Intro

November 8, 2021

1 Algorithms and Data Structures

1.1 Why

1. Helps you to practice.
2. Basic concepts and ideas that you will se again and again.
3. Solving these problems can help you solve bigger problems.
4. Builds confidence.
5. Rehersal for technical interviews.
6. Helps understand how more advance software works.
7. General Knowledge of programming.
8. Forces you to try to solve a problem step by step.
9. You need to be able to communicate ideas.
10. Convert ideas into code.
11. You need to think about inputs, outputs, test cases and edge cases.

2 Algorithms

It is a finite sequence of well defined steps to solve a specific problem.

2.1 Steps to solve a problem

1. We have to define the problem with very clear language.
2. Think about possible data scenarios.
3. Define solution for the problem.
4. Implement the solution in your language of preference.
5. Test solution and use cases.
6. Analyze algorithm's complexity and performance.
7. Find a better solution, if possible.

2.1.1 Problem definition

Given a sorted list of number find a the location of a given number in that list.

2.1.2 Data scenarios

The following are some examples of data input for the problem

[1]:
```python
numbers=[1,3,7,13,21,45,67,89]
search=45
result=5
```

2.2 Linear Search

[2]:
```python
def find_number_linear(numbers, search):
    for i in range(len(numbers)):
        if numbers[i]==search:
            return i;
    return -1
```

[3]:
```python
find_number_linear(numbers,search)==result
```

[3]: True

2.3 Validation as Dictionary

[4]:
```python
validation= {
    'data':{
        'numbers': [1,3,7,13,21,45,67,89],
        'search': 45
    },
    'result' : 5,
    'description':'Simple test'
}
```

[5]:
```python
find_number_linear(**validation['data']) == validation['result']
```

[5]: True

2.4 Multiple validations

[6]:
```python
def test_validations(function,validations):
    for test in validations:
        result=function(**test['data'])
        if result== test['result']:
            print('Test Passed ->: {}'.format(test['description']))
        else:
            print('Test Failed ->: {}'.format(test['description']))

        print('\t Input   ->: Input {}'.format(test['data']))
        print('\t Results ->: Output {}, expected {}'.
 ↪format(result,test['result']))
```

[7]:
```python
validations= []
```

[8]:
```
validations.append({
    'data':{
        'numbers': [1,3,7,13,21,45,67,89],
        'search': 1
    },
    'result' : 0,
    'description':'Number at the beginning'
})
```

[9]:
```
validations.append({
    'data':{
        'numbers': [1,3,7,13,21,45,67,89],
        'search': 89
    },
    'result' : 7,
    'description':'Number at the end'

})
```

[10]:
```
validations.append({
    'data':{
        'numbers': [1,3,7,13,21,45,67,89],
        'search': 100
    },
    'result' : -1,
    'description':'Missing number'
})
```

[11]:
```
validations.append({
    'data':{
        'numbers': [89],
        'search': 89
    },
    'result' : 0,
    'description':'One number only'
})
```

[12]:
```
validations.append({
    'data':{
        'numbers': [1,3,3,4,7,13,21,45,67,89],
        'search': 7
    },
    'result' : 4,
    'description':'Number Repetions'

})
```

[13]:
```
test_validations(find_number_linear,validations)
```

```
Test Passed ->: Number at the beginning
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
1}
         Results ->: Output 0, expected 0
Test Passed ->: Number at the end
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
89}
         Results ->: Output 7, expected 7
Test Passed ->: Missing number
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
100}
         Results ->: Output -1, expected -1
Test Passed ->: One number only
         Input   ->: Input {'numbers': [89], 'search': 89}
         Results ->: Output 0, expected 0
Test Passed ->: Number Repetions
         Input   ->: Input {'numbers': [1, 3, 3, 4, 7, 13, 21, 45, 67, 89],
'search': 7}
         Results ->: Output 4, expected 4
```

[14]:
```
validations.append({
    'data':{
        'numbers': [],
        'search': 1
    },
    'result' : -1,
    'description':'Empty'

})
```

[15]:
```
test_validations(find_number_linear,validations)
```

```
Test Passed ->: Number at the beginning
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
1}
         Results ->: Output 0, expected 0
Test Passed ->: Number at the end
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
89}
         Results ->: Output 7, expected 7
Test Passed ->: Missing number
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
100}
         Results ->: Output -1, expected -1
Test Passed ->: One number only
         Input   ->: Input {'numbers': [89], 'search': 89}
```

```
         Results ->: Output 0, expected 0
Test Passed ->: Number Repetions
         Input   ->: Input {'numbers': [1, 3, 3, 4, 7, 13, 21, 45, 67, 89],
'search': 7}
         Results ->: Output 4, expected 4
Test Passed ->: Empty
         Input   ->: Input {'numbers': [], 'search': 1}
         Results ->: Output -1, expected -1
```

[16]:
```python
large_validation = {
    'data': {
        'numbers': [i for i in range(10000000)],
        'search': 99999999
    },
    'result': 99999999,
    'description':'Very large list'
}
```

[17]:
```python
%%time
res=find_number_linear(**large_validation['data'])
```

```
CPU times: user 295 ms, sys: 0 ns, total: 295 ms
Wall time: 294 ms
```

2.5 Binary Search

2.5.1 Steps

1. Find the number in the middle.
2. If it matches then you are done.
3. If the number is lower than the search look in the left side of the array.
4. If the number is higher than the serach look in the rigth side of the array.
5. If there are no more numbers return -1

[18]:
```python
def find_number_binary(numbers, search):
    rigth= 0
    left= len(numbers)-1

    while rigth <= left:
        center = (rigth + left) // 2
        center_number = numbers[center]

        if center_number == search:
            return center
        elif center_number < search:
            rigth = center + 1
        elif center_number > search:
            left = center - 1
```

```python
    return -1
```

[19]:
```python
test_validations(find_number_binary,validations)
```

```
Test Passed ->: Number at the beginning
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
1}
         Results ->: Output 0, expected 0
Test Passed ->: Number at the end
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
89}
         Results ->: Output 7, expected 7
Test Passed ->: Missing number
         Input   ->: Input {'numbers': [1, 3, 7, 13, 21, 45, 67, 89], 'search':
100}
         Results ->: Output -1, expected -1
Test Passed ->: One number only
         Input   ->: Input {'numbers': [89], 'search': 89}
         Results ->: Output 0, expected 0
Test Passed ->: Number Repetions
         Input   ->: Input {'numbers': [1, 3, 3, 4, 7, 13, 21, 45, 67, 89],
'search': 7}
         Results ->: Output 4, expected 4
Test Passed ->: Empty
         Input   ->: Input {'numbers': [], 'search': 1}
         Results ->: Output -1, expected -1
```

[20]:
```python
%%time
res=find_number_binary(**large_validation['data'])
```

```
CPU times: user 13 µs, sys: 2 µs, total: 15 µs
Wall time: 17.2 µs
```

[21]:
```python
import time
def get_time(function,validation):
    start = time.time()
    function(**validation['data'])
    end = time.time()
    return(end - start)

def simulation(function,validation,sim_number, increment):
    step=sim_number//increment
    results=[]
    for i in range(0,step):
        validation['data']['search']=i*increment
        validation['result']=i*increment
        results.append(get_time(function,validation))
```

```python
    return results
```

3 Complexity

The complexity of a algoritm is based on the amount of time or space it take to complete the process for a input of size N.

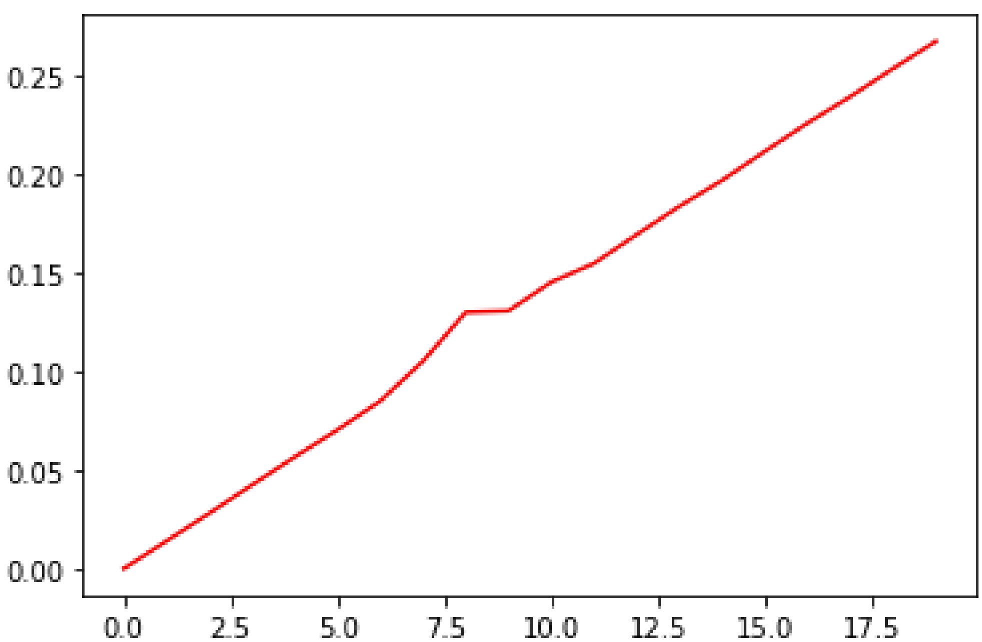
For the previous search function the time complexity is O(N) where N is the size of the array passed to the function and O(1) for space complexity.

3.1 Linear Search Complexity

During linear Search the Maxinum number of operations that are possible is N. Therefore, the complexity of this algorithm is $O(N)$

3.2 Simulation Results

[22]:
```python
import matplotlib.pyplot as plt
sim_number=10000000
increment=500000
ypoints_lineal=simulation(find_number_linear,large_validation,sim_number,increment)
plt.plot(ypoints_lineal,color='r')
plt.show()
```



3.3 Binary Search Complexity

Binary Search break the search list in smaller pieces. On each interation the size of the array is divided by 2. The size of the list in the k iteration is then defined as $\frac{N}{2^k}$

Where - N is size of array - k iteration number

Since the size of the final array is 1. We then have $\frac{N}{2^k} = 1$. If we rearrange the terms and solve for k we have then

$2^k = N$

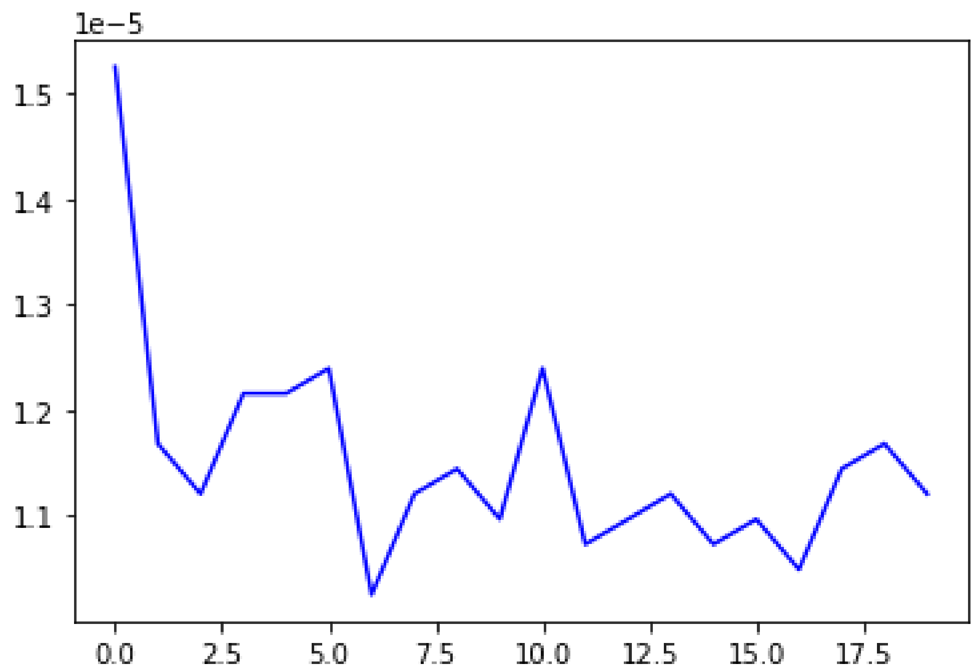
taking log_2 on both sides

$k = log_2(N)$

Therefore, the maxium number if iteration is $log(N)$ and the complexity for the algorith is $O(log(N))$

3.4 Simulation Results

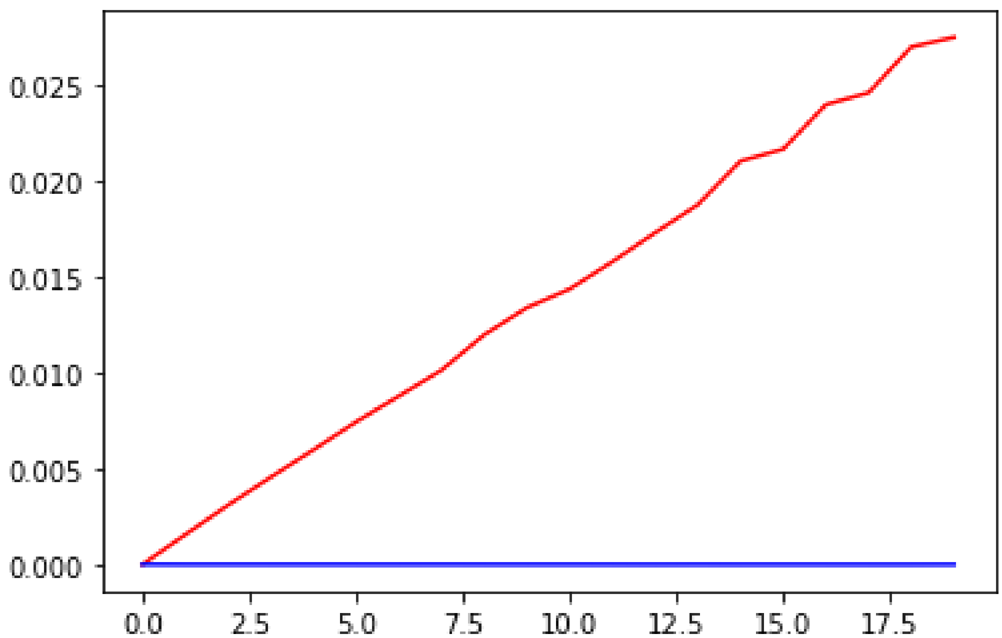
[23]:
```
import matplotlib.pyplot as plt
sim_number=10000000
increment=500000
ypoints_binary=simulation(find_number_binary,large_validation,sim_number,increment)
plt.plot(ypoints_binary, color = 'b')
plt.show()
```



3.5 Combined

[24]:
```
import matplotlib.pyplot as plt
sim_number=1000000
increment=50000

ypoints_lineal=simulation(find_number_linear,large_validation,sim_number,increment)
ypoints_binary=simulation(find_number_binary,large_validation,sim_number,increment)
plt.plot(ypoints_lineal,color='r')
plt.plot(ypoints_binary, color = 'b')
plt.show()
```



4 Building environment

4.1 Create virtual enviroment

```
mkdir ~/.virtualenvs
cd ~/.virtualenvs
python3 -m venv ads
. ~/.virtualenvs/ads/bin/activate
```

4.2 Install Libraries

```
python3 -m pip install jupyterlab
python3 -m pip install matplotlib

mkdir ~/ads
```

```
cd ~/ads
```

4.3 Execute Jupyter Lab

```
. ~/.virtualenvs/ads/bin/activate
cd ~/ads
jupyter lab
```