Dafny Binary Search Tutorial

Teaching you about algorithms using software specifications

Hassaan Malik

Winter 2018

Table Of Contents

How to install Dafny

Before starting to learning about Binary Search you need to install Dafny on your computer. There are two ways to run Dafny, one is to run it through the online IDE or install it on your computer. The online IDE is not reliable since the website goes down often, so it is best to install it on your computer.

How to install on Windows

To install on a windows computer follow the steps in the following link https://github.com/Microsoft/dafny/wiki/INSTALL and follow those steps.

How to install on Mac OS and Linux

To install on a Mac OS and Linux computer follow the steps in the following link https://github.com/Microsoft/dafny/blob/master/INSTALL.md and follow those steps. To fully execute your dafny code you will also need wine for Mac OS so download from the following link https://wiki.winehq.org/Download

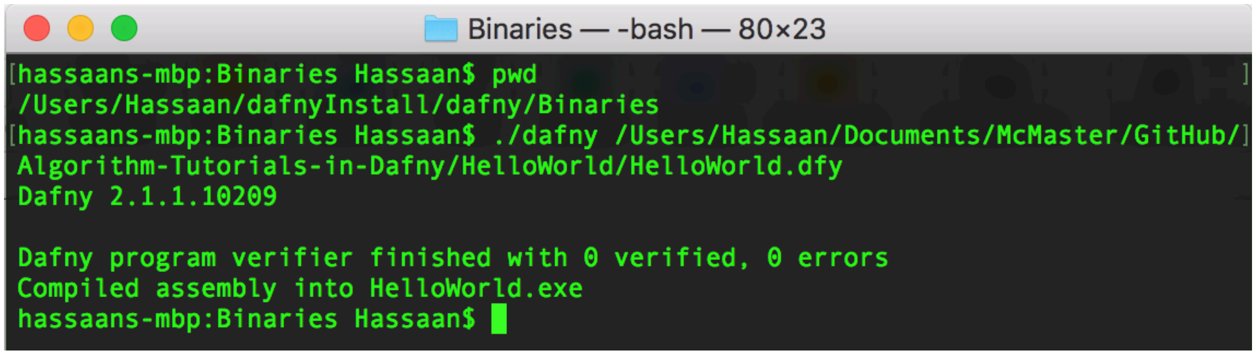
Running a program in Dafny

Lets create a hello world program in dafny and run the code as well. The code is available at my GitHub: https://github.com/Hassaanmalik/Algorithm-Tutorials-in-Dafny

```
method Main() {
  print "Hello World \n";
}
```

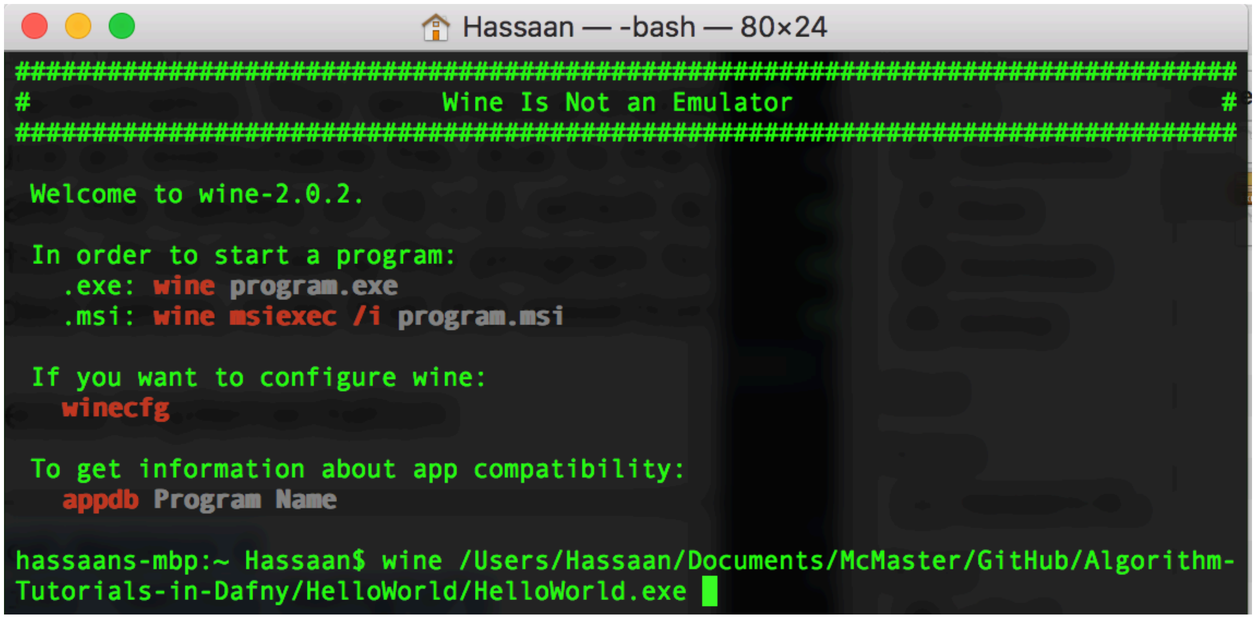
The code above will just print out the string “hello world” onto your screen. Save this code in a file call HelloWorld.dfy. To run your code now you need to navigate to the directory where you installed dafny and then the binary directory.

In the screenshot bellow it shows the directory you need to be in and how to compile the hello world code.



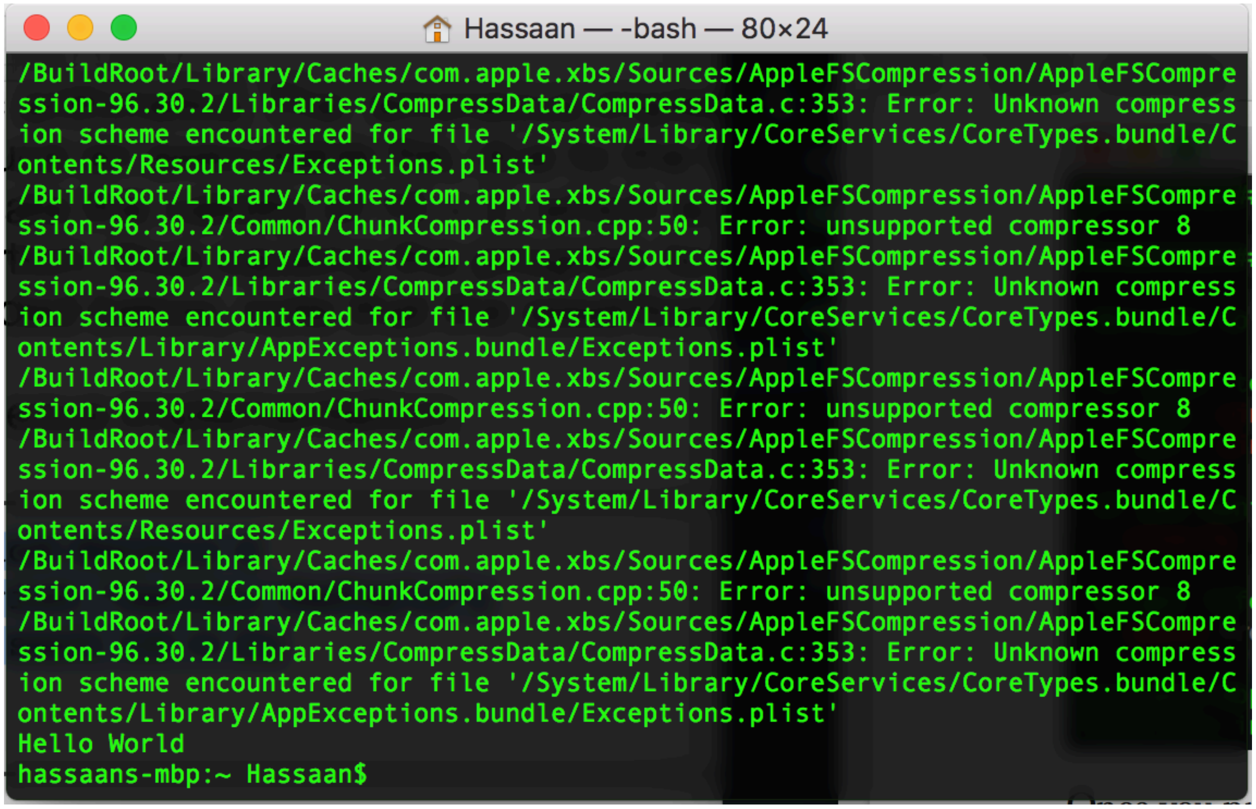
```
hassaans-mbp:Binaries Hassaan$ pwd
/Users/Hassaan/dafnyInstall/dafny/Binaries
hassaans-mbp:Binaries Hassaan$ ./dafny /Users/Hassaan/Documents/McMaster/GitHub/
Algorithm-Tutorials-in-Dafny/HelloWorld/HelloWorld.dfy
Dafny 2.1.1.10209

Dafny program verifier finished with 0 verified, 0 errors
Compiled assembly into HelloWorld.exe
hassaans-mbp:Binaries Hassaan$
```

Dafny complies all of your code into an executable file, these executable files can only be run by the windows operating system unless you have wine installed on any other OS. If you are on a windows computer just open the HelloWorld.exe file in your terminal. If you are on a mac or linux you will need to run this through wine stable. For that double click on the wine stable icon and follow the screenshot bellow.



```
################################################################################
#                          Wine Is Not an Emulator                             #
################################################################################

 Welcome to wine-2.0.2.

 In order to start a program:
   .exe: wine program.exe
   .msi: wine msiexec /i program.msi

 If you want to configure wine:
   winecfg

 To get information about app compatibility:
   appdb Program Name

hassaans-mbp:~ Hassaan$ wine /Users/Hassaan/Documents/McMaster/GitHub/Algorithm-
Tutorials-in-Dafny/HelloWorld/HelloWorld.exe
```

Once you press enter it may throw some warnings in wine. But at the end of the warning output it will display Hello World.



```
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Libraries/CompressData/CompressData.c:353: Error: Unknown compress
ion scheme encountered for file '/System/Library/CoreServices/CoreTypes.bundle/C
ontents/Resources/Exceptions.plist'
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Common/ChunkCompression.cpp:50: Error: unsupported compressor 8
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Libraries/CompressData/CompressData.c:353: Error: Unknown compress
ion scheme encountered for file '/System/Library/CoreServices/CoreTypes.bundle/C
ontents/Library/AppExceptions.bundle/Exceptions.plist'
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Common/ChunkCompression.cpp:50: Error: unsupported compressor 8
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Libraries/CompressData/CompressData.c:353: Error: Unknown compress
ion scheme encountered for file '/System/Library/CoreServices/CoreTypes.bundle/C
ontents/Resources/Exceptions.plist'
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Common/ChunkCompression.cpp:50: Error: unsupported compressor 8
/BuildRoot/Library/Caches/com.apple.xbs/Sources/AppleFSCompression/AppleFSCompre
ssion-96.30.2/Libraries/CompressData/CompressData.c:353: Error: Unknown compress
ion scheme encountered for file '/System/Library/CoreServices/CoreTypes.bundle/C
ontents/Library/AppExceptions.bundle/Exceptions.plist'
Hello World
hassaans-mbp:~ Hassaan$
```

Assertions in Dafny

An assertion in programming is a statement at some point in which a predicate is presumed to hold true. In dafny, if an assertion returns false then the program will throw an error stating the results of the assertion that don't allow the program to continue. We will use assertions in our algorithms to make sure that they will always take the correct inputs and always output the correct solutions.

This is an example of how we can use assertions

```
method Main() {
  print "Hello World \n”;
  assert 3<2;
}
```

The code above when compiled will fail. This is because the assertion that we set is incorrect. 3 is not less than 2 so the assertion is return false and the compiler will say there is 1 error in the code. To fix this issue we have to make that statement true and this can be done in any way we want and one way we can do this is by changing the less than symbol to the greater than symbol.

```
method Main() {
  print "Hello World \n”;
  assert 3>2;
}
```

This will now return true and the compiler will say there are 0 errors.

Loop Invariants in Dafny

A loop invariant is an expression that must hold true when in a loop and after every iteration through the loop as well. The reason we need loop invariants in dafny is because dafny does not know how many times to iterate through the loop. So invariants allow us to set conditions so the loop does not go through an infinite amount of times.

This is an example of how we can use invariants

```
method Main() {
  testFunction(4);
}
method testFunction(n:
int)
{
   var i := 0;
   while i < n
      invariant 0 <= i
   {
      i := i + 1;
   }
}
```

In the code above we are specifying that the function will take in an integer and run a while loop n times. We also have an invariant in there, the invariant says that the value of "i" must always be greater than or equal to 0. By setting this invariant any time we change the value of "i" we will always check that it is greater than or equal 0 even if the changed to something else later on. If we change the value of "i" to -1 the invariant will not hold true and compiler will throw an error.

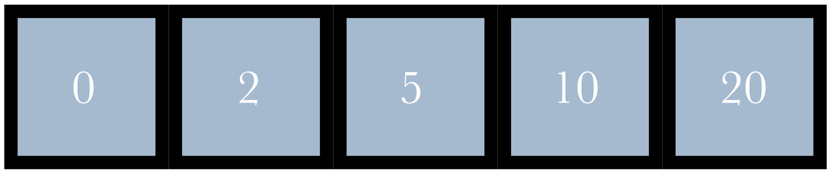
Binary Search

Binary search is an algorithm that searches through a sorted array repeatedly by dividing the index by half each time. It has a worst case performance time of O(log n) and a best case time of O(1). The average performance of this algorithm is O(log n).

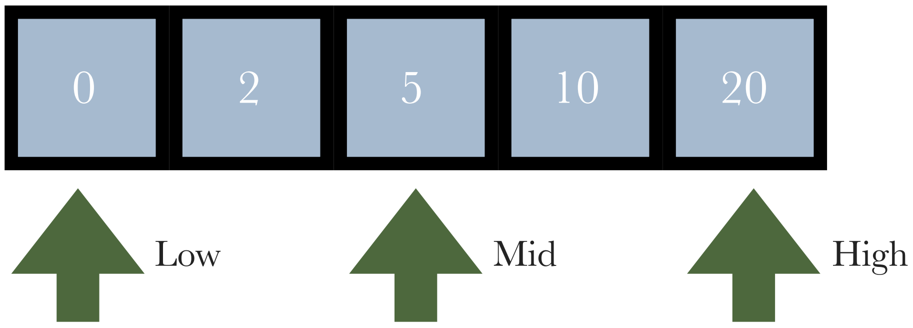
A little history lesson about Binary search, In 1946 a physicist by the name of John Machly first mentioned the binary search algorithm as part of the Moore School Lectures. These lecture are the first ever set of lectures regarding any computer-related topic.

Now Lets go through an example of how binary search works.

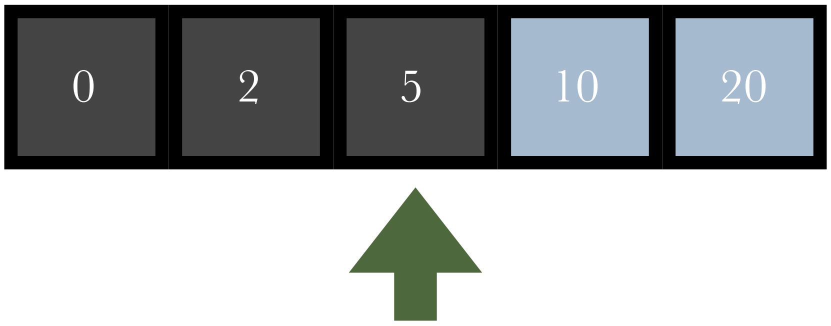
Here we have a sorted array, Binary search does not work if we do not have a sorted array so that is the first thing we need to check. Lets say we want to search for if 20 is in the array.



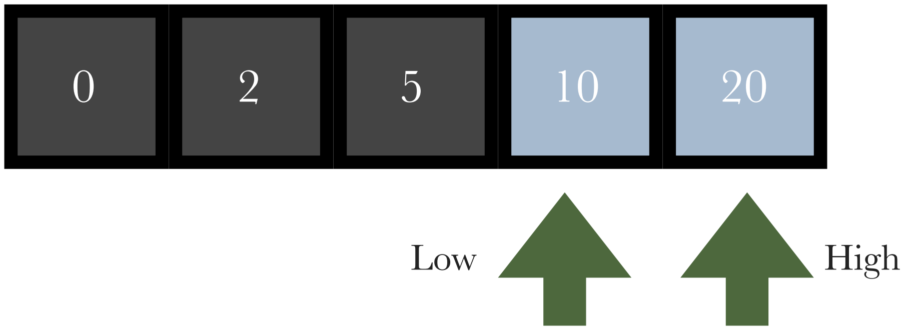
The first thing the algorithm will do is go to the middle element of the array and check if it is greater than, less than or equal to the value we are looking for which is 20. Middle element is calculated by taking the highest index and the lowest index and dividing by 2.

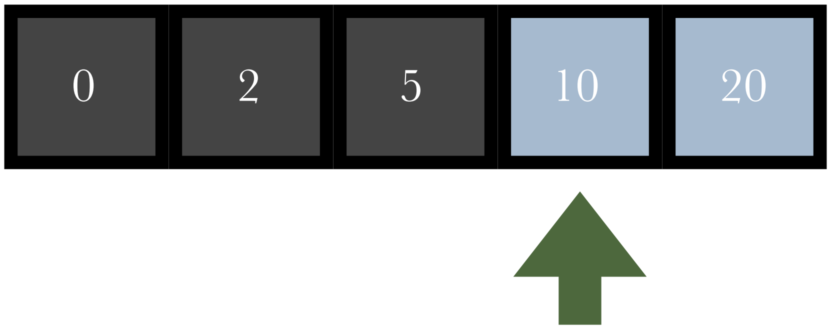


In this case since the value 20 is greater than the value 5 it will start by searching the right side of the array and ignore the whole left side.

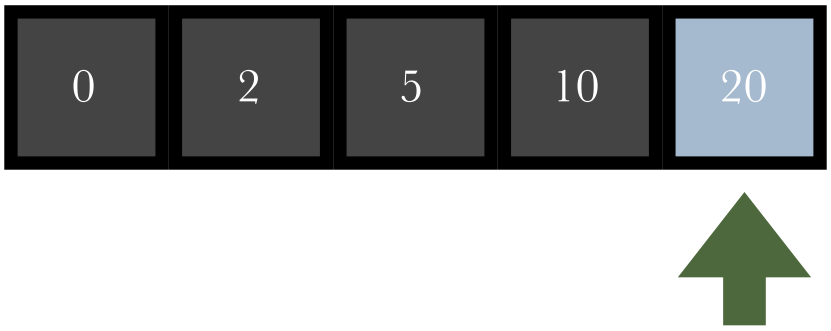


Now that we know which side we need to check we will once again add the highest index and the lowest index and divide them by 2. In our following case it would be index (3+4)/2 = 3 since we always take the lower bound.





We once again compare if the value at the index we are at is greater than, less than or equal to the value we want to find. In this case the low and high are the same value so we will be at the value 20 and the algorithm will return the index 4.



Binary Search Code in Dafny

The following binary search code is in dafny and is able to compiled without any errors.

```
method Main(){
  var a := new int[5];
  a[0], a[1], a[2], a[3], a[4] := 0,2,5,10,20;
  var value := 20;
  var index := BinarySearch(a, value);
  print "The value ",value, " was found at index
",index,"\n";
}

method BinarySearch(a: array<int>, value: int) returns
(index: int)
   requires a.Length >= 0
{
   var low, high := 0, a.Length;
   while low < high
      invariant 0 <= low <= high <= a.Length
   {
      var mid := (high + low) / 2;
      if a[mid] < value
      {
         low := mid + 1;
      }
      else if value < a[mid]
      {
         high := mid;
      }
      else
      {
         return mid;
      }
   }
   return -1;
}
```

The code we see above is incomplete, we are missing a few checks in there that you should see right away. But first we need to understand what requires. Requires is a precondition and must hold true for the function to continue, so in this case the array length has to be greater than or equal to 0 otherwise we could get an array out of bounds error. We also have an invariant in the code above as well, this invariant makes sure that the values are always in between the range of our array. So the low value must be less than equal to the high value which is less than equal to the length of the array. This allows the while loop to know what condition must hold true throughout the execution of the loop.

To fix the code above we need to remember the first thing that binary search needs to have in order to run and that is we need a sorted array. In the code above we do not check if the array is sorted. So lets write out a predicate function that is required when function binary search is ran.

```
predicate sorted(a: array<int>)
   reads a
{
   forall i, j :: 0 <= i < j < a.Length ==>
a[i] <= a[j]
}
```

Lets try and understand what this function is doing. The predicate function will always return true or false if the condition we give it is true or false. It then uses a quantifier to check the indices of the array a. This quantifier is just checking that the each value i is always preceding j and is within the bounds of the array, for each index i and j, the value of a[i] is less than or equal to a[j]. This quantifier makes sure that the values will be sorted. If they are not sorted then out binary search function should not run.

To make sure that the binary search does not run unless it is sorted we need to add a condition in the first requires clause.

```
requires a.Length >= 0 && sorted(a)
```

Exercise

We are still missing some conditions in the code to fully ensure that binary search will always output the correct value. In the code bellow some of the conditions are incorrect, all of them need to be their but are incorrect in some way. If you understand binary search works correctly you should be able to solve the errors. (Hint: try and compile the code, it will tell you which lines have the error)

```
method Main(){
  var a := new int[5];
  a[0], a[1], a[2], a[3], a[4] := 0,2,5,10,20;
  var value := 20;
  var index := BinarySearch(a, value);
  print "The value ",value, " was found at index ",index,"\n";
}

predicate sorted(a: array<int>)
   reads a
{
   forall i, j :: 0 <= i < j < a.Length ==> a[i] <= a[j]
}
method BinarySearch(a: array<int>, value: int) returns (index: int)
   requires a.Length >= 0 && sorted(a)
   ensures 0 <= index ==> index > a.Length && a[index] == value
   ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value
{
   var low, high := 0, a.Length;
   while low < high
      invariant 0 <= low <= high <= a.Length
      invariant forall i ::
         0 >= i > high && !(low >= i < high) ==> a[i] != value
   {
      var mid := (high + low) / 2;
      if a[mid] < value
      {
         low := mid + 1;
      }
      else if value < a[mid]
      {
         high := mid;
      }
      else
      {
         return mid;
      }
   }
   return -1;
}
```

Application of Binary Search

Binary search is an algorithm that has very general usages, since the algorithm has an O(log n) time complexly it is useful in most cases. It can be used anytime you want to find anything in a given sorted array. Another use of binary search is searching through a binary search tree, this example can be seen almost everywhere. For example in gaming, binary space partitioning is used to create a tree to define space in a game and then the searching algorithm is used to traverse through the tree to find what to display. This is one of the many applications of the algorithm.

Variations of the algorithm

Binary search is the base other variations of itself. One variation of the binary search implementation is that we specify the high and low value. This allows binary search to only search the area that we specify instead of the whole array. This is how the algorithm exponential search is done where it find the range where the number could be and then tries to find the value using binary search. Documentation for exponential search can be found on my GitHub as well.

Conclusion

After going through this tutorial, you should gain a good understand of how Binary Search works in dafny. Now you should be able to code binary search in any programming language as well as understand each aspect of it. This will allow you to succeed in interviews when they ask you about your knowledge of searching algorithms which will be used often in the software field.

References

Dafny: A Language and Program Verifier for Functional Correctness. (n.d.). Retrieved April 1, 2018, from https://www.microsoft.com/en-us/research/project/dafny-a-language-and-program-verifier-for-functional-correctness/

Binary Search. (2018, March 29). Retrieved April 1, 2018, from https://www.geeksforgeeks.org/binary-search/

(n.d.). Retrieved April 1, 2018, from https://www.topcoder.com/community/data-science/data-science-tutorials/binary-search/

The binary search algorithm in dafny was found on https://rise4fun.com/dafny/tutorialcontent/guide#h211

Getting Started with Dafny: A Guide. (n.d.). Retrieved April 1, 2018, from https://rise4fun.com/dafny/tutorialcontent/guide#h211