Computer algorithms searching & sorting:

-: Are two most common types of problems that we try to solve.

Let’s say you have list of numbers as shown below and from this list you want to search number for example 32 ??

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | 9 | 11 | 17 | 21 | 25 | 29 | 32 | 38 |

Search for 32 --🡪

As we started before that indexes from the left side begins with ZERO index position in order to find the number itself the known method is we will count

Element by element to find it and we will approach and see the number in index number (7)

So it is quite not large value to find it in the ordinary way but like for example

Electronics store has more than 30000 transactions it Is quite difficult to make it in the normal way we used to it so every transaction has device id and amount it costs like below:

Name: rana

Device\_id: 12ab56

Amount: 500$

Device id you want to know whom that device was sold to perform a linear search

You will write this kind of code in python where your are just iterating over all the transactions in the transaction list and when you find the device id : 12ab56 you will return that index so the main issue what will be the time complexity of this code

If element [ “device\_id”] == “12ab56”

Return index

Return -1

So it is like order of n because you are doing n number iterations so this is called a linear search so it is not very efficient so if you have one million transactions it is pretty way too much for a computer to handle = performing iterations one million times.

So we will get back to our numbers example linear search took 7 iterations to find number 32 … thinking about better method less than 7

The list is sorted and the binary search works on the concept of sorted list so in the sorted list how can we find 32 so we do not have to do 7 iterations so the

Technique is we will take the MIDDLE element which is 21 and comparing it with 32 so of course it is less than 32

Now by doing this comparison we ruled out the possibility the number exist in the left side of the array so all the numbers from 4 to 21 so the number is not in the particular portion of the array so we can ignore it so the first one you discarded half of the elements then you will discard the left side of the array which is

25 &29 the remaining is two elements 32 & 38 so when you take the middle

So now we can do three iterations so this is called BINARY SEARCH.

|  |
| --- |
| So in the binary search we are diving search space by have |

The iteration one we discarded n by two elements

Iteration two It is two raised to two as we are doing n by 2 divided by 2

The third iteration it will be n divided by 3 divided by 2

In a simple meaning the binary search is order of log n

so if we do it in python for example we have list of number

numbers\_list = [12,15, 17 , 19 , 21 , 24 , 45 , 67]

number to find = 45

first we will find it with linear search is just running a simple for loop on these numbers in PYTHON we will write

for index, element in enumerate(numbers\_list):

if element == number\_to\_find:

return index ( or if not found )

return -1

so if we run it we will find it at index 6 using LINEAR SEARCH and if you write any other number it will give you -1

we will do the same thin using binary search:

left\_index = 0

right\_index = len(numers\_list) -1

mid index = 0

while left\_index <= right\_index:

mid\_index = (left\_index + right index) // 2

mid numer = numbers\_list[mid\_index]

if mid\_number == number\_to\_find:

return mid\_index

if mid\_number < number\_to\_find:

left\_index = mid\_index -1

else:

right\_index = mid index -1

return -1

|  |
| --- |
| This search algorithm takes advantage of a collection of elements that is already sorted by ignoring half of the elements after just one comparison. |