3. Algorithms

2023-11-20

Section materials: jrsacher.github.io/cs50/

Upcoming (U.S.) holidays

- Thursday, November 23 (section) – Thanksgiving
- Monday, December 25 (office hours) – Christmas
- Monday, January 1 (section) – New Year's Day

Agenda

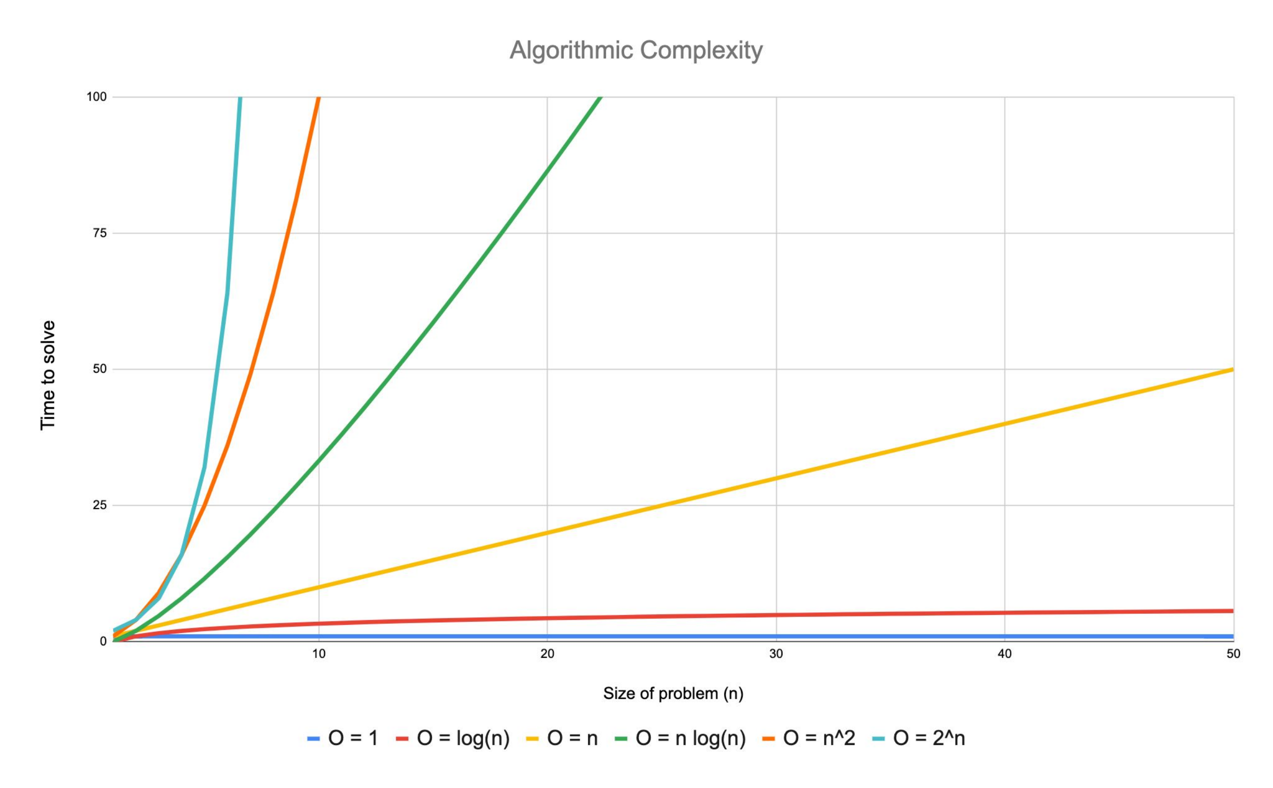
- Big *O* notation
- Searching
    - Linear search
    - Binary search
- Sorting
    - Selection sort
    - Bubble sort
    - Merge Sort
- Recursion
- Data structures

Algorithmic complexity

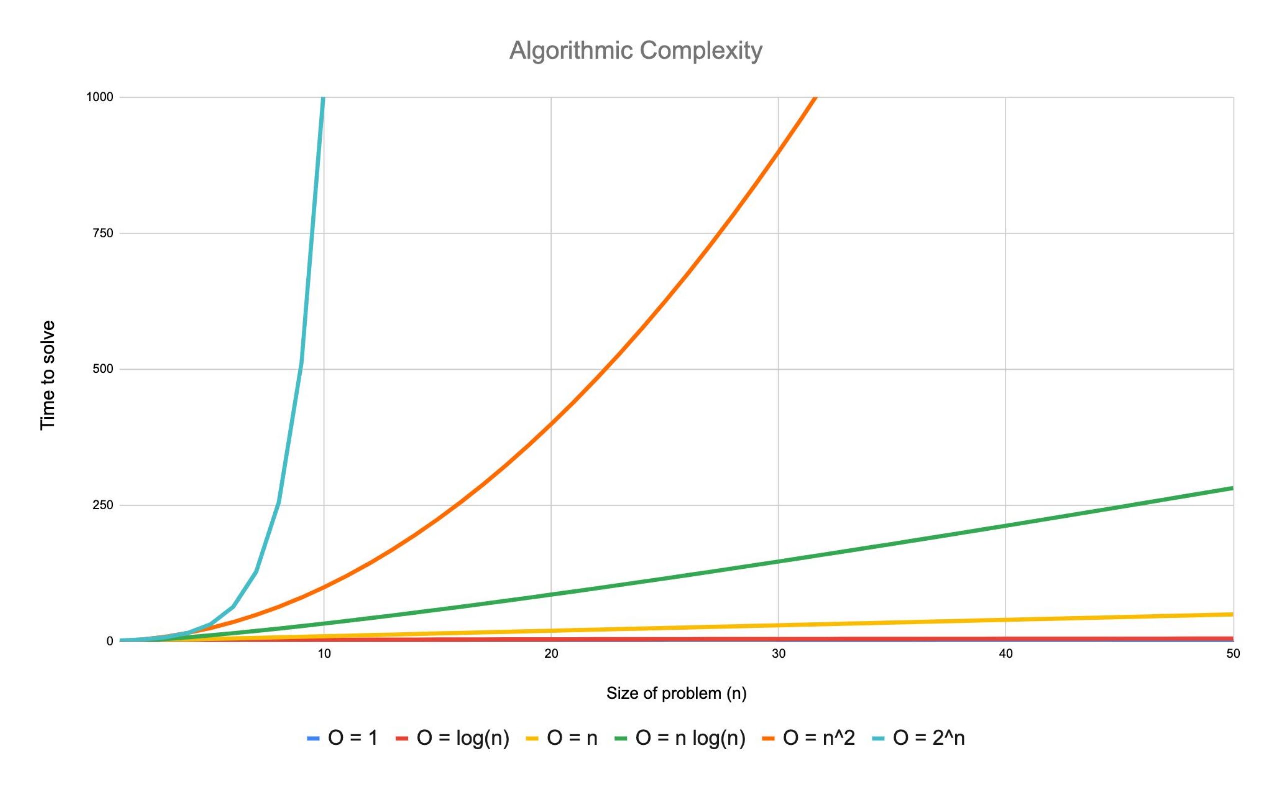
Big O

- How many steps does your algorithm take for each value passed into it?
  - O ("big O") is the worst-case for your algorithm – this is what we want to consider
  - Ω ("omega") is the best-case – but we usually don't care about that
  - Θ ("theta") is a special case where O == Ω
- Common running times (low to high)
  - Constant: $O(1)$
  - Log: $O(\log n)$
  - Linear: $O(n)$
  - Log-linear: $O(n \log n)$
  - Quadratic: $O(n^2)$ (or, more generally, polynomial)
  - Exponential: $O(2^n)$

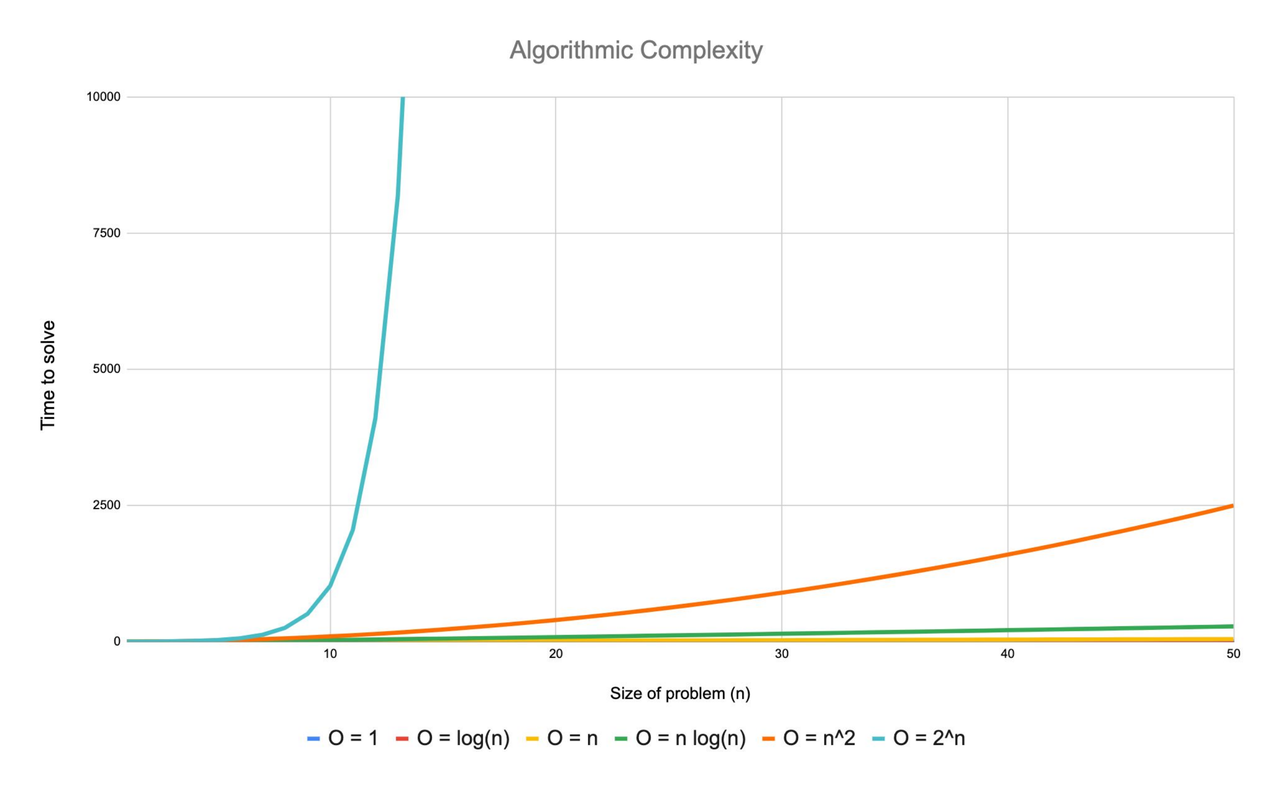
Visualizing *O*



Visualizing *O*



Visualizing *O*



Search

Linear search

- Algorithm:
  - Iterate through each item in the array
    - If the desired value is found, `return true`
    - If you reach the end of the array and do not find the value, `return false`
- Benefits:
  - Simple
  - Works with unsorted data
- Drawbacks:
  - Slow-ish – *O*(n)

Binary search

- Algorithm:
  - If no values remain, `return false`
  - Find middle point of array
  - If your value is found, `return true`
  - Else if your value is less than the current value
    - Search the left half
  - Else if your value is greater than the current value
    - Search the right half
- Benefit:
  - Faster – *O*(log n)
- Drawback:
  - Array needs to be sorted – additional "upfront" cost

`binary_search_0.c`

Sorting

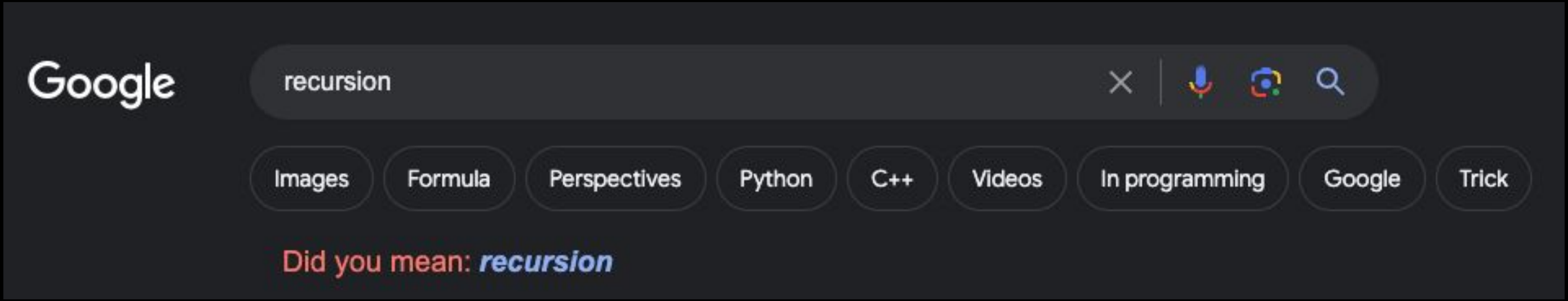
13, 5, 3, 10, 8, 4, 1, 14, 2, 6, 9, 12, 7, 15, 11

https://visualgo.net/en/sorting

Sorting types

- Selection sort
  - Find smallest element, move to the front of the unsorted portion
- Bubble sort
  - Compare number and its neighbor. If the first number is bigger, swap them
  - If no swaps, quit
- Merge sort
  - Look at half the array at a time (and half of that… and half of that…)
  - Merge partially sorted arrays to create larger sorted arrays

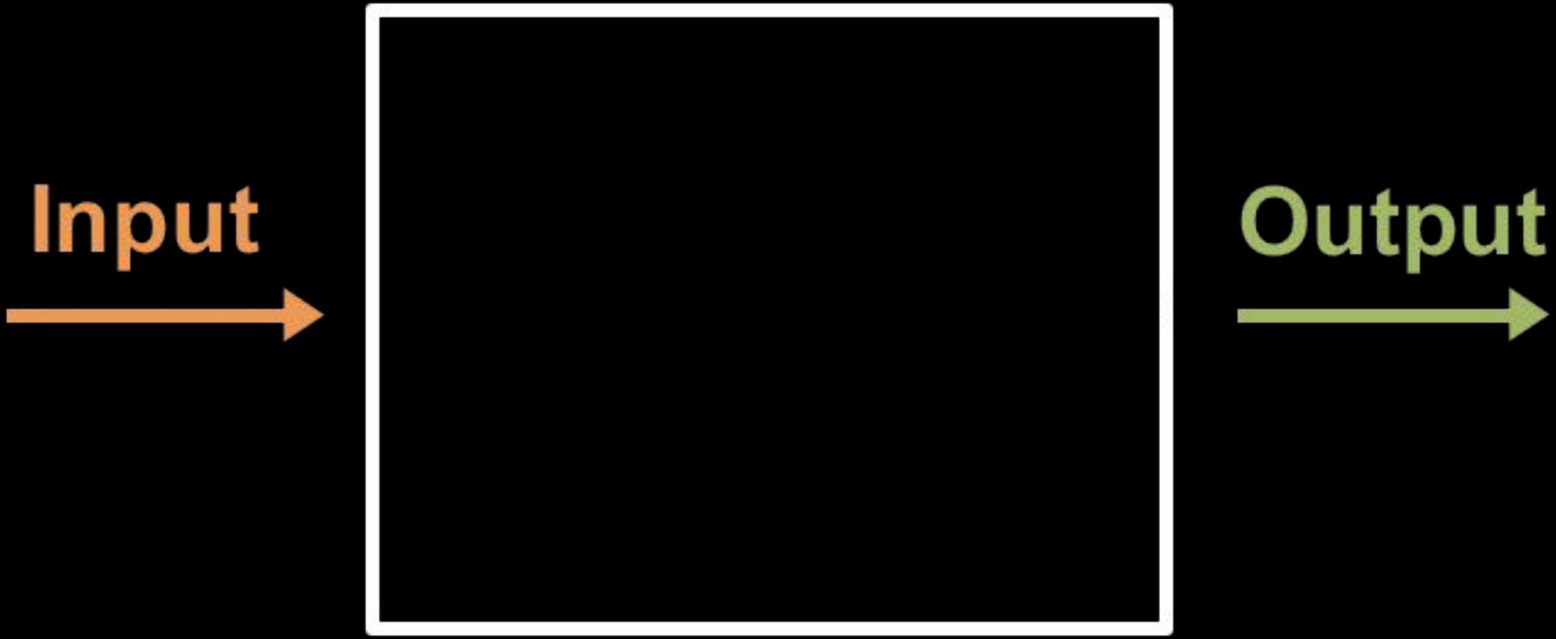
Recursion



recursion

Did you mean: ***recursion***

Recursive functions



Recursion example: Factorial!

factorial(1)	1
factorial(2)	2 * 1
factorial(3)	3 * 2 * 1
factorial(4)	4 * 3 * 2 * 1
factorial(5)	5 * 4 * 3 * 2 * 1

Recursion example: Factorial!

`factorial(1)`	`1`
`factorial(2)`	`2 * factorial(1)`
`factorial(3)`	`3 * 2 * 1`
`factorial(4)`	`4 * 3 * 2 * 1`
`factorial(5)`	`5 * 4 * 3 * 2 * 1`

Recursion example: Factorial!

factorial(1)	1
factorial(2)	2 * factorial(1)
factorial(3)	3 * factorial(2)
factorial(4)	4 * 3 * 2 * 1
factorial(5)	5 * 4 * 3 * 2 * 1

Recursion example: Factorial!

factorial(1)	1
factorial(2)	2 * factorial(1)
factorial(3)	3 * factorial(2)
factorial(4)	4 * factorial(3)
factorial(5)	5 * 4 * 3 * 2 * 1

Recursion example: Factorial!

factorial(1)	1
factorial(2)	2 * factorial(1)
factorial(3)	3 * factorial(2)
factorial(4)	4 * factorial(3)
factorial(5)	5 * factorial(4)

Recursion example: Factorial!

factorial(1)	1
factorial(2)	2 * factorial(1)
factorial(3)	3 * factorial(2)
factorial(4)	4 * factorial(3)
factorial(5)	5 * factorial(4)
factorial(*n*)	*n* * factorial(*n* - 1) -- for all n >= 1

Parts of a recursive function

- Base case (otherwise our function would run forever!)
- Recursive case

```
int fact(int n)
{
    // Base case

    // Recursive case

}
```

```
int fact(int n)
{
    if n == 1
        return 1;
    else
        return n * fact(n - 1);
}
```

factorial.c
binary_search_1.c

Data Structures

Structs

- In C, you have to define your variable type.
- But what if different types make sense grouped together

```
favorites[] = {“purple”, 13, 3.14}
```

- Structs get around this by grouping things together

typedef

```
typedef struct
{
    string color;
    int number;
    float irrational;
} favorites;
```

```
favorites josh;
josh.color = “purple”;
josh.number = 13;
josh.irrational = 3.14;
```

accessing info in structs

```
typedef struct favorites
{
    string color;
    int number;
    float irrational;
} favorites;
```

```
favorites josh;
josh.color = “purple”;
josh.number = 13;
josh.irrational = 3.14;

printf(“%s”, josh.color);
```

airports.c

Questions?