Chapter 1

Algorithm:

Set of instructions to accomplish a certain task.

Any piece of code could be called an algorithm no matter how simple it is the most important thing about an algorithm is how efficient it is in terms of time and storage (the less time and storage the better)

In this chapter you will be introduced to two search algorithms and understand how one of them is better than the other. After that you will be introduced to big O notation used in comparing speed of different algorithms.

Search:

Let's say you have a book and you want to go to page number 100 the most simple thing to do is that you turn the pages one by one till you reach page number 100 most of you will think this is a very boring thing to do and actually it really is the most common thing to do is that you will grab some pages that you think are nearly 100 (they could be more or less) and then you flip the few remaining pages(you may be lucky enough to get to 100 directly) this is a problem that will confront you a lot during programming you will have a list or an array (explained thoroughly in chapter 2) and you want to search for something in that list let's say you have a list containing numbers sorted from 1 to 100 (the two algorithms can help you search for any thing we are using numbers to make it simple) and you want to find the position of number 54 you can use the following algorithms.

Linear search:

This algorithm follows the simple way of searching it will go through the list one by one till it find the number you are looking for if it exist so in our example it'll go for the first position containing the number 1 and ask is it equal to 54 definitely the answer is no so we go to the next position and repeat and so on till we either find the number so we return it's position or it's not found in our case we will repeat the process 54 times till we reach our target and if we are searching for the last number which is 100 it will take us hundred times.

Binary search:

Remember the common way we mentioned first where you grab some pages and flip the remaining pages binary search do something that is very similar to it but in more organized way to ensure that the algorithm always succeed in finding the element if it exists so what we are going to do is to split our list into two halves in our case the middle of the list is 50 and we ask the question is the number I'm searching for greater or less than 50 so 54 is greater than 50 then the first 50 elements are thrown away now our list goes from 51 to 100 so we are going to split it again and this time the middles is 75 and ask again is 54 less or greater than 75 the answer is less so the elements from 75 to100 are thrown away now our list contains elements from 51 to 74 so we split the list again and again till we reach the number that we want this method saves a lot of time specially when the list is too big in our case we'll find 54 after 6 tries that's a huge improvement compared to linear search so how do we know if the number doesn't exist if you noticed the manner by which the start and the end are changing we either make the beginning = middle +1 or the end= middle -1 so if the element you are searching for doesn't exist eventually the start will be greater than the end at this point you know that the element doesn't exist.

Is binary search always the solution?

The answer to this question is no but why no that's what are going to say first of all most of you may have thought that the problem with number is to trivial to even search if I had 100 sorted numbers so number 54 is going to be in 54th position(actually it's the 53rd but we'll dive into that when we speak about the arrays in the next chapter) but the example I used is only for illustration so I made them simple you'll need search if the list is increasing in other manner than the default (and I don't know that manner) the list could go like that {2, 4, 6, 8, 10} or {1,2,10,15,16,17} , etc.. in that case we need search algorithms and binary search will be better but keep in mind that binary search is only helpful when the list is sorted also binary search and linear search could work with any data type numbers , characters, words ,etc…. the only condition to keep in mind is when you using binary search is that the list must be sorted a list like this {1,4,2,3,100,56, 7} can't be searched using binary search you'll have to use linear search.

Points of comparison	Binary search	Linear search
Speed	Faster	Slower
Data type of element	Any data type	Any data type
Order of the list	Must be ordered	Doesn't matter if it's ordered or not

Big O notation:

You might've noticed me talking about speed and time efficiency and I just said that binary search is faster than linear search so how is this speed really measured and how can we say that one algorithm is faster than the other that what the big O notation is for (it's called that because you put big O in front of the number of operations you will do) it's easier way for programmers to communicate and it make it more simple and clear when we are comparing algorithms .

So how do we use it? Let us say you are using linear search to search through a list of one hundred elements. What is the worst-case scenario? It is when the element searching for is at the end of the list which means you will do 100 operations till you reach the element you are looking for what if the list contains 1000 elements? you will do 1000 operations and so on no matter if you increase or decrease the number of elements the number of operations will always be equal to the number of elements in the list at worst you may think what if the element is in the middle or may be even the begging those are called best and average cases the big O is only concerned about the worst case I need to what is the worst thing that could happen and se if it suits me or not.

As you noticed the number of operations in linear search is always equal to number of elements so if n= number of operations then the complexity of linear search is O(n) which is called linear time (that's why it's called linear search) so what does O(n) means it means no matter what happens the biggest number of operations I'll take is equal to the number of elements I'm dealing with.

So now how can we calculate the big O for binary search if the list contains 4 elements it will take 2 operations and any number of elements between 4 and 8(more than 4 and less than 8) will take 3 operations and between 8 and 16 elements it takes 4 operations doesn't that ring a bill these are logarithms the big O for Binary search is big O (log(n))(the base is always 2 when we talk about logs in this book) so if we had 4 billion elements in our list the binary search will need only 32 operations to find your element at worst that's how convenient the binary search is compared to linear search.

Most common run times:

Fastest **O (log (n)), O (n), O (n log(n)), O (n^2), O (n!)** slowest

Example for an algorithm that takes n! is

Travelling salesman problem