Chapter 4

In this chapter we will dive more into algorithms while learning recursion in the previous chapter and we said that its main advantage is dividing your problem into simpler problems which is the core of the new skill we are going to focus on in this chapter "divided and conquer"

Divide & conquer:

The main concept of this method is that you try to reach the base case of your problem which according to Euclidean algorithm will work for the bigger problem you had so in other words you are working towards a simpler version of your problem so lets get deeper let's say you are having a farm of dimensions 1680*640 and you want to divide it following some constraints all constraints must be satisfied 1-all the pieces must be squares 2- all of them must be equal in area 3- it must be the largest square possible this might seem tricky at first but lets think whats the easiest case its when one of the sides is a multiple of the other (GCD (x ,y)=x or GCD(x ,y)=y) if we try to divide our farm using the smallest side we will get two farms of dimensions 640*640 and one farm and one farm of dimensions 640*400 this is a simpler version lets try working with it if we try to divide this farm we will get one 400*400 farm and one 400*240 farm again we will work with this smaller version we will get a smaller version which is 240*160 we still have no farm with no side as a multiple of the other so we divide again we get 160*80 farm and finally here we are this farm will be divided into two 80*80 squares so for our original farm the largest square to divide our form into to satisfy all the constraints is an 80*80.

Note: the main theorem of Euclidean algorithm is that the if a = b * q + r then GCD (a, b) = GCD (b, r) you can find further explanation in the link above as it would take to long to explain Euclidean algorithm in a summary.

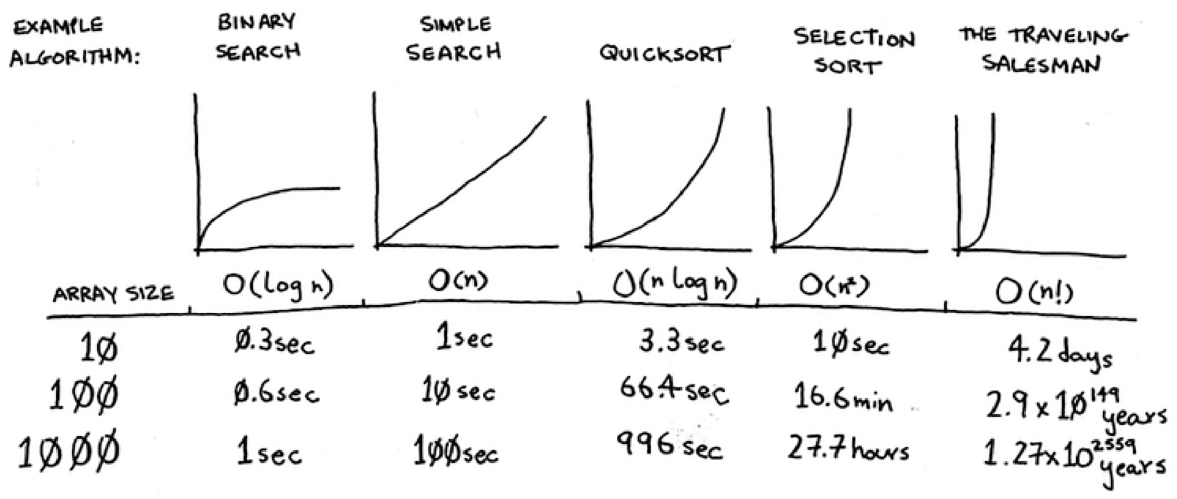
Quicksort:

Quicksort is a sorting algorithm that is much faster much faster than selection sort and is frequently used in real life.

The main advantage of quicksort is that it makes used of divide and conquer which mean we have to think of a base case for sorting arrays that's a bit easy for an array there is a case where you don't have to even sort the array it's when the array only has only one element (there's nothing to sort) let's try an example this here is our array {33,,15,10} you are using D&C so you have to break down this array till you reach the base case so in quicksort you first you pick an element from the array we call it the pivot (any element could be a pivot but some pivots will help you sort the array faster for now we will just choose the first element) now we will make to arrays one containing all the elements smaller than the pivot and one for all elements greater than the pivot so now you have a pivot sub array of elements greater than the pivot and sub array for elements larger than the pivot {10,15}[3]{} (the other sub array is empty because the pivot is the biggest number in the array so what's next quick sort knows how to sort arrays of 2 elements or less right away so for our case you now have to combine the two sub arrays and the pivot so you will do something like that quicksort({15,10})+ [33]+quicksort({}) and since you are already at the base case the function will return a sorted array let's try another example to clear things up {33,10,15,7} this array will be partitioned as follows {10,15,7}[33]{} the right sub array is already at base case the left one needs to be partitioned and it will be as follows {7}[10][15] and here you hit the base case the smaller array is now sorted and you just need to add the original pivot to it and the same will go on if you can sort an array of three elements then you can sort an array of four elements and you can sort one of five elements and so on this is called inductive proof so this raps up our talk about the quick sort and finally we go to a topic you are quite familiar with.

Big O notation:

most common big O run times



EXAMPLE ALGORITHM:	BINARY SEARCH	SIMPLE SEARCH	QUICKSORT	SELECTION SORT	THE TRAVELING SALESMAN
ARRAY SIZE	$O(\log n)$	$O(n)$	$O(n \log n)$	$O(n^2)$	$O(n!)$
10	0.3sec	1sec	3.3sec	10sec	4.2 days
100	0.6sec	10 sec	66.4sec	16.6min	2.9×10^{149} years
1000	1sec	100sec	996 sec	27.7hours	1.27×10^{2559} years

Estimates based on a slow computer that performs 10 operations per second

You may notice that quicksort is O (n log n) but actually it isn't that straight forward first you need to know that there is another sorting algorithm called merge sort that takes O(n log n) and actually quicksort is O(n^2) in worst case but O(n log n) on average case so what does that mean.

Merge sort vs. quicksort:

First let's make something clear big O notations aren't always that accurate some algorithms have the same big O run times but in reality one maybe faster than the other let's say we have a function that loops over an array and print it's element the big O complexity of this function will be O(n) is we have the same function but we added a line that makes the function sleeps one second between iterations it's big O complexity will still remain O(n) but actually it is much slower than the first implementation actually in the big) notation there is a constant c that is multiplied by n so in reality it’s O(c*n) but we usually ignore that constant because if the two big O run times are different this constant can be negligible like between binary search and linear search binary search is much faster that even if there is a constant multiplied by n binary search will still be faster than linear search but sometimes this constant can make a difference.

We previously mentioned average case and worst case we all know what is worst case but what is average case let’s take average case as an example while choosing the pivot we choose the first element which lift us with a completely empty sub array this isn’t really a good practice but if you chose a good pivot which is the middle of the array that will help you reduce the run time to O(n log n) which is the average case the good thing about the quick sort is that you can frequently hit the average case .

Now that you understand the two concepts above why do we prefer quicksort over merge sort first like I just said you can hit the average case more frequently and second because it’s one of the cases where the constant multiplied by n could make a difference so that’s why the quicksort is one of the fastest sorting algorithms and that’s all for this chapter.