Session 0

Kasra Eskandari

November 15, 2020

Contents

1 Time Complexity

1.1 What is time complexity ?

Imagine I want to find specific name in the class(with N members), there is several aproaches to do so:

- ask one by one indivuals whether they are that specific person or not. in this case you will need to ask N questions
- Fist break class into two equal groups and then ask if that specifice person is in the one of groups or not, and recursively do the same thing on the target group. in this case you will need $\log(N)$ questions(WHY?).

lets turn this into code: find an specific number in given sorted array.

[1]:
```
def approach0(num,arr):
    ans = False
    for i in arr:
        if num == i:
            ans = True
    return ans

def approach1(num,arr):
    for i in arr:
        if num == i:
            return True
    return False

def approach2(num,arr):
    low = 0
    high = len(arr)
    while high-low>2:
        mid = (low+high)//2
        middleValue = arr[mid]
        if num > middleValue:
            low = mid
        elif num < middleValue:
            high = mid
        elif num == middleValue:
            return True
    return False
```

[2]:
```
# lets test functions using radom values
from random import randint

N = 100
# generate test cases
testCases = sorted([randint(0,N) for _ in range(N)])
```

```
num = 10

print(10 in testCases)
print(approach1(10,testCases))
print(approach2(10,testCases))
```

```
True
True
True
```

[3]:
```
############# Bonuce part ####################
# lets count number of equality checks

class myNum(int):
    def __init__(self,num):
        super().__init__()
        self.numberOfEquality = 0
        self.numberOfInequality = 0
        self.num = num

    def __eq__(self,other):
        self.numberOfEquality+=1
        return self.num == other

    def __lt__(self,other):
        self.numberOfInequality+=1
        return self.num<other

    def __gt__(self,other):
        self.numberOfInequality+=1
        return self.num>other


N = 10**6
testCases = sorted([randint(0,N) for _ in range(N)])


for i in [N//4,N//2,N//4*3,N-1]:
    num = myNum(i)
    print(f"{num=}")
    print(f"{approach1(num,testCases)=}")
    print(f"{num.numberOfEquality=}")

    num = myNum(10)
    print(f"{approach2(num,testCases)=}")
    print(f"{num.numberOfEquality=}")
    print(f"{num.numberOfInequality=}")
```

```
    print("-"*10)
```

```
num=250000
approach1(num,testCases)=False
num.numberOfEqulity=1000000
approach2(num,testCases)=True
num.numberOfEqulity=1
num.numberOfInequality=32
----------
num=500000
approach1(num,testCases)=True
num.numberOfEqulity=500936
approach2(num,testCases)=True
num.numberOfEqulity=1
num.numberOfInequality=32
----------
num=750000
approach1(num,testCases)=True
num.numberOfEqulity=750005
approach2(num,testCases)=True
num.numberOfEqulity=1
num.numberOfInequality=32
----------
num=999999
approach1(num,testCases)=True
num.numberOfEqulity=999999
approach2(num,testCases)=True
num.numberOfEqulity=1
num.numberOfInequality=32
----------
```

1.1.1 More Examples

[4]:
```
N = 100

# Example 1
x=0
for i in range(N):
    for j in range(N):
        x+=1
print(f"Example 1: {x}")

# Example 2
x=0
for i in range(N):
    for j in range(i,N):
        x+=1
```

```python
print(f"Example 2: {x}")

# Example 3
i=1
x=0
while i<N:
    x+=1
    i*=2
print(f"Example 3: {x}")

# Example 4
x=0
for i in range(2*N):
    j=1
    while j<i:
        x+=1
        j*=2
print(f"Example 4: {x}")
```

```
Example 1: 10000
Example 2: 5050
Example 3: 7
Example 4: 1337
```

Example 1 its obvious :)

Example 2 - number of second loop iteration in first iteration of first loop($i = 0$): N - number of second loop iteration in second iteration of first loop($i = 1$): $N - 1$ - number of second loop iteration in third iteration of first loop($i = 2$): $N - 2$ - number of second loop iteration in 4th iteration of first loop($i = 3$): $N - 3$ - number of second loop iteration in 5th iteration of first loop($i = 4$): $N - 4 \ldots$ - number of second loop iteration in (N-1)th iteration of first loop($i = N - 2$): 1 - number of second loop iteration in Nth iteration of first loop($i = N - 1$): 0

So:

$$x = 0 + 1 + ... + (N - 1) + N = \frac{N(N+1)}{2} = 5050$$

Example 3 clearly the Values of i are power of 2:

$$i \in \{2^k | k \in \mathbb{N}\} = \{1, 2, 4, 8, 16, 32, 64, ...\}$$

We know $i = 2^x$ at the end of each loop iteration

lets find maximum i

$$\begin{aligned} i &< N \\ 2^x &< N \\ x &< \lg(N) \end{aligned}$$

And we know $\lg(100) = 7$

Example 4

$$x = \sum_{i=0}^{2N} \lg(i) = \lg(2N!)$$

[5]:
```
from math import factorial, log2
log2(factorial(2*N))
# WHY ?
```

[5]: 1245.3805070592086

2 Lets dive into mathematics

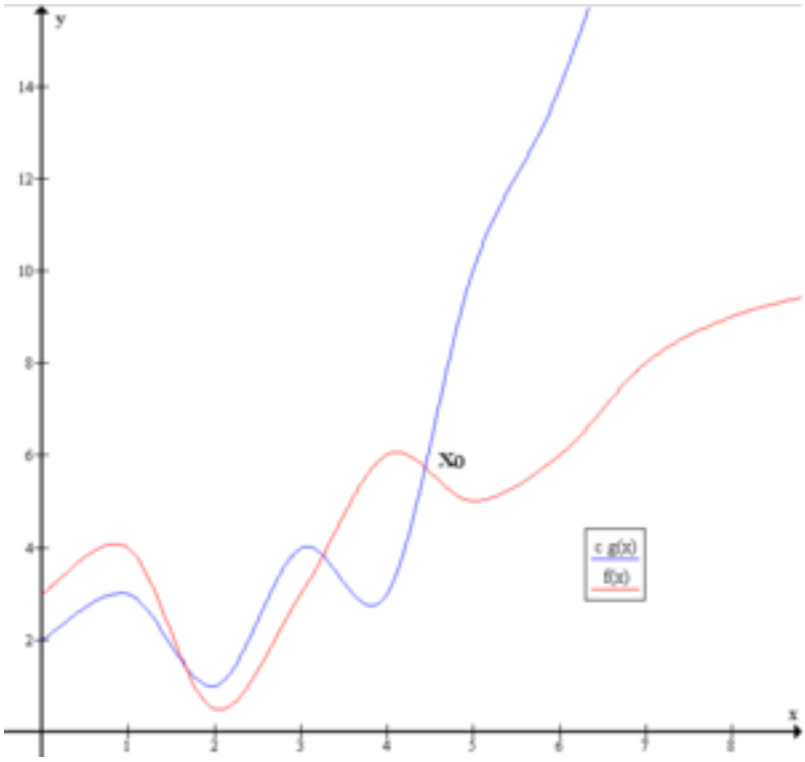
2.1 Big Oh Notation

Since O-notation describes an upper bound, when we use it to bound the worstcase running time of an algorithm, we have a bound on the running time of the algorithm on every input—the blanket statement we discussed earlier.

$$\mathcal{O}(f(n)) = \{g(n) | \exists\, c > 0,\ n_0 \ \ where\ f(n) \leqslant cg(n)\ \forall\ n > n_0\}$$

OR

$$g(n) \in \mathcal{O}(f(n)) \implies \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$



Example - $n^2 + 13n + 43 \in \mathcal{O}(n^2)$ - $\frac{1}{1000}n^4 + 10000000n^3 + 10^{10^{10}} \in \mathcal{O}(n^4)$

2.1.1 Summation and Production

- $\mathcal{O}(f(n)) + \mathcal{O}(g(n)) = \mathcal{O}(f(n) + g(n))$
- $\mathcal{O}(f(n)) \times \mathcal{O}(g(n)) = \mathcal{O}(f(n) \times g(n))$
- $\mathcal{O}(cf(n)) = \mathcal{O}(f(n))$

Notation	Name	Example
$\mathcal{O}(1)$	constant	Determining if a binary number is even or odd
$\mathcal{O}(\log(n))$	logarithmic	Finding an item in a sorted array with a binary search or a balanced search tree as well as all operations in a Binomial heap
$\mathcal{O}(n)$	linear	Finding an item in an unsorted list
$\mathcal{O}(n \log n) = \mathcal{O}(\log n!)$	linearithmic, loglinear, quasilinear	quick sort
$\mathcal{O}\left(n^c\right)_{,c>1}$	fractional power	bubble sort($c = 2$)
$\mathcal{O}\left(c^n\right)_{,c>1}$	exponential	Finding the (exact) solution to the travelling salesman problem using dynamic programming; determining if two logical statements are equivalent using brute-force search
$\mathcal{O}(n!)$	factorial	Solving the travelling salesman problem via brute-force search; generating all unrestricted permutations of a list

check python operations complexity here

3 Recursive Functions

[6]:
```
def fact(n):
    if n==2:
        return 2
    return n*fact(n-1)
print(fact(50))
```

```
30414093201713378043612608166064768844377641568960512000000000000
```

This function works in $\mathcal{O}(n)$

[7]:
```
def fib0(n):
    if n in [2,1]:
        return 1
    return fib0(n-1)+fib0(n-2)

def fib1(n):
    a,b=1,1
    for i in range(n-1):
        a,b = b,a+b
    return b
print(fib0(10))
print(fib1(10))
```

```
55
89
```

Question: find the complexity of both functions

4 Arrays

a contiguous area of memory that is one chunk of memory that can either be on the stack or it can be in the heap, doesn't really matter where it is. It is broken down into **equal-sized elements**, and each of those elements are **indexed by contiguous integers**.

4.1 Why Arrays are Important?

the address of our particular array element is:

$$array\_addr + elem\_size \times (i - first\_index)$$

memory address	value	index
100	4	0
104	5	1
108	67	2
112	8	3
116	9	4

memory address	value	index
120	2	5

[8]:
```
from array import array
arr = array('i',[4,5,67,8,9,2])
buffer_info = arr.buffer_info()
print(f"{buffer_info=}")
print(hex(buffer_info[0]))
```

```
buffer_info=(140304787604656, 6)
0x7f9b410684b0
```

4.2 Two Dimensional Arrays

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

4.2.1 Row-major

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} & a_{10} & a_{11} & a_{12} & a_{13} & a_{20} & a_{21} & a_{22} & a_{23} & a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Now to access a_{ij} for $M \times N$ matrix:

$$array\_adr + elem\_size(M(i - I) + (j - J))$$

4.2.2 Column-major

$$\begin{bmatrix} a_{00} & a_{10} & a_{20} & a_{30} & a_{01} & a_{11} & a_{21} & a_{31} & a_{02} & a_{12} & a_{22} & a_{32} & a_{03} & a_{13} & a_{23} & a_{33} \end{bmatrix}$$

Now to access a_{ij} for $M \times N$ matrix:

$$array\_adr + elem\_size((i - I) + N(j - J))$$

5 Homeworks :)

Question1: consider the question "find the number of zeros at the end of $n!$ number

```
def findLastZeros0(n):
    nfact=fact(n)
    counter = 0
    while nfact%10 == 0:
        nfact/=10
        counter+=1
    return counter
```

And:

```
def findLastZeros1(n):
    counter = 0
    t = 1
    for i in str(fact(n))[::-1]:
        t = int(i)%11
        counter+=t
    return counter
```

Or this is as same as:

```
def findLastZeros2(n):
    t=1
    return sum((t:=int(i=='0' and t)) for i in str(fact(n))[::-1])
```

your task is to: 1. find the time complexity of these functions, and find out wich one is faster 2. find a faster way to compute same value

[9]:
```
def findLastZeros0(n):
    nfact=fact(n)
    counter = 0
    while nfact%10 == 0:
        nfact//=10
        counter+=1
    return counter
def findLastZeros1(n):
    counter = 0
    t = 1
    for i in str(fact(n))[::-1]:
        t = int(i=='0' and t)
        counter+=t
    return counter

def findLastZeros2(n):
    t=1
    return sum((t:=int(i=='0' and t)) for i in str(fact(n))[::-1])

print(f"{fact(30)=}")
print(f"{findLastZeros0(30)=}")
print(f"{findLastZeros1(30)=}")
print(f"{findLastZeros2(30)=}")
```

```
fact(30)=265252859812191058636308480000000
findLastZeros0(30)=7
findLastZeros1(30)=7
findLastZeros2(30)=7
```

Question2: implement two functions to convert two-dimensional array to one-dimensional and vise versa here is a template code for you.

[10]:
```
def _2Dto1D(arr,method='Row-major'):
    if method not in ['Row-major','Col-major']:
        raise ValueError(f"mathod must be either 'Row-major' or 'Col-major' not␣
 ↪{method}")
    # write your code

def _1Dto2D(arr,n,m,method='Row-major'):
    assert method in ['Row-major','Col-major'], f"method must be either␣
 ↪'Row-major' or 'Col-major' not {method}"
    assert n*m==len(arr), f"{n*m=} which must be {len(arr)}"
    # write your code here
```