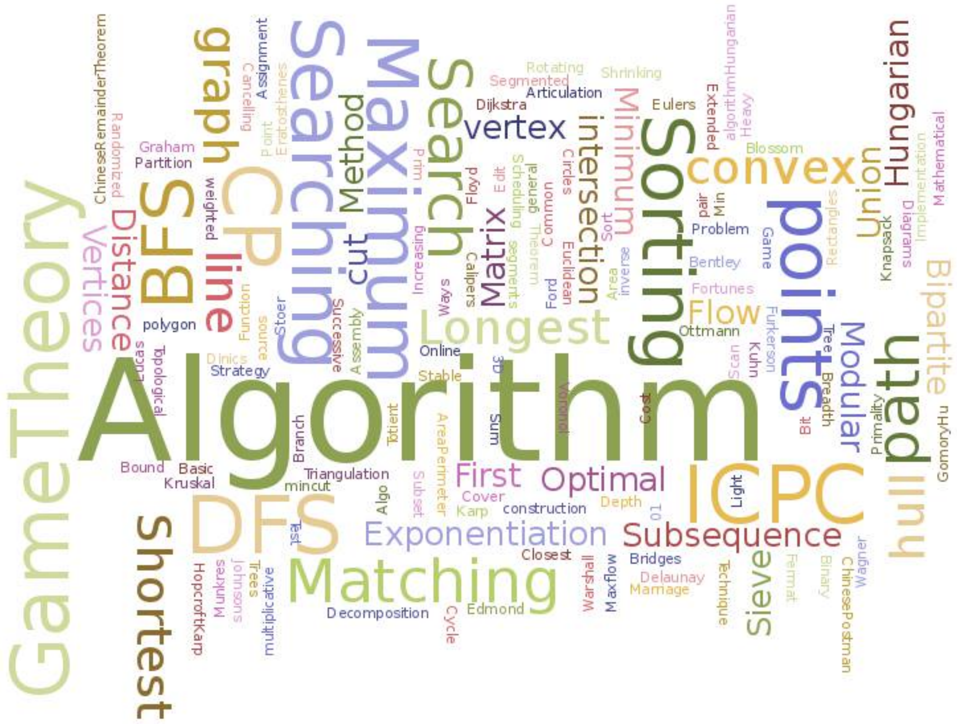


Content

- Introduction
- Motivation
- Analysis complexity
- Algorithm categories
- Competitive programming
- Let's do it!

Motivation

- **What is this for?**

Motivation

- What is this for?
- **Work for companies like Google, Amazon, Microsoft, etc?**

Motivation

- What is this for?
- Work for companies like Google, Amazon, Microsoft, etc?
- **Glory, prizes, and more**

Motivation

- What is this for?
- Work for companies like Google, Amazon, Microsoft, etc?
- Glory, prizes, and more
- **Solve chanllenges is fun!**

> **It's like practicing any other discipline**

Introduction

- **What is an algorithm?**

Introduction

- **What is an algorithm?**

> In mathematics and computer science, **an algorithm is a finite sequence of well-defined**, computer-implementable instructions, typically **to solve a class of problems or to perform a computation** - Wiki (https://en.wikipedia.org/wiki/Algorithm)

Analysis complexity

- **Time complexity**

Analysis complexity

- Time complexity
- **Space complexity**

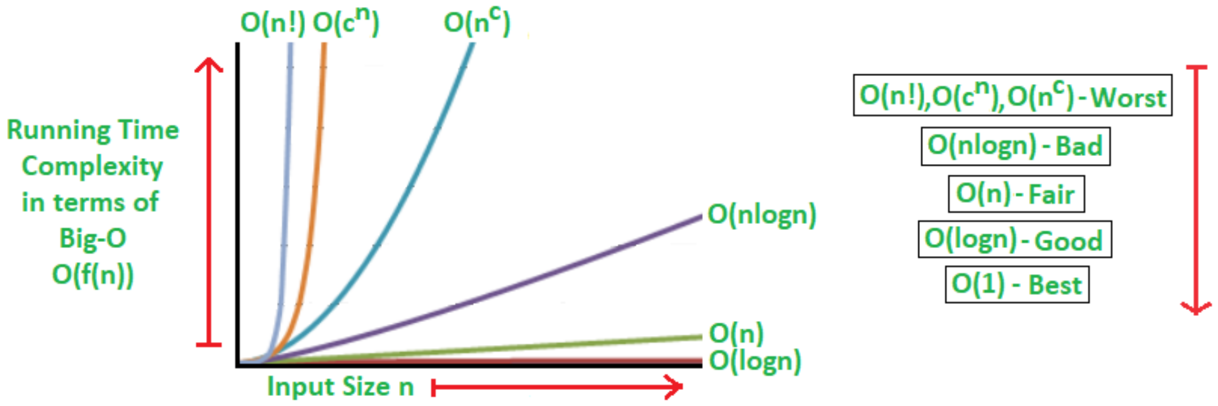
Analysis complexity

- Time complexity
- Space complexity
- **Big O notation**: describes how complex an algorithm is

Analysis complexity

- Time complexity
- Space complexity
- Big O notation: describes how complex an algorithm is

- **Best case** — Big Omega or Ω(n)
- **Average case** — Big Theta or Θ(n)
- **Worst case** — Big O Notation or O(n)



In [3]:

```
# Print first N numbers
n = 100
for i in range(n):
    print(i,' ',end='')

# O(n)
```

```
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  2
0  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37
38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55
56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73
74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91
92  93  94  95  96  97  98  99
```

In [4]:

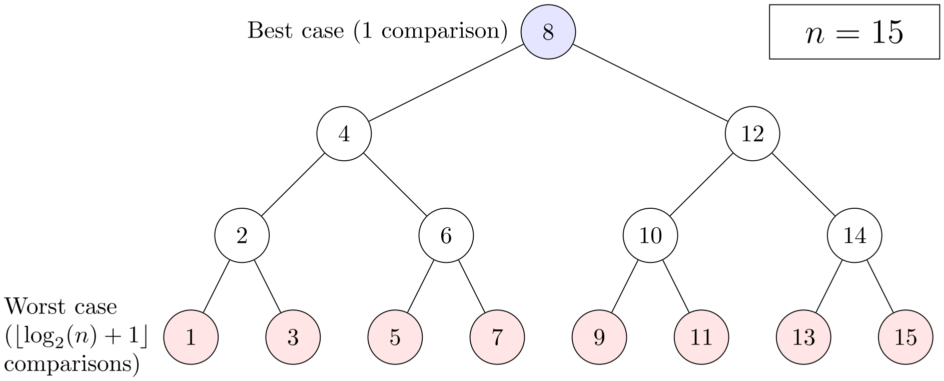
```
# Print all possible combinations of product between 2 different numbe
rs
# n = len(n)
arr = [1, 3, 4, 5]
out = []

for i in range(0, len(arr)): # n
    for j in range(0, len(arr)): #n
        out.append(arr[i] * arr[j])

# O(n^2)
print('arr: ', out)
```

```
arr:  [1, 3, 4, 5, 3, 9, 12, 15, 4, 12, 16, 20, 5, 15, 20, 25]
```

In [11]:

```
# Lower bound
#arr = [5, 3, 7, 9, 4, 1]
arr = [1] * 100
arr.sort() # O(n*log(n))
# arr = [1, 3, 4, 5, 7, 9]
# index [0, 1, 2, 3, 4, 5]
steps = 0
def binary_search(x, arr):
    lo = 0
    hi = len(arr) - 1
    global steps

    while (lo < hi):
        steps += 1
        mid = int((lo + hi + 1) / 2)
        if (arr[mid] <= x):
            lo = mid
        else:
            hi = mid - 1
    return lo
# O(n*log(n)) + O(log_2(n)) + c
# O(n*log(n))
x = 10
ind = binary_search(x, arr)
print('index: ', ind)
print('steps', steps)
```

```
index:  99
steps 7
```



In [6]:

```
# log_b(x) = log_c(x) / log_c(b)

from math import log
def calc_log2(n):
    return log(n) / log(2)

calc_log2(100)
```

Out[6]:

```
6.643856189774725
```

Algorithm categories

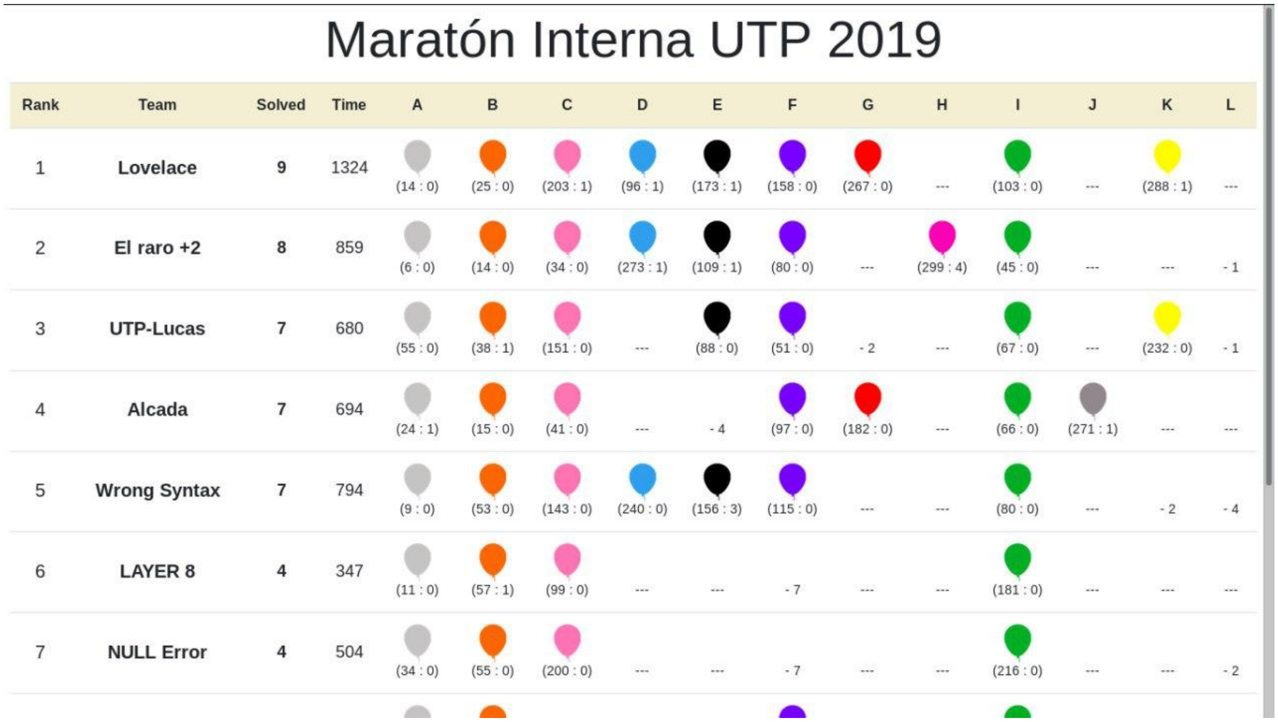
- Math: GCD, primes, modular arithmetic...
- Graphs: Shortest paths, toposort, Traveling,...
- Dynamic programming: Knapsack problem, LIS...
- Geometry: convex hull, Area of a polygon, ..
- Greedy
- Implementation

Competitive programming

- Description
- Problem set
- Scoreboard
- Teams or individual
- On site or on Internet
- Prizes

Competitive programming

- IOI **International Olympiad in Informatics**: College
- ICPC **International Collegiate Programming Contest**: University
- Google Codejam: online



Maratón Interna UTP 2019

Rank	Team	Solved	Time	A	B	C	D	E	F	G	H	I	J	K	L
1	**Lovelace**	**9**	1324	(14 : 0)	(25 : 0)	(203 : 1)	(96 : 1)	(173 : 1)	(158 : 0)	(267 : 0)	---	(103 : 0)	---	(288 : 1)	---
2	**El raro +2**	**8**	859	(6 : 0)	(14 : 0)	(34 : 0)	(273 : 1)	(109 : 1)	(80 : 0)	---	(299 : 4)	(45 : 0)	---	---	- 1
3	**UTP-Lucas**	**7**	680	(55 : 0)	(38 : 1)	(151 : 0)	---	(88 : 0)	(51 : 0)	- 2	---	(67 : 0)	---	(232 : 0)	- 1
4	**Alcada**	**7**	694	(24 : 1)	(15 : 0)	(41 : 0)	---	- 4	(97 : 0)	(182 : 0)	---	(66 : 0)	(271 : 1)	---	---
5	**Wrong Syntax**	**7**	794	(9 : 0)	(53 : 0)	(143 : 0)	(240 : 0)	(156 : 3)	(115 : 0)	---	---	(80 : 0)	---	- 2	- 4
6	**LAYER 8**	**4**	347	(11 : 0)	(57 : 1)	(99 : 0)	---	---	- 7	---	---	(181 : 0)	---	---	---
7	**NULL Error**	**4**	504	(34 : 0)	(55 : 0)	(200 : 0)	---	---	- 7	---	---	(216 : 0)	---	---	- 2

Let's do it!

Problem description

Toby And The Coins Toby is going to buy a machine to send love letters to his girlfriend, the machine costs **N** pesos. Toby works very hard and he has a lot of money, in fact, he can pay the machine with any combination of coins! **Toby wants to know what is the minimum number of coins he needs to buy the machine.**

In the Toby's city there are coins with the following values: **1, 2, 5, 10, 20**

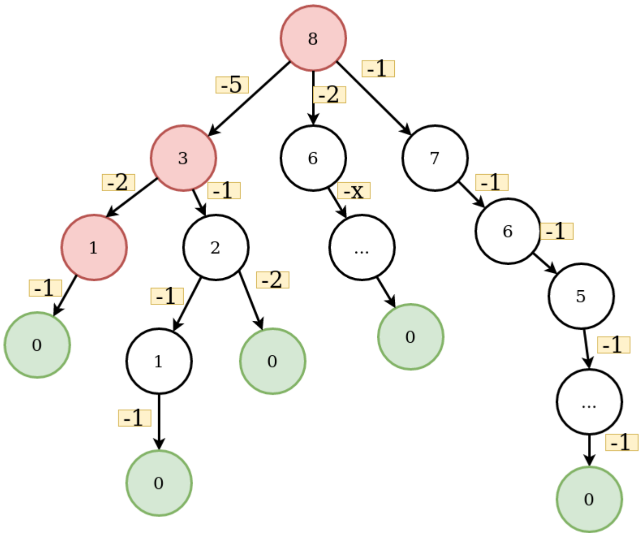
Sample Input	Sample Output
15	2
8	3
22	2

Explanation

Values to use: [1, 2, 5, 10, 20]

N	Sumatory	Total used
15	10 + 5	2
8	5 + 2 + 1	3
22	20 + 2	2

First approach



First approach

Implementation

In [2]:

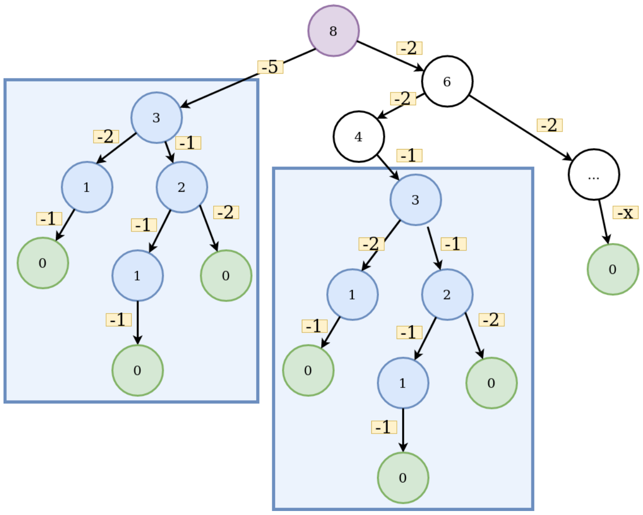
```python
array = [1, 2, 5, 10, 20]
# 24 - 20 -> 4 - 2 -> 2 - 2 -> 0
# 20 + 2 + 2
def solve(n, steps):
    if n == 0:
        return steps

    ans = 1e9
    for num in array: # 5
        if (n - num >= 0):
            ans = min(solve(n - num, steps + 1), ans)
    return ans
# O(5 ^ k)

solve(26, 0)
```

Out[2]:

```
3
```

First approach

How to improve this?



Conditions to use Dynamic programming

```
a) Optimal Substructure
b) Overlapping subproblem
```

In [3]:

```
array = [1, 2, 5, 10, 20]
memo = [-1] * 1000
# state
def solve(n):
    if n == 0: return 0
    if memo[n] != -1: return memo[n]

    ans = 1e9
    for num in array:
        if (n - num >= 0):
            ans = min(solve(n - num) + 1, ans)
    memo[n] = ans
    return memo[n]
# O(5^k)
# O(n * 5) = O(n)
solve(200)
```

Out[3]:

```
10
```

Second approach

Looking for an strategy...

In [4]:

```
#array = [1, 2, 5, 10, 20]
array = [20, 10, 5, 2, 1] # sorted

def solve(n):
    steps = 0
    ind = 0
    for num in array:
        while (n - num >= 0):
            n -= num
            steps += 1

    return steps

solve(100)
```

Out[4]:

```
5
```

One last example

Print the first **N** prime number

In [1]:

```
def is_prime(n):
    if (n == 1): return False
    i = 2
    while i * i <= n:
        if n % i == 0: return False
        i += 1
    return True

for i in range(1, 20):
    print(i, is_prime(i))
```

In [4]:

```
criba = [1] * 30
primes = []
def fill_criba(MX):
    i = 2
    while i <= MX:
        if criba[i]:
            primes.append(i)
            for j in range(i,MX + 1,i):
                criba[j] = 0
        i += 1

fill_criba(20)
```

Thanks!

Resources

- https://codeforces.com (https://codeforces.com)
- https://www.hackerrank.com (https://www.hackerrank.com)
- https://www.codechef.com (https://www.codechef.com)
- https://codingcompetitions.withgoogle.com/codejam (https://codingcompetitions.withgoogle.com/codejam)