REPORT

The goal of the experiment is to demonstrate the speed difference for searching between a BST and a traditional array. Two applications were used which are VaccineBSTApp and VaccineArrayApp to conduct the experiment.

These applications use OO design to accomplish the task at hand. VaccineArrayApp depends on Vaccination and VaccineArray classes meaning it has most dependency whereas Vaccination class it least dependent on other classes. VaccineArrayApp creates VaccineArray object to use its methods, VaccineArray creates Vaccination object to build the data structure and Vaccination class implements the Comparable interface. VaccineArrayApp uses Vaccination class constructor and VaccineArray method to **add** vaccines data (*country*, *date* and *numOfVaccines*) from the *vaccinations.csv* file to create the traditional array. The array data structure is then queried using input from the user *(date+country)* which is invoked by the userInterface method in the VaccineArrayApp by using the **search** method from the VaccineArray class. In this way input from the user can be compared with contents in the array data structure using the **compareTo** method from the Vaccination class.

VaccineBSTApp uses BinarySearchTree class, BinarySearchTree class extends BinaryTree, extends Comparable and uses Vaccination, BinaryTree uses BinaryTreeNode therefore BinaryTreeNode has least dependencies whereas VaccineBSTApp has most dependencies. VaccineBSTApp creates BinarySearchTree<Vaccination> generic type object to **insert** data from the *vaccinations.csv* file to create a binary tree data structure using the insert method from the BinarySearchTree class. The binary tree data structure is then queried using the input from the user *(date+country)* which is done by the userInterface method in the VaccineBSTApp which invokes the **find** from the BinarySearchTree class. In this way input from the user can be compared with contents in the binary tree data structure invoking the **compareTo** method found in the Vaccination class.

In trial runs of Part 1 of the experiment, trial 1 input was **2022-02-17** with country names **Norway, Suriname, Italy, and France.**

For VaccineArrayApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Norway = 5803, 2314 operations
Suriname = 134, 2324 operations
Italy = 181285, 5875 operations
France = 161028, 1428 operations
```

For VaccineBSTApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Norway = 5803, 42 operations
Suriname = 134, 28 operations
Italy = 181285, 42 operations
France = 161028, 36 operations
```

Trial 2 input was **2022-01-09** with country names **Namibia, Mali, South Africa, Kenya, and Zimbabwe.**

For VaccineArrayApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Namibia = 908, 9801 operations
Mali = 60615, 3596 operations
South Africa = 50049, 7030 operations
Kenya = 67992, 7616 operations
Zimbabwe = 12119, 486 operations
```

For VaccineBSTApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Namibia = 908, 58 operations
Mali = 60615, 50 operations
South Africa = 50049, 42 operations
Kenya = 67992, 44 operations
Zimbabwe = 12119, 20 operations
```

Trial 3 input was **2022-03-12** with country names **Congo, Malaysia, Denmark, Republic of Congo, South Asia, and North America.**

For VaccineArrayApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Congo = <Not Found>, 9919 operations
Malaysia = 211152, 4452 operations
Denmark = 33752, 5643 operations
Republic of Congo = <Not Found>, 9919 operations
South Asia = <Not Found>, 9919 operations
North America = 2328630, 3942 operations
```

For VaccineBSTApp the output was:

```
Enter the date:
Enter the list of countries (end with an empty line):
Results:
Congo = <Not found>, 21 operations
Malaysia = 211152, 38 operations
Denmark = 33752, 32 operations
Republic of Congo = <Not found>, 12 operations
South Asia = <Not found>, 16 operations
North America = 2328630, 54 operations
```

To execute the experiment, I created two python scripts which are **createInputs.py** and **runApp.py**. **createInputs.py** generates a query list which includes a random **date** and **countries**. The list is generated using the vaccinations.csv file. The code uses **csv.reader()** built-in function which reads each line (country, date, vaccine) of the file converting each line into a list [country, date, vaccines] it further uses the **list()** function to make a 2D array data structure [[country1, date1, vaccines1], [country2, date2, vaccines2], …]. In this way I could easily extract dates and countries from the data structure to generate random inputs. Every input file has a random date and a list of random countries. I created in total 5 lists of data each having different sizes.

runApp.py creates subsets of lists (n = 991, 1982, …., 9910) using the vaccinations.csv therefore at each iteration it compares the query list against each subset. By running VaccineArrayApp and VaccineBSTArray in the **runApp.py** script this process is automated. While the comparisons are done using the **search()** and **find()** methods in the applications search operations are counted and recorded. The tables below show the results of the traditional array and the binary tree data structures.

VaccineArrayApp Search operations

Countries	n = 991	n = 1982	n = 2973	n = 3964	n = 4955	n = 5946	n = 6937	n = 7928	n = 8919	n = 9910
Taiwan	398	398	398	398	398	398	398	398	398	398
Guyana	991	1982	2973	3964	4955	5964	6937	7928	8438	8438
Ghana	991	1982	2973	3667	3667	3667	3667	3667	3667	3667
Lower middle income	991	1099	1099	1099	1099	1099	1099	1099	1099	1099
Suriname	991	1812	1812	1812	1812	1812	1812	1812	1812	1812
Iceland	991	1982	2973	3481	3481	3481	3481	3481	3481	3481
Russia	991	1982	2973	3964	4955	5964	6937	7659	7659	7659
Costa Rica	991	1982	2973	3964	4955	5964	6937	7928	8418	8418
San Marino	991	1652	1652	1652	1652	1652	1652	1652	1652	1652
Germany	991	1982	2058	2058	2058	2058	2058	2058	2058	2058

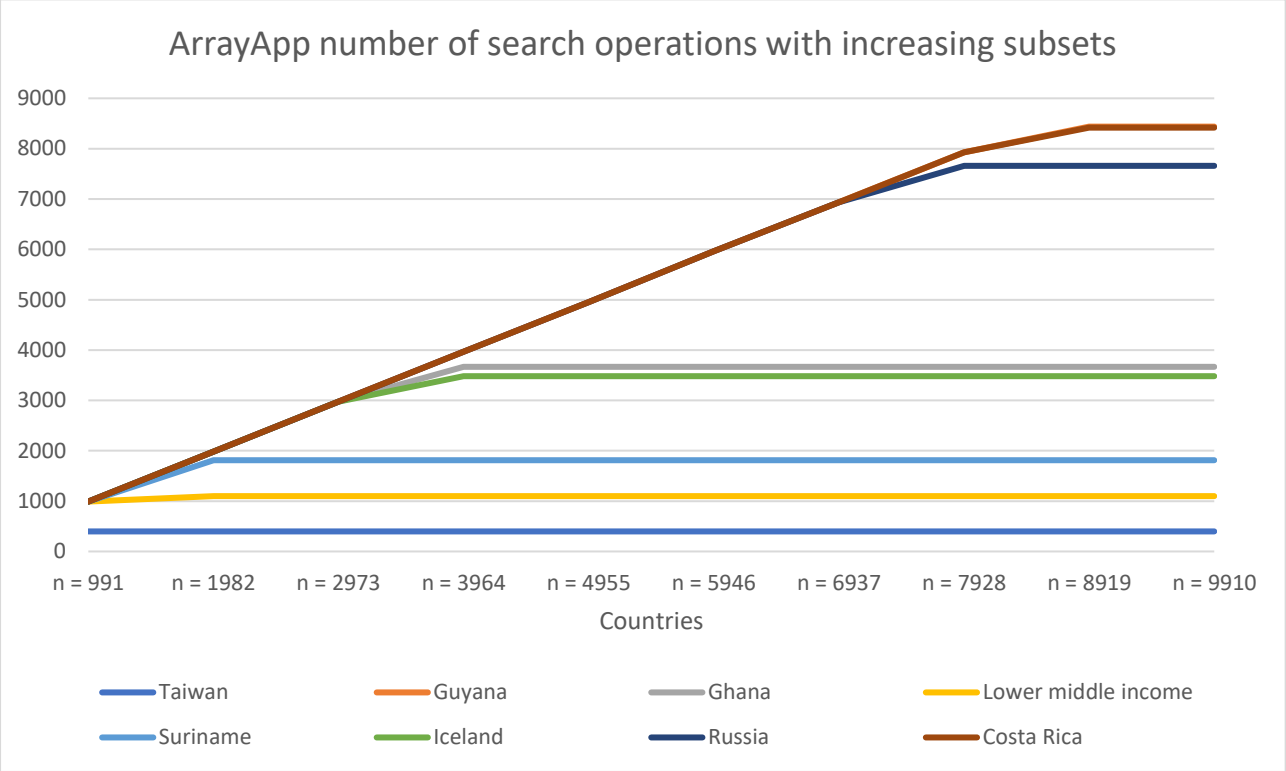
VaccineBSTApp Search operations

Countries	n = 991	n = 1982	n = 2973	n = 3964	n = 4955	n = 5946	n = 6937	n = 7928	n = 8919	n = 9910
Taiwan	18	18	18	18	18	18	18	18	18	18
Guyana	12	13	13	13	15	15	16	17	36	36
Ghana	14	16	16	34	34	34	34	34	34	34
Lower middle income	19	40	40	40	40	40	40	40	40	40
Suriname	14	30	30	30	30	30	30	30	30	30
Iceland	12	15	17	36	36	36	36	36	36	36
Russia	14	15	16	18	19	19	19	40	40	40
Costa Rica	10	12	14	15	16	16	16	16	34	34
San Marino	14	36	36	36	36	36	36	36	36	36
Germany	14	16	34	34	34	34	34	34	34	34

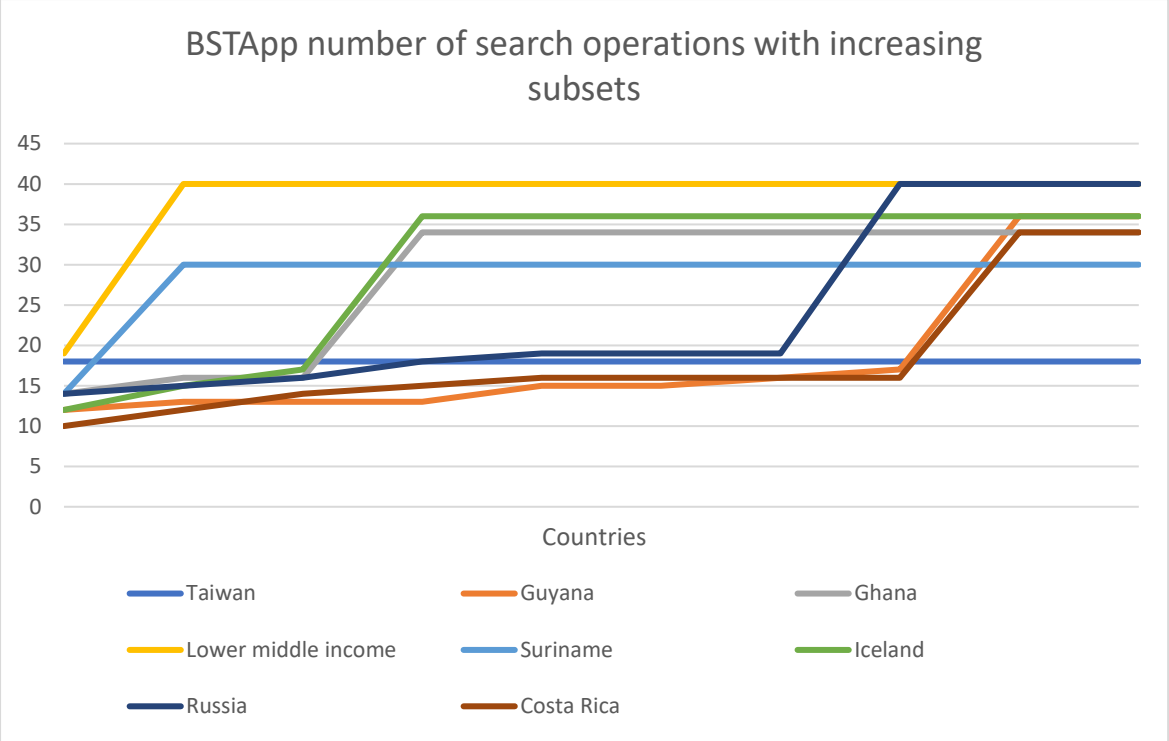
From the results above the following are observed. In VaccineArrayApp if the country is not found in a certain subset of data, the number of search operations are equal to the length of whole subset whereas in VaccineBSTApp the number search operations are far much less than the length of the subset. Furthermore, when the n values increase and the item being searched for *(date+country)* is not found, the number search operations increase with increasing dataset. In VaccineArrayApp the search operations increase with a bigger factor whereas changes in VaccineBSTApp are far less or remain the same.

The more items are there in the data structure, the more time it takes to check whether the item is found or not. This increases the number of search operations. When the item is found, the number of search operations remain the same for the rest of the subsets.

The data above showed in a graphical form:



VaccineArrayApp best case O(1) is represented by the country Taiwan, it takes a constant time to search for an item no matter the amount of data in the set. Worst case O(n) can be seen by inspecting countries like Costa Rica and Guyana, it takes an amount of linear time to search for an item with the size of the set for example, it took 1982 search operations to search Guyana in a subset of n = 1982 elements. Average case O(n/2) scenario is represented by countries like Iceland and Ghana, by inspecting VaccineArrayApp table Iceland at n = 6937 the number of search operations was 3481.



VaccineBSTApp best case O(1) is represented by Taiwan, it takes a constant time to search for an item no matter the amount of data in the set. Worst case scenario O(n) is represented by Lower middle income as seen above. Although the number search operations are not equal or far more less than the amount of data in the set Lower middle income has the highest search operations than any other country. Average case scenario O(log n) is represented by Russia as its line lies more in the middle than any other line.

With that all been stated clearly that the speed of searching thorough a binary search tree is much faster than that of a traditional array. Therefore, it is efficient to choose the binary search tree data structure over a traditional array.

Git usage:

0:commit bd989a660db8767bf939fb170eef31c26a6403aa

1:Author: karabo-debug <lehlohonolomosikili143@gmail.com>

2:Date: Mon Mar 7 21:22:12 2022 +0200

3:

4:Change the placement of opCount variable

5:

6:commit edcdee8bb4317806270ac3885d8dabb3b2b536cd

7:Author: karabo-debug <lehlohonolomosikili143@gmail.com>

8:Date: Sun Mar 6 09:13:42 2022 +0200

9:

...

11:

12:commit 435fffa1f2c245ef579badbe3d0a7614d70c7c9a

13:Author: karabo-debug <lehlohonolomosikili143@gmail.com>

14:Date: Fri Feb 25 17:01:33 2022 +0200

15:

16:first version