Big O, Arrays And Strings

5th August, 2022

ANJALI VIRAMGAMA

Big O

- The big O notation is used to show how the runtime or space requirement of a function grows as the input grows.
- Time complexity is the rate of growth of time with the growth of input, and space complexity is rate of growth of space.
- Let's say you want to count numbers, and you take 1 second per number. If I ask you to count upto 10, you will take 10 seconds, upto 100 will take 100 seconds, and upto a number n will take n seconds. So the time complexity here is n seconds for any number n, and is shown as O(N)
- Similarly, if you need 1 parking slot for 1 car, 10 for 10 cars, you will need n for n cars and space complexity here will be O(N)
- It is extremely important to understand time and space complexities because they help you judge how good your solution is, and brainstorm about if and how you can make it better.

Big O from an interview POV

- There is a lot more to asymptotic notation than the big O, there's big-omega (lower bound) and big theta (tight bound).
- 99% of times, the only Big O question you'll ever be asked is "What is the big O of the solution that you just coded"

Drop all constants.

- All constants are dropped. O(2N) becomes O(N), O(20N +1000) becomes O(N).
- Here's a really good thread to understand why: https://stackoverflow.com/questions/22188851/why-is-the-constant-always-dropped-from-big-o-analysis
- The gist of it is "the constants are definitely significant, they do make a difference in the total time/space required in real life, but multiplying a function by a constant only influences its growth rate by a constant amount, so linear functions still grow linearly, logarithmic functions still grow logarithmically, exponential functions still grow exponentially. Since big-O notation only describes the long-term growth rate of functions, constants aren't relevant to asymptotic analysis and we drop them off”

Some example questions

```
public static void printNum(int x){
    System.out.println(x);
}
```

```java
public static void printNum3times(int x) {
    System.out.println(x);
    System.out.println(x);
    System.out.println(x);
}
```

Answer: O(1)

```java
public static void printUsingForLoop(int x) {
    for(int i = 0; i < x; i++){
        System.out.println(x);
    }
}
```

Answer to previous Ques: O(1)

```java
public static void printUsingForLoop2(int x) {
    for(int i = 0; i < 10; i++){
        System.out.println(x);
    }
}
```

Answer to previous Ques: O(N)

```java
public static void printUsingForLoop3(int x){
    for(int i = 0; i < 2*x; i++){
        System.out.println(x);
    }
    for(int i = 0; i < 20*x; i++){
    System.out.println(x);
    }
    for(int i = 0; i < x/2; i++){
    System.out.println(x);
    }
}
```

Answer to previous Ques: O(1)

```java
public static void printUsingForLoop3(int n, int m){
    for(int i = 0; i < 2*n; i++){
        System.out.println("Hello");
    }
    for(int i = 0; i < m/2; i++){
        System.out.println("World");
    }
}
```

Answer to previous Ques: O(N)

```java
public static void printUsingDoubleForLoop(int x){
    for(int i  = 0; i < x; i++){
        for(int j = 0; j < x; j++){
            System.out.println(x);
        }
    }
}
```

Answer to previous Ques: O(N+M)

```java
public static void printUsingTripleForLoop(int x){
    for(int i  = 0; i < x; i++){
        for(int j = 0; j < x; j++){
            for(int k = 0; k < x; k++){
                System.out.println(x);
            }
        }
    }
}
```

Answer to previous Ques: O(N^2)

```java
public static void printUsingTripleForLoop2(int x){
    for(int i  = 0; i < x; i++){
        for(int j = 0; j < 20; j++){
            for(int k = 0; k < x; k++){
                System.out.println(x);
            }
        }
    }
}
```

Answer to previous Ques: O(N^3)

```java
public static void practice(int N){
    for (i = 0; i < N; i++) {
        for (j = N; j > i; j--) {
            System.out.println("Hi");
        }
    }
}
```

Answer to previous Ques: O(N^2)

```java
public static void practice2(int N){
    for (int j = 2; j <= N; j = j * 2) {
        System.out.println("Hi");
    }
}
```

Answer to previous Ques: O(N^2)
any situation with 1+2+3+4+5...+N runs
N(N+1)/2 times

```java
public static void practice3(int n){
    for (int i = n / 2; i <= n; i++) {
        for (int j = 2; j <= n; j = j * 2) {
            System.out.println("Hi");
        }
    }
}
```

Answer to previous Ques: O(LogN) any loop that doubles* the variable, in other words halves the number at each step takes log amount of time.

***It could be doubles/triples/multiplies by x times, will lead to Log based x, which is still log since constant is dropped.**

```java
public static void practice4(int n){
    for (int i = N; i > 0; i /= 2){
        for (int j = 0; j < i; j++){
            System.out.println("Hi");
        }
    }
}
```

Answer to previous Ques: O(NlogN)

```java
public static int recursive1(int n)
{
    if (n < 0)
        return 0;
    else
        return 1 + recursive1(n-1);
}
```

Answer to previous Ques: O(N)

first loop = prints N times

second loop = prints N/2 times

third loop = prints N/4 times

N+N/2+N/4+.....= Geometric series =O(N)

```java
public static int recursive2(int n)
{
    if (n < 0)
        return 0;
    else
        return 1 + recursive2(n-10);
}
```

Answer to previous Ques: O(N)

```java
public static int recursive3(int n)
{
    if (n <= 0)
        return 1;
    else
        return 1 + recursive3(n/3);
}
```

Answer to previous Ques: O(N/10) = O(N)

```java
public static void recursive4(int n, int m, int o)
{
    if (n <= 0)
    {
        System.out.println("Hi");
    }
    else
    {
        recursive4(n-1, m+1, o);
        recursive4(n-1, m, o+1);
    }
}
```

Answer to previous Ques: O(Logn)

```java
public static int recursive5(int n)
{
    for (i = 0; i < n; i += 2) {
        for(int j = 0; j < n; j++){
            System.out.println("Hi");
        }
    }

    if (n <= 0)
        return 1;
    else
        return 1 + recursive5(n-5);
}
```

Answer to previous Ques: O(2^n)

Answer to previous Ques: O(n^3)

Arrays and strings

- Arrays and Strings are vastly underestimated, but most interviews I have given ask tricky questions from this topic, especially regarding string manipulation and matrices. Arrays and Strings will always be a subpart of the complex problems you'll solve later, and having a solid base here is necessary for success in the upcoming topics.
- One of the most important topics for programming interviews. Strings can be dealt with as char arrays
- Index starts from 0
- collection of items of the same type stored at contiguous memory locations.
- They have a specific length which is decided when they are first built
- Can be iterated in O(N)
- To fetch an element, you need to know it's index. Fetching takes O(1) time

Some things to think about when given an array question

- Is this array already sorted? (If yes, consider binary search)
- Can I sort it and get a solution faster?
- What is the range of elements in the array?
- Did I ensure there will be no out of bounds exception?
- Do I need to fetch values multiple times without knowing their indices? Can I keep track of values in O(1) space? (Consider using HashSet/HashMap)

Edge cases:

- Empty arrays
- Arrays with 1 or 2 elements
- Repeated elements
- If you encounter elements that are not allowed
- Out of bounds exception

Some string terminology you should be familiar with:

- String =”abcdef”
- Substring: Continuous part of string Eg: “ab”, “abcd”
- Subsequence: Not necessarily continuous but Order is maintained Eg: “acf”, “bde”

Let's get coding!

- Easy: Two Sum - LeetCode
- Easy: Remove Duplicates from Sorted Array – LeetCode
- Medium: Search in Rotated Sorted Array - LeetCode
- Medium: Longest Substring Without Repeating Characters - LeetCode
- Hard: Largest Rectangle in Histogram - LeetCode

Homework

- Easy: Fri, Sat
- Sorting the Sentence - LeetCode,
- First Missing Positive - LeetCode
- Medium: Sun, Mon
- Permutation in string
- Valid Sudoku – LeetCode

- Medium: Tues, Wed
- Palindromic String
- String to Integer
- Hard: optional for MAANG enthusiasts.
- Median of Two Sorted Arrays
- Edit Distance - LeetCode
- Scramble string

Hard problem solution

Please see next slides for solution of hard problem.

Screenshot is provided instead of code that you can copy-paste to ensure you type it out and understand it line by line.

It will be discussed again in the next class.

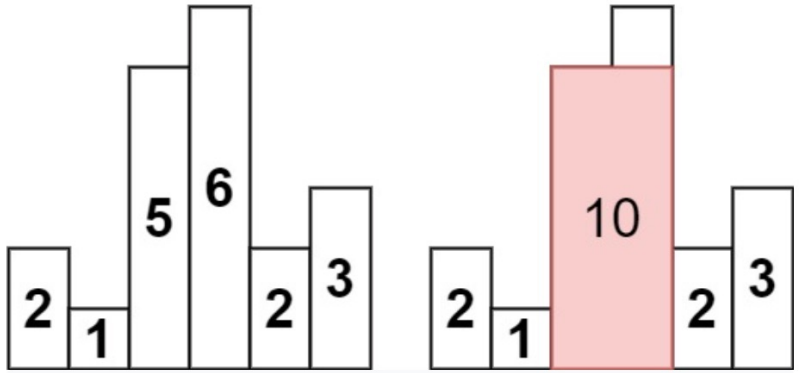
Thanks.

84. Largest Rectangle in Histogram

Hard 11541 163 Add to List Share

Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram*.

Example 1:



Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.

Example 2:

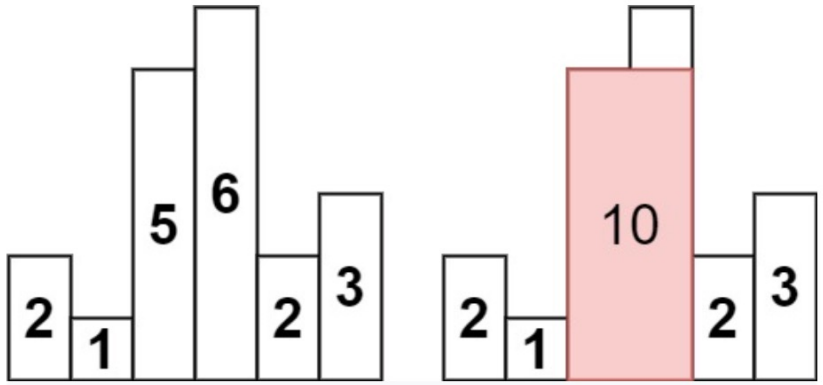
```
class Solution {
    public int largestRectangleArea(int[] heights) {
        //To make time O(N), use space to store what rec on left and right of curr have lesser height than curr.
        //Eg: Ht = [3,4,3], for curr = 4, index = 0 and index = 2 make its boundaries.
        //Max len for 4 = rt_index - lt_index - 1 = 2-0-1 = 1
        //Eg 2: Ht = [10, 8, 7, 9, 8], for curr = 7, rt_index = 4 and lt_index = -1
        //Max len = 4- (-1) -1 = 4

        /*
        Trick 2, lt_index for 7 = lt_index of 8 or less. If nothing is shorter than 8 on left, there will be nothing shorter than 7 on left. Hence lt_index = -1.
        On right, rt_index for 7 equal to or more than rt_index of 9. So it is equal to or beyond 8 (need to check beyond 8)
        */

/* Intuition behind time complextiy of O(N)
Consider the test case
indices.......... : 0 1 2 3 4 5 6 7 8 9 10 11
heights.......... : 4 9 8 7 6 5 9 8 7 6  5  _
lessFromLeft :    :-1 0 0 0 0 0 5 5 5 5  0
lessfromRight:    :11 2 3 4 5 11
In this, when we reach 5 at index 10, we start searching for idx=9, i.e. p points at 6.
6 > 5.
Now, we want something which is smaller than 5, so it should definitely be smaller than 6. So 6 says to 5:

I've already done the effort to find the nearest number that's smaller than me and you needn't traverse the array again till that point. My lessFromLeft points at index 5 and all the elements between that and me are greater than me so they are surely greater than you. So just jump to that index and start searching from there.

So you next p directly points at idx = 5, at value 5.

There, this 5 again says the same statement to current 5 and asks it to jump directly to idx = 0. So in the
```

84. Largest Rectangle in Histogram

Hard 11541 163 Add to List Share

Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram*.

Example 1:



Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.

Example 2:

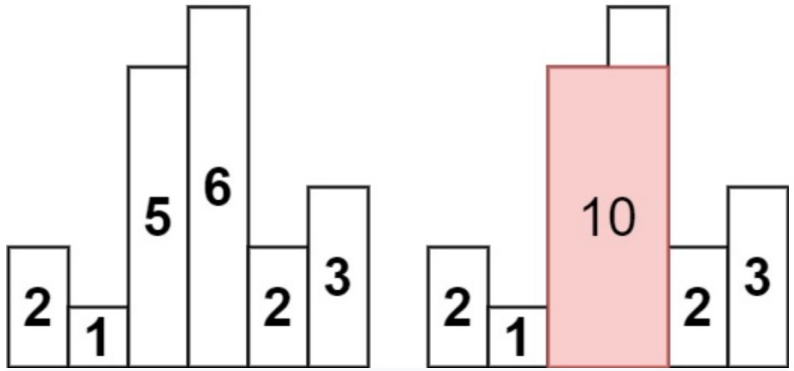
```
So you next p directly points at idx = 5, at value 5.

There, this 5 again says the same statement to current 5 and asks it to jump directly to idx = 0. So in the second iteration itself, our search has reached idx=0 and that's our answer for the current element.

Similarly, for the next element 4, it'll take 3 steps.

And for all the elements following 4, if they are greater than 4, their search will stop at 4 itself.

Bottomline: we are not traversing the array again and again. it's O(n).
*/
        if(heights.length==1){
            return heights[0];
        }
        int[] left = new int[heights.length]; //index of the first element of smaller ht than curr
        int[] right = new int[heights.length];
        left[0] = -1;
        right[heights.length-1]= heights.length;
        for(int i = 1; i < heights.length; i++){
            int temp = i-1; //let temp start from immediate left element
            while(temp>=0 && heights[temp]>=heights[i]){
                temp = left[temp];
            }
            left[i]= temp;
        }
        for(int i = heights.length-2; i>=0; i--){
            int temp = i+1; //let temp start from immediate right element
            while(temp<heights.length && heights[temp]>=heights[i]){
                temp = right[temp];
            }
            right[i]= temp;
        }
        //Done with left and right. Calculate max area
        int max = 0;
```

84. Largest Rectangle in Histogram

Hard 11541 163 Add to List Share

Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the largest rectangle in the histogram*.

Example 1:



Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.

Example 2:

```
        }
        //Done with left and right. Calculate max area
        int max = 0;
        for(int i = 0; i < heights.length; i++){
            max = Math.max(max,heights[i]*(right[i]-left[i]-1));
        }
        return max;

    }
}
   Time = O(N^2)
    public int largestRectangleArea(int[] heights) {
        //check left half and right half. O(N^2) time
        int max = 0;
        for(int i = 0; i < heights.length; i++){
            int count = 0;
            for(int j = i+1; j< heights.length; j++ ){
                //all recs taller than itself on the right side
                if(heights[j]>=heights[i]){
                    count++;
                }else{
                    break;
                }
            }
            for(int j = i-1; j>=0; j-- ){
                  //all recs taller than itself on the left side
                if(heights[j]>=heights[i]){
                    System.out.println(j);
                    count++;
                }else{
                    break;
                }
            }
            max = Math.max((count+1)*heights[i],max);
        }
        return max;
    }
```

Disclaimer

All content and material is copyrighted material belonging to Anjali Viramgama and is purely for the dissemination of education. You are permitted to access print and download extracts from this site purely for your own education only and on the following basis:-

- You can download this document from the provided link for self use only.
- Any copies of this document, in part or full, saved to disc or to any other storage medium may only be used for subsequent, self viewing purposes or to print an individual extract or copy for non commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of content for any other commercial/unauthorized purposes in any way which could infringe the intellectual property rights of its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document may be reproduced or stored in any other web site or included in any public or private electronic retrieval system or service the contributors' prior written permission.
- Any rights not expressly granted in these terms are reserved.