



Recitation2

Analysis of Algorithms



Question 1 (Theory):

Prove that running time $T(n) = n^2 + 20n + 1$ is $O(n^2)$

Need to show a pair of constants $\begin{Bmatrix} c \\ n_0 \end{Bmatrix}$, such that for all $n \geq n_0$, $T(n) \leq c * n^2$

$$n^2 + 20n + 1 \leq cn^2$$

Question 1 (Theory):

Prove that running time $T(n) = n^2 + 20n + 1$ is not $O(n)$

Need to show we can't find a pair of constants $\begin{Bmatrix} c \\ n_0 \end{Bmatrix}$, such that for all $n \geq n_0$, $T(n) \leq c * n$

Question 2 (Code snippet analysis)

```
1. def func1(N):  
2.     for i in range(N):  
3.         for j in range(N, 0, -2):  
4.             print("hi")
```

N times

0.5N times

Question 2 (Code snippet analysis)

```
1. def func2(N):  
2.     for i in range(N):  
3.         for j in range(N, 0, -2):  
4.             print("hi")  
5.  
6.     x = 0  
7.     while x < N:  
8.         x += 1  
9.         print("hiiii")
```

N times

0.5N times

N times

Question 2 (Code snippet analysis)

```
1. def func3(N):  
2.     i = 0  
3.     while i < N:  
4.         j = N  
5.         while j > 0:  
6.             j //= 2  
7.             print("hi")  
8.         i += 1
```

N times

logN times

Question 3 (Concept):

You have an N -floor building and plenty of eggs. Suppose that an egg is broken if it is thrown from floor F or higher, and unhurt otherwise.

1. Describe a strategy to determine the value of F such that the number of throws is at most $\log N$.

logN solution

N is the total number of floors. Thus, we can binary search total floors.

Let lowest = 1 and highest = N

Then repeat

Go to floor $k = (\text{highest} - \text{lowest}) / 2$

Throw egg and check result

If egg breaks then highest = k-1, otherwise lowest = k+1

until highest is inferior or equal to lowest

Question 3 (Concept):

You have an N -floor building and plenty of eggs. Suppose that an egg is broken if it is thrown from floor F or higher, and unhurt otherwise.

2. Find a new strategy to reduce the number of throws to at most $2 \log F$.

2logF solution

F is where egg just start to break.

Let $k = 0$

Repeat

- Go to floor 2^k

- Throw egg and check result

- If egg doesn't break then $k = k+1$

- until egg breaks

Finally, use binary search between floors 2^k and 2^{k+1}

Question 4 (Prime number):

A number is said to be prime if it is divisible by 1 and itself only, not by any third variable.

1. Divide N by every number from 2 to N - 1, if it is not divisible by any of them hence it is a prime.
2. Instead of checking until N, we can check until \sqrt{N} .

```
flag = True
for i in range(2, N):
    if N % i == 0:
        flag = False
return flag
```

Average runtime: $O(N)$

```
flag = True
for i in range(2, int(math.sqrt(N)) + 1):
    if N % i == 0:
        flag = False
return flag
```

Average runtime: $O(N^{\frac{1}{2}})$

Question 5 (permutation):

1. Fill the array **a** from **a[0]** to **a[N-1]** as follows: To fill **a[i]**, generate random numbers until you get one that is not already in **a[0]**, **a[1]**, \dots , **a[i-1]**.

- The Runtime for this algorithm is $O(n^2 \log n)$

```
def permutation1(array):  
    result = [None] * len(array)  
    for i in range(len(array)):  
        rand = random.randint(0, len(array) - 1)  
        while (rand in result):  
            rand = random.randint(0, len(array) - 1)  
        result[i] = rand  
    return result
```

O(N) checks for in operator

Need total $O(N \log N)$ random numbers

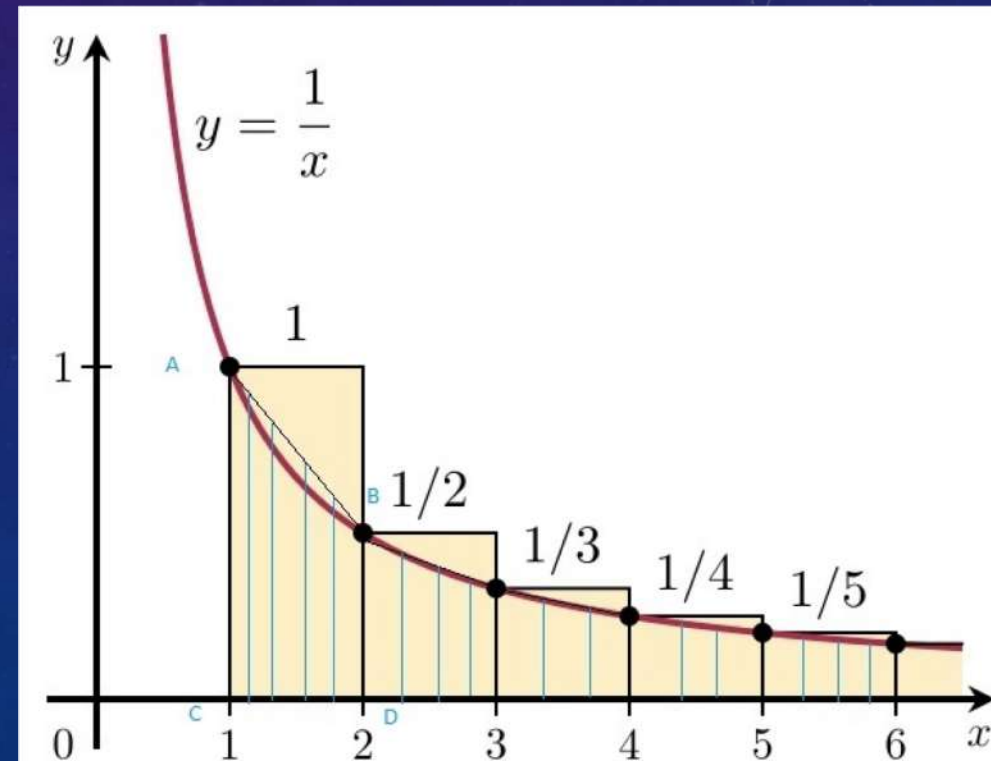
Why total $O(N \log N)$ computations for all the random numbers?

- Suppose we need 10 numbers
- First random number, the expected try time is $\frac{10}{10} = 1$ try.
- Second random number, the expected try time is $\frac{10}{9} = 1.11$ tries
- Third random number, the expected try time is $\frac{10}{8} = 1.25$ tries
- ...
- Last random number, the expected try time is $\frac{10}{1} = 10$ tries

- So we have a series of

$$\begin{aligned} & \frac{N}{N} + \frac{N}{N-1} + \frac{N}{N-2} + \dots + \frac{N}{1} \\ &= N \left(\frac{1}{N} + \frac{1}{N-1} + \frac{1}{N-2} + \dots + \frac{1}{1} \right) \end{aligned}$$

- The summation of $\left(\frac{1}{N} + \frac{1}{N-1} + \frac{1}{N-2} + \dots + \frac{1}{1} \right)$ is the integral (area under the curve)
- Integral of $y = \frac{1}{x}$ is:
- $y = \ln x + C$



Question 5 (permutation):

2. Same as algorithm (1), but keep an extra array called the **used** array. When a random number, **ran**, is first put in the array **a**, set **used[ran] = true**. This means that when filling **a[i]** with a random number, you can test in one step to see whether the random number has been used, instead of the (possibly) **i** steps in the first algorithm.

```
def permutation2(N):  
    result = [None] * N  
    extra_array = [False] * N  
    for i in range(N):  
        rand = random.randint(0, N - 1)  
        while (extra_array[rand] == True): • look up by index, O(1)  
            rand = random.randint(0, N - 1)  
        result[i] = rand  
        extra_array[rand] = True • for loop with random, O(nlogn)  
    return result
```

The Runtime for this algorithm is $O(n \log n)$

Question 5 (permutation):

3. Fill the array such that $a[i] = i$. Then:

```
for i in range(len(array)):
```

```
    swap( a[ i ], a[ randint( 0, i ) ] );
```

```
def permutation3(array):  
    result = []  
    for i in range(len(array)): • Fill the array, O(N)  
        result.append(i)  
    for i in range(len(array)): • O(N) swaps  
        random_index = random.randint(0, i)  
        result[i], result[random_index] = result[random_index], result[i]  
    return result
```

The Runtime for this algorithm is $O(N)$