Linear Growth

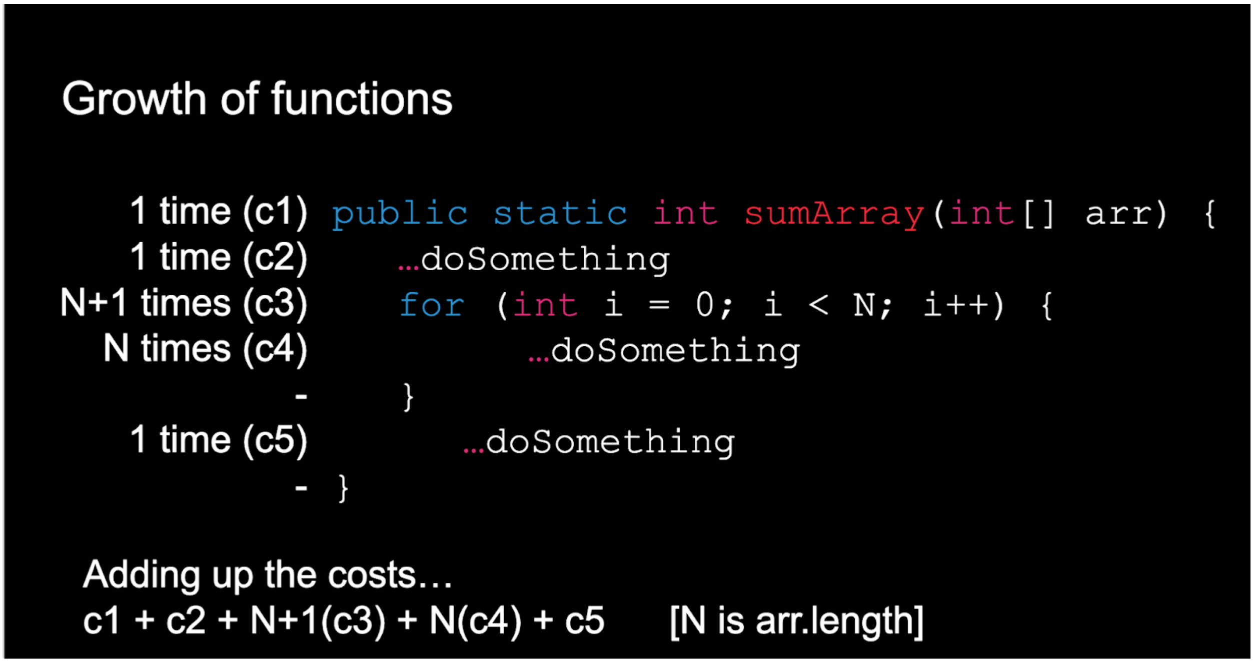
Say the function has an arbitrary cost c1, whenever its called? Whatevers under as long as it isnt a for loop happens 1 time again lets say c2
Say the for loop is executed n+1 times,

- It is n+1 and not just n b/c it has to check the condition? C3 the condition is always checked until the end , lets say its > , it checks it until the statement is false
- The thing inside the for loop happens n time lets say c4
    - N+1 for for loop
    - N for whats inside for loop

Then the thing after is 1 times say c5
Add it up

- We can see it its ax+b form , a polynomial, this is linear specifically
    - Therefore this function has a linear order of growth

- Programs can be measured two diff ways
    - Time and space
    - How fast it runs and how many resources it uses



Growth of functions

```
   1 time (c1) public static int sumArray(int[] arr) {
   1 time (c2)     …doSomething
N+1 times (c3)     for (int i = 0; i < N; i++) {
  N times (c4)          …doSomething
             -     }
   1 time (c5)        …doSomething
             - }
```

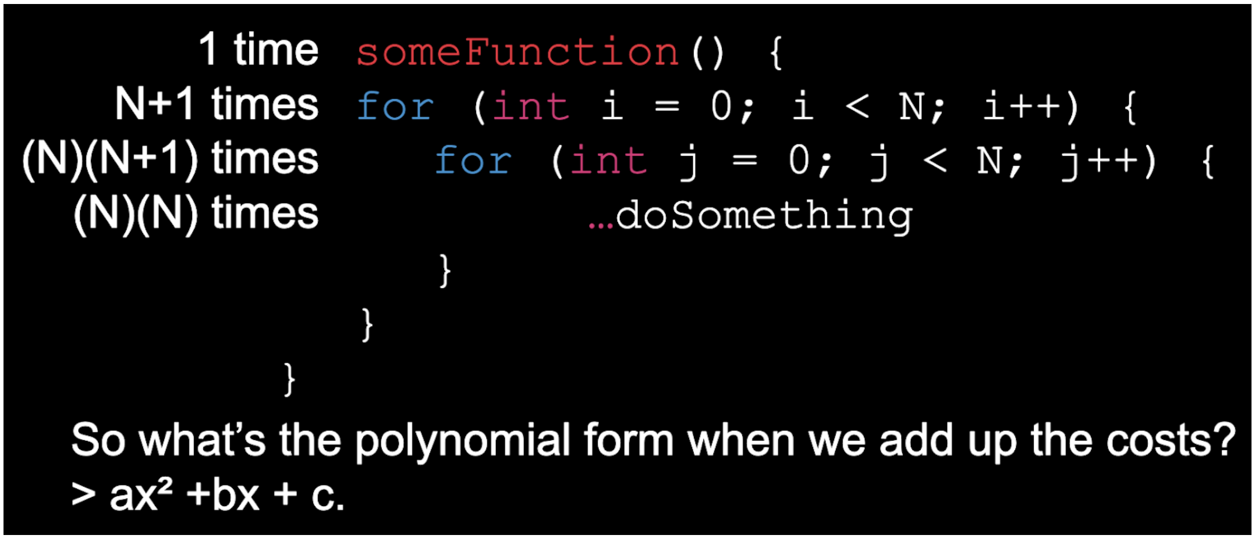
Adding up the costs…
c1 + c2 + N+1(c3) + N(c4) + c5   [N is arr.length]

Any such function of a for loop, (like image on slide)

- We can say it's *usually* **linear**

Quadratic Growth

Now a diff example
Two for loops, i and j goes to end, we only do something inside the nested for loop



```
       1 time   someFunction() {
    N+1 times   for (int i = 0; i < N; i++) {
(N)(N+1) times      for (int j = 0; j < N; j++) {
   (N)(N) times             …doSomething
                    }
                }
            }
```

So what's the polynomial form when we add up the costs?
> $ax^2 +bx + c$.

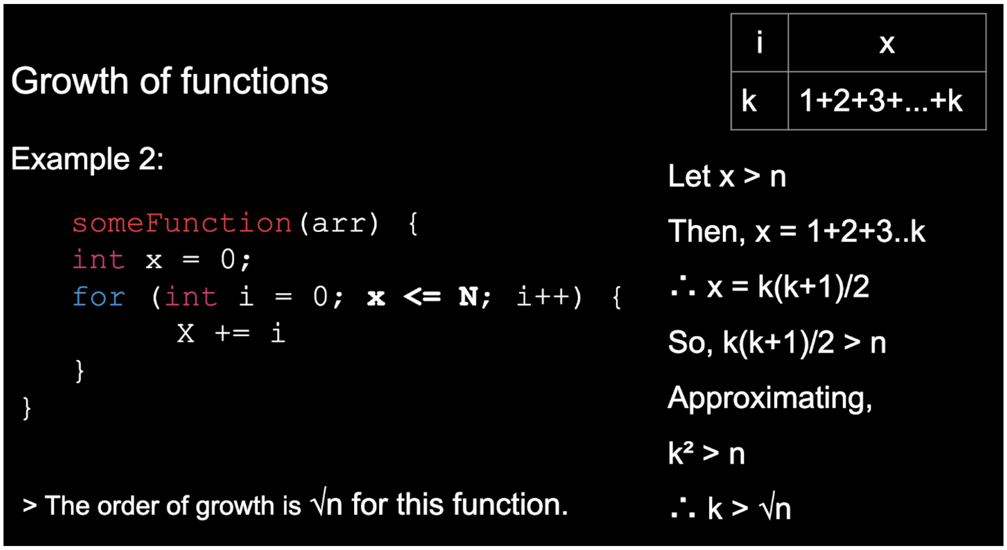
- Anything inside a for loop is n times, but its in another for loop so its n*n+1
- Outside for loop still n+1
- And the thing inside for loop is n*n times
- In polynomial form this is quadratic formula, the order of growth will be quadratic
    - There are some exceptions to this, but when we see this it will usually be quadratic growth

Whenever we add a loop it seems like a factor is added, **not all the time though**

Example1

- Two for loops but not nested
    - Still linear order of growth (n)?
    - >ax+b?

Example2



Growth of functions

Example 2:

```
    someFunction(arr) {
    int x = 0;
    for (int i = 0; x <= N; i++) {
           X += i
    }
}
```

> The order of growth is √n for this function.

i	x
k	1+2+3+...+k

Let x > n

Then, x = 1+2+3..k

∴ $x = k(k+1)/2$

So, $k(k+1)/2 > n$

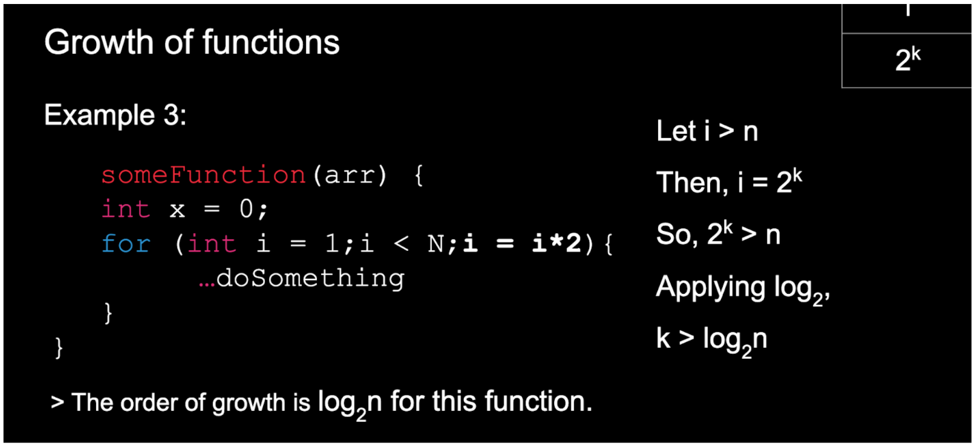
Approximating,

$k^2 > n$

∴ $k > \sqrt{n}$

- For loop but x **<=** n
    - Think abt the last value of i,x
    - Look at the table and how it increments, 1+2+3…+k
        - Say at k , x > n to terminate (terminal condition for loop to end)
        - Then x is the value of 1+2+3..+k
            - The formula for this is k(k+1)/2 (calc formula for sigma n?)
                - So k(k+1)/2 is > n
                - By apporx. k^2 > n
                - So k >sqrt. N
                - The order of growth for this function is sqrt n
    - Once the statement changes, the order of growth changes, so dont jump to conclusions

Example3



Growth of functions

Example 3:

```
    someFunction(arr) {
    int x = 0;
    for (int i = 1;i < N;i = i*2){
         …doSomething
    }
 }
```

> The order of growth is $\log_2 n$ for this function.

2^k

Let i > n

Then, $i = 2^k$

So, $2^k > n$

Applying $\log_2$,

$k > \log_2 n$

- For loop, but ;i<n **;i = i*2**
    - Think about the final value of i
    - 1,2,4,...,2^k

- Final value is 2^k , let i > n which is the terminal condition of the for loop,(we are figuring out how much times the loop runs)
    - I = 2^k , so 2^k > n
    - Applying logbase 2
    - K > log base 2 n
    - So the growth of this function is logarithmic , log base 2 n