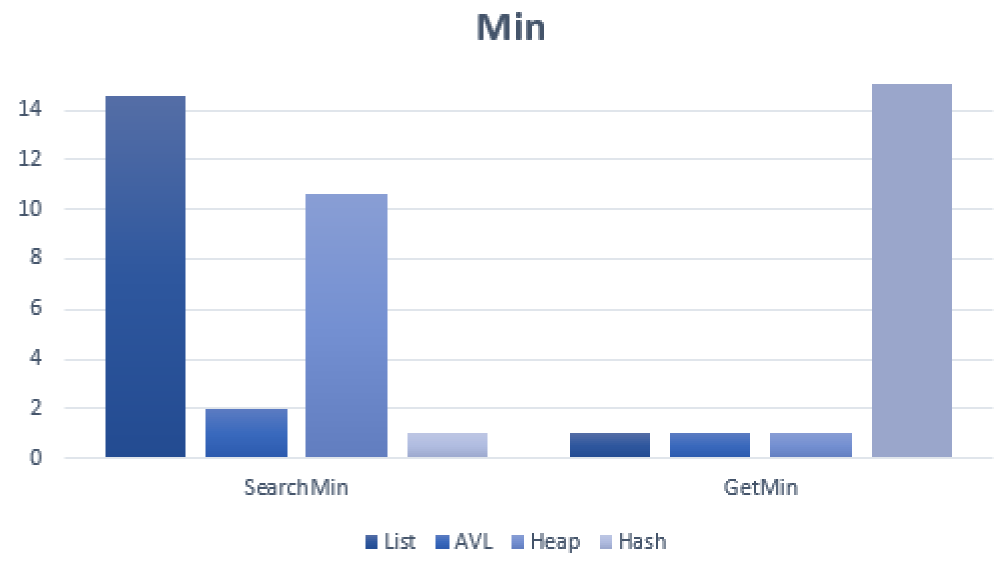
**EXTRACTMIN**

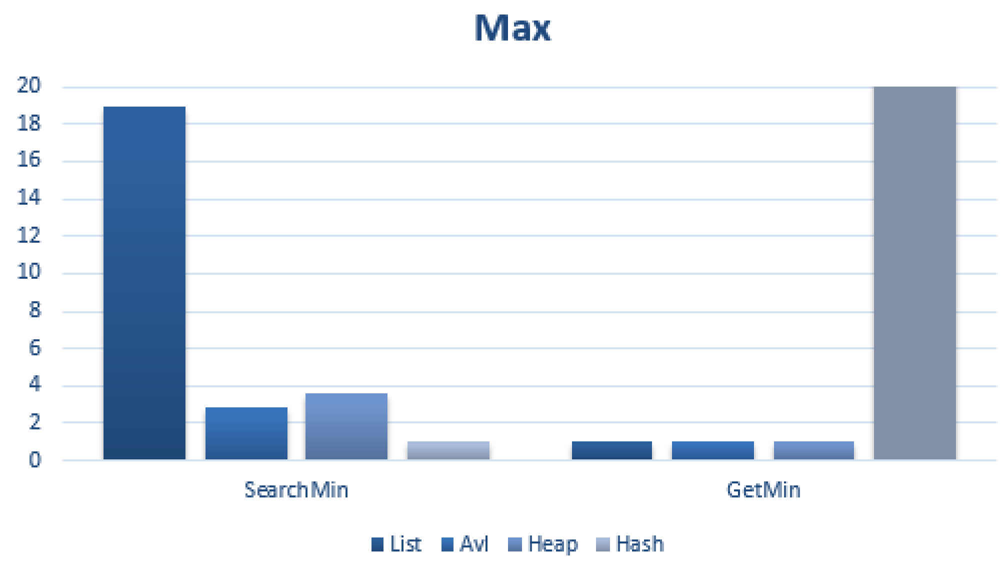
| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 28 ±0 | 1 ±0 | 6±0 | 1445,2 ±3 |
| **100.000** | 4012,2 ±4,7 | 12,8 ±3,9 | 72,2±0,4 | 164378,6 ±2414,6 |
| **1.000.000** | 836228 ±198929,6 | 109,4 ±0,8 | 928,8 ±13 | timeout |

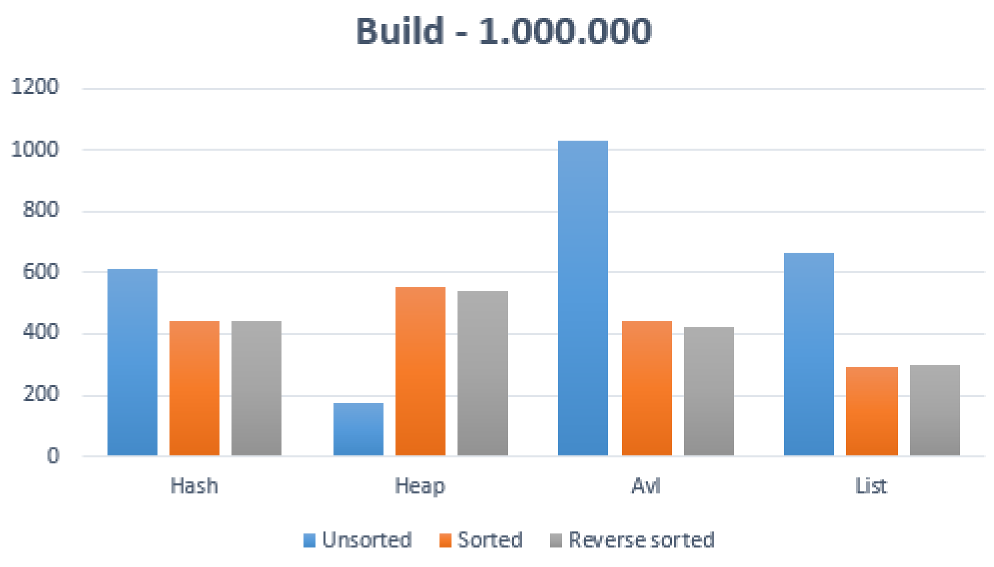
**EXTRACTMAX**

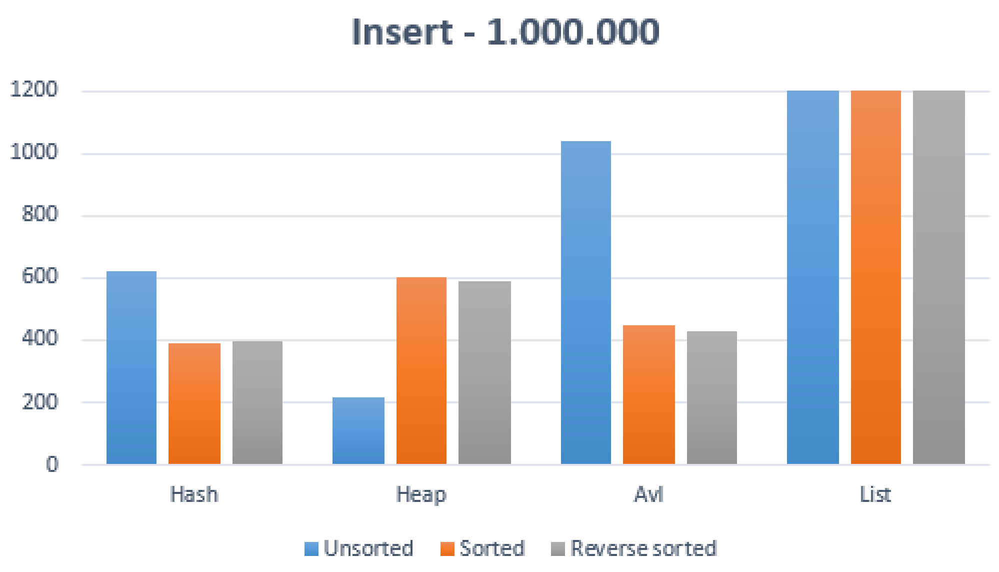
| | **List** | **AVL Tree** | **Interval Heap** | **Hash table** |
|---|---|---|---|---|
| **10.000** | 4 ±0 | 1 ±0 | 6,2 ±0,4 | 1444,4 ±2 |
| **100.000** | 41 ±0 | 10 ±0 | 75±0 | 165799 ±299,2 |
| **1.000.000** | 482,6 ±25,7 | 110,2 ±0,7 | 951,6±1 | timeout |

## 4.2 Graphs









## 4.3 Hypotheses and statistical tests

### 4.3.1 Hypotheses changes

While testing the already changed second hypotheses: ”The search action will be executed the fastest by the hash table and the AVL tree and the slowest by the Min-Max tree”, we noticed a significant difference between the speed of the hash table and the AVL tree. We realized that we should be more strict in our hypotheses and therefore we changed some other hypotheses before testing them. We changed the following hypotheses:

- The build action will be executed the fastest by the hash table and slowest by the Min-Max heap.

- The search action will be executed the fastest by the hash table and the AVL tree and the slowest by~~ the list and ~~the Min-Max heap.
- The insert action will be executed the fastest by the hash table~~, the AVL tree and the Min-Max heap~~ and slowest by the list.
- The delete action will be executed the fastest by the hash table ~~and the AVL tree~~ and the slowest by ~~the Min-Max heap and~~ the list.
- The getMin action will be executed the fastest by the list and the Min-Max heap and the slowest by the hash table.
- The extractMin action will be executed the fastest by ~~the AVL tree and~~ the Min-Max heap and the slowest by the hash table.
- The getMax action will be executed the fastest by the list and the Min-Max heap and the slowest by hash table.
- The extractMax action will be executed the fastest by the list and the Min-Max heap and the slowest by the hash table.

### 4.3.2 Hypotheses testing

For all our hypotheses we use a dependend sample T-test to identify whether a hypotheses holds or not. We create new sub-hypotheses for all our hypotheses. A hypothesis consists of 3 parts, which we will name A, B and C. The general structure of a hypothesis is:

The A action will be executed fastest by B1 and B2 ... and slowest by C1 and C2 .. .

Here A determines the action executed, actions are specified in section 3 $B = B1..Bn$ referes to n datastructures, which should execute A the fastest. $C = C1..Cn$ referes to n datastructures, which should execute A the slowest. We also define D as the set of all

To make the general hypotheses true, several sub-hypotheses should be accepted For all combinations $b \in B$ and $d \in D - B$, create the next hypothesis: b is significantly faster than d on A, with every datastructure-size 10.000, 100.000, 1.000.000. For all combinations $c \in C$ and $d \in D - C$, create the next hypothesis: c is significantly slower than d on A, with every datastructure-size 10.000, 100.000, 1.000.000.

For testing, we used excel's one-sided T-tests. To check correctness of our implementation, we checked the first outcomes of our handy excel-sheet to an online T-tester.

### 4.3.3 Statistics results

- The build action will be executed the fastest by the hash table and slowest by the Min-Max heap.
  This hypothesis is not accepted, since the Min-Max heap is faster than the hash table on the build action.
- The search action will be executed the fastest by the hash table and the AVL tree and the slowest by the Min-Max heap.

This hypothesis is accepted, since each sub-hypothesis is accepted. Th maximum p-vale of the sub-hypotheses is $1.61 \times 10^{-6}$

- The insert action will be executed the fastest by the hash table and slowest by the list.
  On the unsorted list is the Min-Max heap significantly faster than the hash table. Here a sub-hypothesis is not accepted. However, on the sorted and reverse sorted list, the hash is indeed significantly faster than the Min-Max heap. Here our main hypothesis is accepted.
- The delete action will be executed the fastest by the hash table and the slowest by the list.
  In this test, we did not use the values of the Min-Max heap, since the delete function is not defined on the Min-Max heap. The hypothesis is accepted on the bigger datastructures (+100.000 elements), with a maximum p-value of 0,003450688.
- The getMin action will be executed the fastest by the list and the Min-Max heap and the slowest by the hash table.
  It appears that the List and the Min-Max heap are significantly faster than the hash table. The hash is also significantly slower than all other datastructures. However, the List and the Min-Max heap are not significantly faster than the AVL-tree. The hypothesis is not accepted.
- The extractMin action will be executed the fastest by the Min-Max heap and the slowest by the hash table.
  This hypothesis is not accepted, since the AVL-tree is significantly faster than the Min-Max heap.
- The getMax action will be executed the fastest by the list and the Min-Max heap and the slowest by hash table.
  It appears that the List and the Min-Max heap are significantly faster than the hash table. The hash is also significantly slower than all other datastructures.
  However, the List and the Min-Max heap are not significantly faster than the AVL-tree. The hypothesis is not accepted.
- The extractMax action will be executed the fastest by the list and the Min-Max heap and the slowest by the hash table.
  This hypothesis is not accepted, since the AVL-tree is significantly faster than the Min-Max heap.

## 5 Discussion en conclusion

It appears that not all hypotheses hold. Especially, the hypotheses about min/max operations are not accepted. This is mainly because we underestimated the performance of the AVL-tree. Apparently, getting the min or max value from an AVL tree is fast, even though it is an $O(lg\, n)$ instead of an $O(1)$ operation. Besides this, the AVL tree outperforms the Min-Max heap on its specialty, the extraction of min/max values. An other hypothesis that does not hold is the build action, where the hash-table is not faster than the Min-Max heap. This could be caused by the hash-function (modulo), which is quite an expensive operation. The insert hypothesis is also not accepted, since an balanced heap (due to random input) is faster on insertion than the hash-table. This could also be caused by the expensive hash-function.

The other two hypotheses hold, the AVL-tree and hash-table are fastest on searching. Deletion is fastest on the hash-table.

In general, we could conclude that the AVL-tree performs quite well on each action, the heap is not proven to be fastest on its specialty and the hash performs best on search and deletion. The list is useful for getMin/Max operations, but does not perform too well on other operations like insert and delete.

## 6 Reflection

Although we cannot conclude that one of the datastructures outperforms all the others, we still observed some interesting results. This research could be used to reformulate the hypotheses and check whether the AVL-tree is never significantly slower than the other datastructures.
We already changed our hypotheses by making them more strict. Only 3 out of the 8 hypotheses were accepted. Because we lacked time, we did not change them again and start new tests.
Time was the biggest problem of our research. Firstly, some actions took way too long, still the computer used more than one week to compute the results. Besides that, we did not have enough time to restart the research with new hypotheses after we got hypotheses not accepted.