# Dafny Exponential 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 Exponential 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

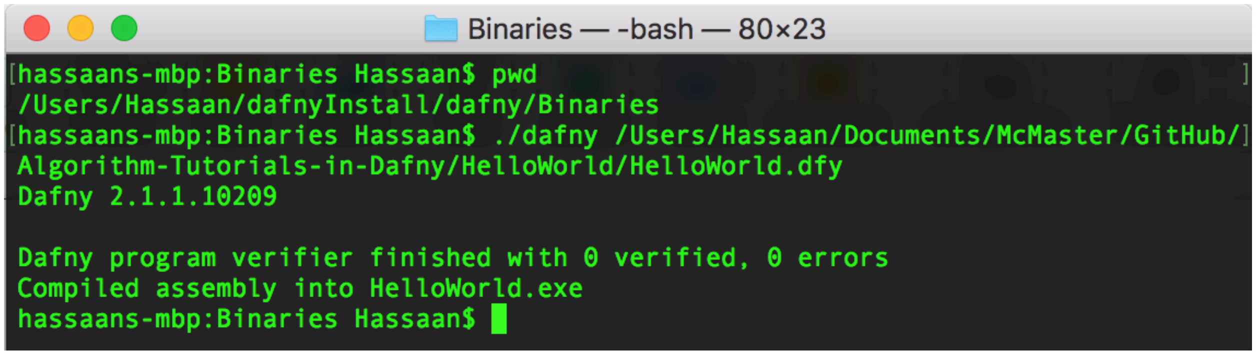
## Running a program in Dafny

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";
}
```

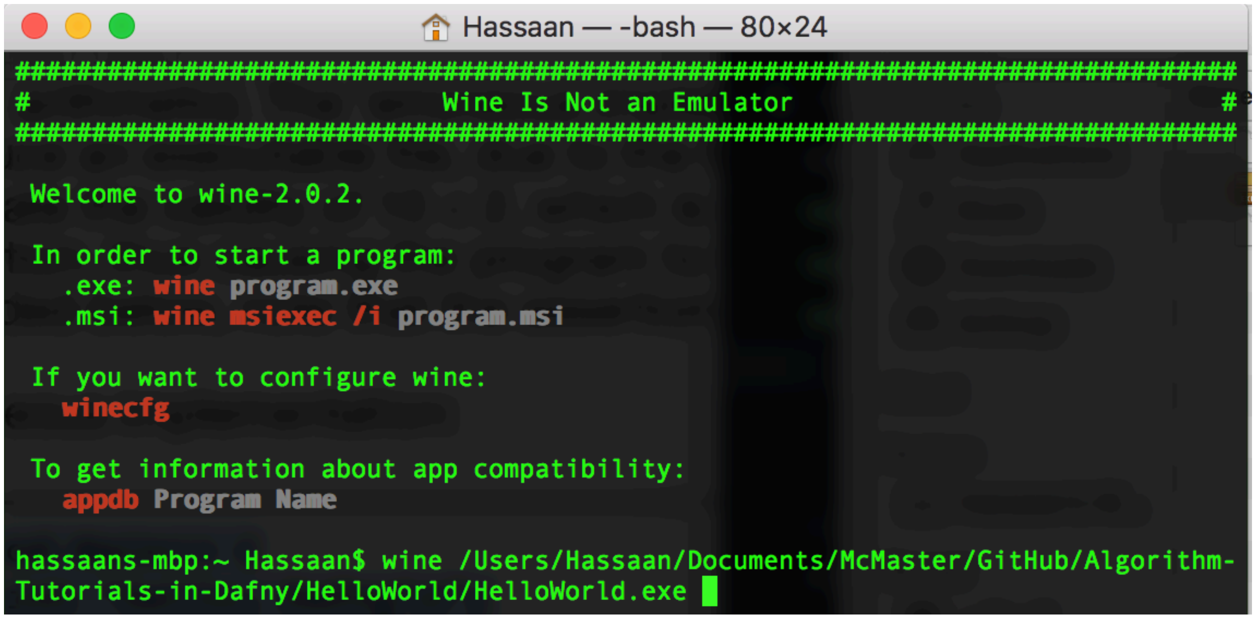
The code above will just print out the string "hello world" onto your screen. Save this code in a file call HelloWorld.dfy. To run your code now you need to navigate to the directory where you installed dafny and then the binary directory.

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



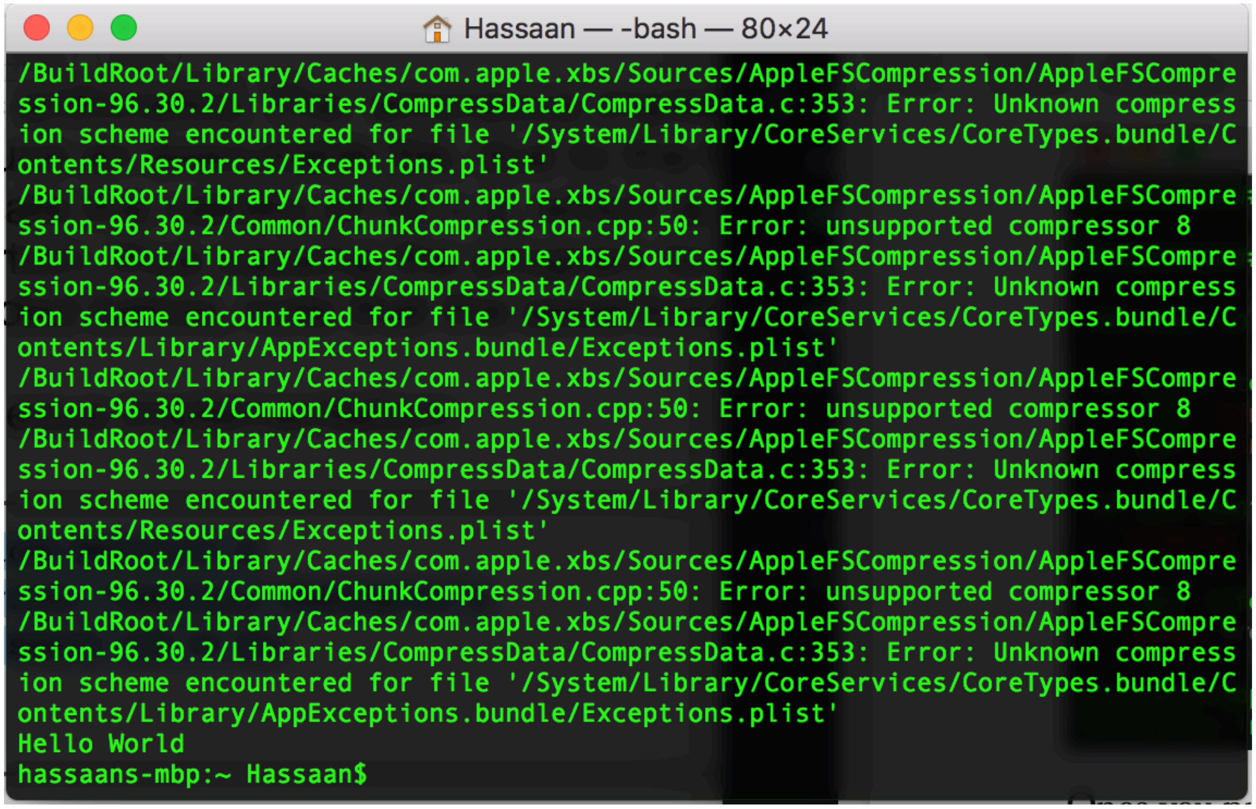
```
hassaans-mbp:Binaries Hassaan$ pwd
/Users/Hassaan/dafnyInstall/dafny/Binaries
hassaans-mbp:Binaries Hassaan$ ./dafny /Users/Hassaan/Documents/McMaster/GitHub/
Algorithm-Tutorials-in-Dafny/HelloWorld/HelloWorld.dfy
Dafny 2.1.1.10209

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.

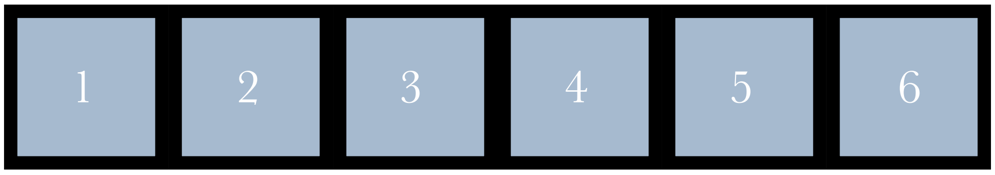
## Exponential Search

Exponential Search is an algorithm who's name is actually misleading as it does not search in exponential time but actually O(Log n) time. Exponential search have two main steps that it runs in, first it finds the range where the element is present. Second it runs binary search on the range that we find in step one and return the index where the value is found. Exponential Search has a best case of O(1) and the worst case of O(Log n).

A little history lesson about Exponential search, the algorithm was created by Jon Bentley and Andrew Chi-Chih Yao in 1976, they created this algorithm to search a sorted, unbounded infinite lists. This is seen in their book An almost optimal algorithm for unbounded searching.

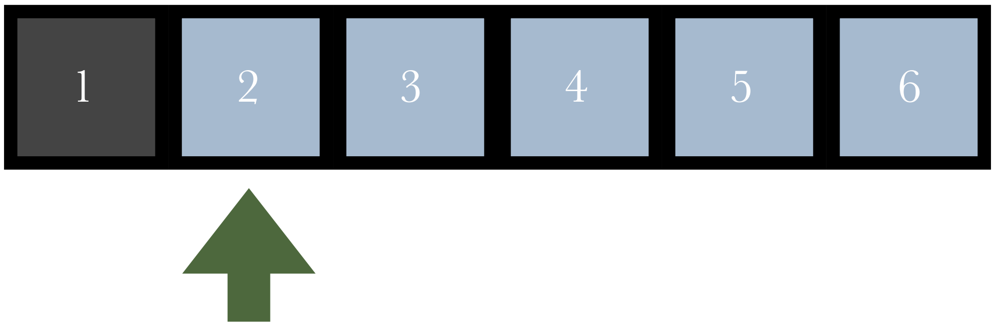
Lets go through an example of how exponential search works.

Here we have a sorted array,  Exponential 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 4 is in the array.

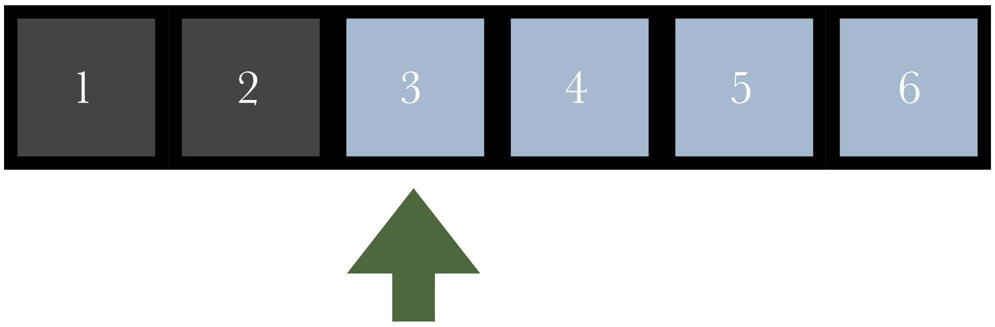


The first thing the algorithm will check is if the value we are trying to find is in the index 0. This is because we will be multiplying our index value by 2 every time and 0 would cause an issue with our calculations. If the value is in index 0 we return 0 otherwise we continue with the rest of the algorithm.

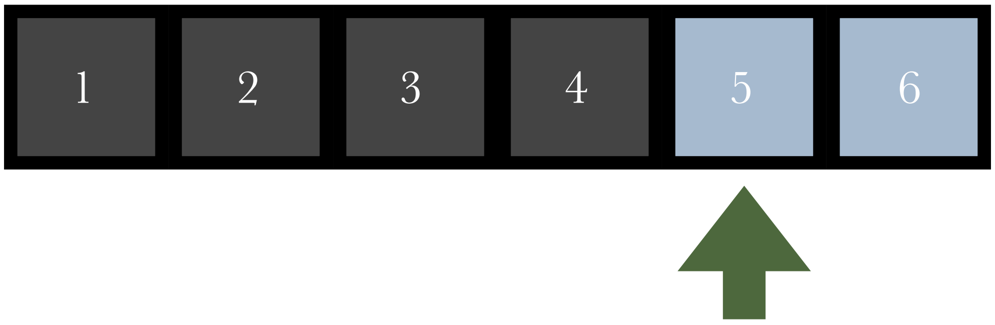
We not set the index value equal to 1. This index value i will be multiplied by 2 at each iteration of the loop and we will check if the value at that index in the array is less than or equal to the value we want to find. Now lets compare with our current array



We compare the value at index 1 and see that our value 4 and find that value at index 1 which is 2 is less than equal to 4. So we continue with the rest of the loop except now we go to index i*2 which is 2.

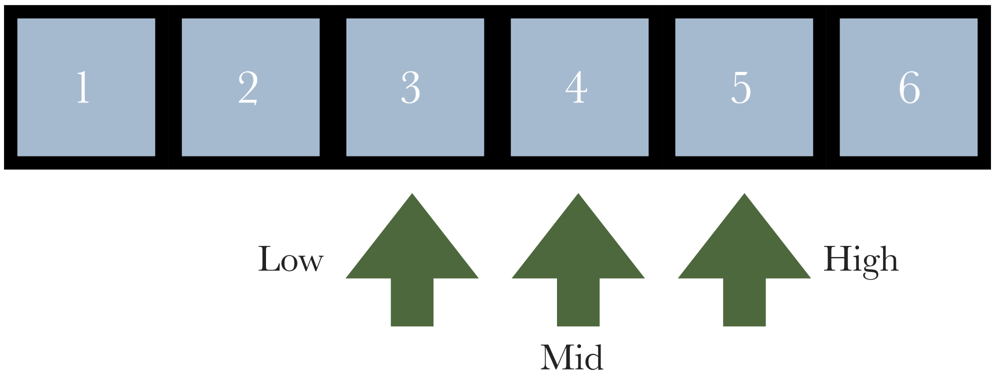


At index 2 we compare the value 3 with the value we want to find 4. Since the value 3 is still less than equal to our value so we continue with the loop. Now we go to the next index which is i*2 = 4



At index 4 we see that the value 5 is greater than our value 4. So now we can break out of our loop and call binary search. We are using a new binary search which takes in the low and high values. For the low value we give binary search i/2. For the high value we give the minimum value between i and the length of the array. In our case we would give the value of 2 as our minimum and for high we would give it the value i which is 4.

Now we begin binary search on our array which these high and low values.



In the diagram above we show the mid which calculated in binary search right away which is the middle of high and low. In this now that we know the value that we are looking for is at mid, we just return mid. So our algorithm would return the index of mid which is 3

## Exponential Search Code in Dafny

The following Exponential search code is in dafny

```
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 n := a.Length;
  var index := ExponentialSearch(a, n, value);
  print "The value ",value, " was found at index ",index,"\n";
}

method min(a: int, b: int) returns (d: int){
  if(a<b){
    return a;
  }
  else{
   return b;
  }
}

method ExponentialSearch(arr: array<int>, n: int, val: int) returns (index: int)
  ensures 0 <= index ==> index < arr.Length && arr[index] == val
  ensures index < 0 ==> forall k :: 0 <= k < arr.Length ==> arr[k] != val
{
  var temp := 0;
  if(arr[temp] == val){
    return 0;
  }

  var i := 1;
  while(1 < n && arr[i] <= val)
    invariant i <= arr.Length;
    invariant forall i :: 1 <= i < n && arr[i] <= val
    decreases n - i*2
  {
    i := i*2;
  }
  var minimum := min(i,n);
  var output := BinarySearch(arr, val, i/2, minimum);
  return output;
}

method BinarySearch(a: array<int>, value: int, x: int, y: int) returns (index: int)
   ensures 0 <= index ==> index < a.Length && a[index] == value
   ensures index < 0 ==> forall k :: 0 <= k < a.Length ==> a[k] != value
{
  var low, high := x, y;
   while low < high
      invariant 0 <= low <= high <= a.Length
      invariant forall i ::
         0 <= i < a.Length && !(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;
}
```

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 exponential 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 exponential 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 exponential search function should not run.

To make sure that the exponential 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)
```

Lastly we missing 2 requires in the new binary search. The new binary search takes in the range as well. By taking in the range exponential search is able to give binary search a range to only search in, instead of searching the whole array. Because we pass in the values x and y which are going to represent the high and low in binary search we need to require that they both meet certain criteria's.

```
requires 0 <= x <= y <= a.Length

requires forall i :: 0 <= i < a.Length && !(x <= i <
y) ==> a[i] != value
```

The above requires need to follow the binary search method. The first ones makes sure that the x and y values are in between the indexes of the array. And the second one

## Exercise

We are still missing some conditions in the code to fully ensure that exponential 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 exponential 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) The code is too long, please take a look at the file expenentialSearch(incorrect).dfy on GitHub.

## Application of Exponential Search

Exponential search is an algorithm that has two main cases where it is better to use it over traditional binary search. The first case being when it needs to search for unbounded searches where the size of the array is very large. The second case being that it also works better for bounded arrays, where the element that we want to find is closer to the beginning of the array.

## Conclusion

After going through this tutorial, you should gain a good understand of how Exponential Search works in dafny. Now you should be able to code exponential 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/

Exponential Search. (2018, April 19). Retrieved April 15, 2018, from https://www.geeksforgeeks.org/exponential-search/

Exponential search. (2018, February 02). Retrieved April 15, 2018, from http://www.techiedelight.com/exponential-search/

An almost optimal algorithm for unbounded searching. (2002, October 03). Retrieved April 15, 2018, from https://www.sciencedirect.com/science/article/pii/0020019076900715?via=ihub