Searching and sorting💛

Searching for information is very common in writing programs

Efficiency of algorithms is very important
If you're gonna use find - it uses a linear search but think about efficiency (use binary)

Importan slide(Big o (1) etc)

Linear Search

Unsorted data for linear search
If its unsorted you cannot use binary

The easiest searching algorithm but not the best

- Start from the beginning of the list until we find the value that we looking for or we determine that we've reached the end of the list and the number is not present in the list

- Loop to look at each value in the list and see if its the one we looking for, if it is we return the index positioning the list of the value
- If its not in the list we return a default value which is -1

Binary Search

- Linear search looks one at a time looking for a value (its not efficient)
- If you have a 1000 elements and the value you're looking for is not in the list, you're gonna search through each value in the list until you reach the end

A better way to do this:

Binary search

Efficiency is lögn = 10

Uses divide and conquer(divides list into half and looks for the value)

- For Binary Search the list needs to be sorted
- Determine which half contains search item

- Recursively search that half
- Stop when item is found or list is empty

If value is found index is returned

Each execution of the recursive method reduces the search space by about a half

- The binary search is extremely fast
- About half the array is eliminated from the consideration right at the start, then a quarter of the array, then an eighth etc

Comparing Algorithms

- We want a machine-independent estimate formula of how long a program will take to run
- Given a particular size of a List, how long will the program take to run on average(some sort of comparison)

We count the number of operations

- Estimates are usually expressed in big-O notation

Upper bound estimate(<u>not an exact count</u> but correct to a constant multiple)

Only include the term with highest exponent and dont pay attention. To constant multiples.
E.g : O(N^2 +3N) => O(N^2)

We measure efficiency of searching algorithms in terms of the upper bound of number of comparisons for n items

Constant or log n are better algorithms s you're making *fewer comparisons*

<u>When comparing algorithms:</u>
Best Case
Worst case
Average case

<u>Linear search</u>
Best case if the number is in the first
Worst case if the number is not in there
Average case: number of comparisons = n/2;O(n)

Binary search
Best case: its O(1)(in the middle)
Average case:O(log2n)
Worst case: O(log2n)

linear search and binary you're searching for a list
In a dictionary it gives you the name etc right away
Dictionaries are more efficient than lists

Sorting

Selection sort

For each value you make n-1 comparisons (10 then 9 then 8 then 7 etc)
Efficiency is **n^2**
Its the worst of them all

How the cpu works - Von Neuman architecture
He's also famous for the merge sort

Start with the smallest item in list and place at beginning then swap

Selection sorts within the list, and at the end we'll have the sorted list it doesnt create a new sorted list

Merge sort

How does a merge sort work (exam question)
In exam show how it splits up for all marks

- Uses divide and conquer
- Selection sort is good for shorter lists

- Most efficient sorting algorithms all follow the divide and conquer strategy.
- It is naturally recursive

- Characterised by dividing problems into sub-problems that are of the same form as the larger problem
- Problems are solved independently and then merged I to a solution for

the whole program.

Sorts each list recursively and merge the 2 sorted lists into a final solution

It was on elf the first methods proposed for computer sorting

Take 2 elements and join them but when we join them we compare!

With a merge sort it creates a new sorted list so now you have 2 lists, the unsorted one and the sorted list

Best case - sorted already
Worst case - sorted in reverse
Average case - some are sorted, some aren't(n log n)

Big 0 notation is an upper bound notation
n = 1000
In one check we have the value(order 1)
Order log n is more than 1 but unto 10
nlogn -If its anywhere between 11 and 100 (e.g average comparisons is 20 :efficiency is n)
N^2 - 1000^2

First value in list(linear search is 1st number and binary search its the middle number)