Complexity, Array and Set

Wei Zhang
08/09/2018

- Data structure and Algorithm
- Time and Space Complexity
- Array
- Set
- Homework

Data Structure and Algorithm

Data strucrue:

- **A collection of data values**
- **The relationships among them**
- **The functions or operations that can be applied to them.**

1. Array
2. Set
3. Lined list
4. Stack
5. Queue
6. Heap
7. Hash map & hash table
8. Tree (BST, AVL, red and black, etc.)
9. Graph

…...

Algorithm:

- **Mathematical abstraction of computer program**
- **Computational procedure to solve a problem**

1. Divide and conquer
2. Search
3. Sorting (merge sort, insertion sort, quick sort, etc.)
4. Backtracking
5. Dynamica programing
6. Topological sort
7. Breadth/ Depth first search

…...

Time and Space Complexity: 3 notations

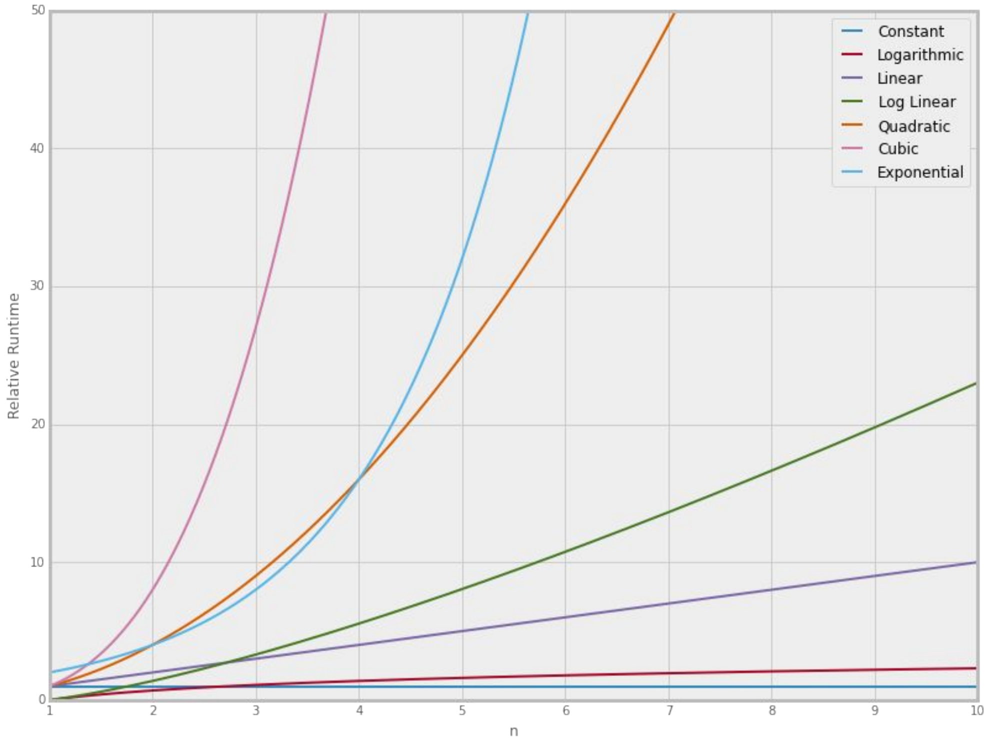
- what operations an algorithm is allowed ?
- cost (time, space, . . .) of each operation ?
- cost of algorithm = sum of operation costs ?

3 most important notations in algorithm complexity:

1. O(g(n)): asymptotic upper bound of f(n): there exist positive constant c and n0 such that
$$0 \leq f(n) \leq cg(n) \quad \forall n \geq n_0$$
2. $\Theta(g(n))$: asymptotic tight bound of f(n): there exist positive constanc c1, c2, n0 such that
$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$
3. $\Omega(g(n))$: asymptotic lower bound of f(n): there exist positive constanc c and n0 such that
$$0 \leq \Omega(g(n)) \leq f(n)$$

Time and Space Complexity

Big-O notation describes *how quickly runtime will grow relative to the input as the input get arbitrarily large*.



Big-O	**Name**
1	Constant
log(n)	Logarithmic
n	Linear
nlog(n)	Log Linear
n^2	Quadratic
n^3	Cubic
2^n	Exponential

Time and Space Complexity: Example 1

Example 1: Given an array of size *n*, find the majority element. The majority element is the element that appears more than ⌊ `n/2` ⌋ times.
You may assume that the array is non-empty and the majority element always exists.

Solution(1): randomization

```
import random

class Solution:
    def majorityElement(self, nums):
        majority_count = len(nums)//2
        while True:
            candidate = random.choice(nums)
            if sum(1 for elem in nums if elem == candidate) > majority_count:
                return candidate
```

- Time complexity : $O(\infty)$
- Space complexity : $O(1)$

!!!

Does that mean this approach won't work?

$$
\begin{aligned}
EV(iters_{prob}) &\le EV(iters_{mod}) \\
&= \lim_{n\to\infty} \sum_{i=1}^{n} i \cdot \frac{1}{2^i} \\
&= 2
\end{aligned}
$$

Time and Space Complexity: Master Theorem

Example 1 --------Solution(2): divide and conquer

```
def majorityElement(self, nums, lo=0, hi=None):
    def majority_element_rec(lo, hi):
        # base case; the only element in an array of size 1 is the majority
        # element.
        if lo == hi:
            return nums[lo]

        # recurse on left and right halves of this slice.
        mid = (hi-lo)//2 + lo
        left = majority_element_rec(lo, mid)
        right = majority_element_rec(mid+1, hi)

        # if the two halves agree on the majority element, return it.
        if left == right:
            return left

        # otherwise, count each element and return the "winner".
        left_count = sum(1 for i in range(lo, hi+1) if nums[i] == left)
        right_count = sum(1 for i in range(lo, hi+1) if nums[i] == right)

        return left if left_count > right_count else right

    return majority_element_rec(0, len(nums)-1)
```

- Time complexity:

$$T(n) = 2T(n/2) + 2n = 4T(n/4) + 2n * 2$$
$$= nT(1) + 2n * log_2 n = n + 2nlgn$$
$$= \Theta(nlgn)$$

- Space complexity: $\Theta(lgn)$ for cuts

Master theorem:

$$T(n) = a\,T\left(\frac{n}{b}\right) + f(n), \quad c_{\text{crit}} = \log_b a$$

1) When $f(n)=n^c$, where $c<c_{crit}$: $T(n)=\Theta(n^c)$

2) When $f(n)=\Theta(n^{Ccrit}lg^k n)$, where $k>=0$: $T(n)=\Theta(n^{Ccrit}lg^{k+1}n)$

3) When $f(n)=n^c$, where $c>c_{crit}$: $T(n)=\Theta(f(n))$

Time and Space Complexity: Example 1

Example 1 --------Solution(3): Boyer-Moore Voting Algorithm

```python
class Solution:
    def majorityElement(self, nums):
        count = 0
        candidate = None

        for num in nums:
            if count == 0:
                candidate = num
            count += (1 if num == candidate else -1)

        return candidate
```

- Time complexity : O(n)
- Space complexity : O(1)

Time and Space Complexity: big O for DS

Common Data Structure Operations

http://bigocheatsheet.com/

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

Array: Static and Dynamic

- C/C++:

  int a[10];   int *a = (int*) malloc(10);

- C++ only:

  int *a = new int [10];

static

std::vector<int> a (10, 0);   std::array<int, 10> a;

- Python:

  a=list();   a=[]

Dynamic
How to grow when capacity is used up?

$m += 1?$

$\Longrightarrow$ rebuild every step

$\Longrightarrow$ n inserts cost $\Theta(1 + 2 + \cdots + n) = \Theta(n^2)$

NO !!!

Correct answer is: Table doubling

Array: table doubling & amortized analysis

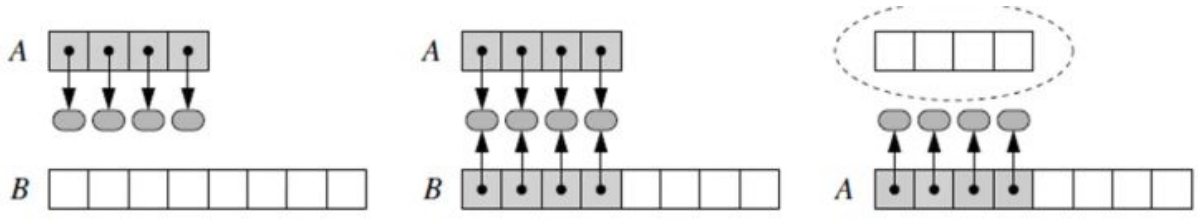
Table doubling:

- $m *= 2$? $m = \Theta(n)$ still ($r+= 1$)
  $\Longrightarrow$ rebuild at insertion 2^i
  $\Longrightarrow$ n inserts cost $\Theta(1 + 2 + 4 + 8 + \cdots + n)$ where n is really the next power of 2
  $= \Theta(n)$

- a few inserts cost linear time, but $\Theta(1)$ "on average". $\Leftarrow$ **Amortized analysis**

- operation has <u>amortized cost</u> $T(n)$ if k operations cost $\leq k \cdot T(n)$

- "$T(n)$ amortized" roughly means $T(n)$ "on average", but averaged over all ops.



(a) (b) (c)

Similar for Delete:
when n decreases to m/4, shrink to half the size
O(1) amortized cost for both insert and delete

Array: Python list operation

(a) L.append(x) $\rightarrow \theta(1)$ time via *table doubling*

(b) $\underbrace{L = L1 + L2}_{(\theta(1+|L1|+|L2|)\text{ time})} \equiv L = [\,] \rightarrow \theta(1)$

for x in $L1$:
  L.append(x) $\rightarrow \theta(1)$ } $\theta(|L_1|)$

for x in $L2$:
  L.append(x) $\rightarrow \theta(1)$ } $\theta(|L_2|)$

(c) $L1$.extend($L2$) $\equiv$ for x in $L2$:
$\equiv L1 += L2$ $L1$.append(x) $\rightarrow \theta(1)$ } $\theta(1 + |L_2|)$ time

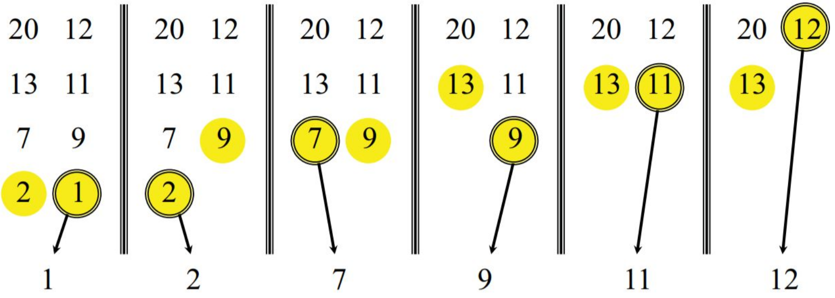
(d) $L2 = L1[i:j] \equiv L2 = [\,]$
for k in range(i, j):
  $L2$.append$(L1[i]) \rightarrow \theta(1)$ } $\theta(j - i + 1) = O(|L|)$

(f) len(L) $\rightarrow \theta(1)$ time - list stores its length in a field (g) L.sort() $\rightarrow \theta(|L| \log |L|)$ - via *comparison sort*

Array: comparison sorting (1)

(1): Merge Sort

MERGE-SORT $A[1 \,..\, n]$	$T(n)$
1. If $n = 1$, done.	$\Theta(1)$
2. Recursively sort $A[\,1 \,..\, \lceil n/2 \rceil\,]$ and $A[\,\lceil n/2 \rceil + 1 \,..\, n\,]$.	$2T(n/2)$
3. *"Merge"* the two sorted lists	$\Theta(n)$



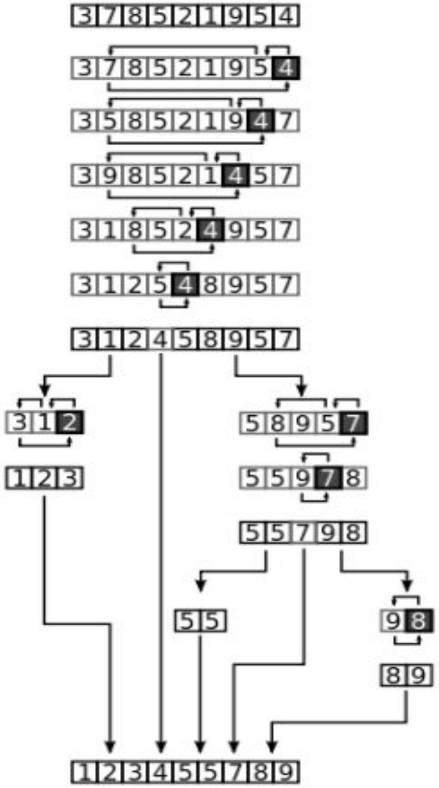

```python
def merge(arr,l,m,r):
    left=arr[l:m+1]
    right=arr[m+1:r+1]
    pl,pr,p=0,0,l
    while pl<len(left) and pr<len(right):
        if left[pl]<=right[pr]:
            arr[p]=left[pl]
            pl+=1
        else:
            arr[p]=right[pr]
            pr+=1
        p+=1
    if pl<len(left):
        arr[p:r+1]=left[pl:m+1]
    if pr<len(right):
        arr[p:r+1]=right[pr:r+1]

def merge_sort(arr,l,r):
    if l<r:
        m=(r-l)//2+l
        merge_sort(arr,l,m)
        merge_sort(arr,m+1,r)
        merge(arr,l,m,r)
```

Array: comparison sorting (2)

(2): Quick Sort

```
def quick_sort(arr,l,r):
    if l<r:
        i,pivot=l,r
        while i<pivot:
            if arr[i]<=arr[pivot]:
                i+=1
                continue
            arr[i],arr[pivot-1],arr[pivot]=arr[pivot-1],arr[pivot],arr[i]
            pivot-=1
        quick_sort(arr,l,pivot-1)
        quick_sort(arr,pivot+1,r)
```



Worst-case analysis: O(n^2)

The most unbalanced partition occurs when one of the sublists returned by the partitioning routine is of size $\sum_{i=0}^{n}(n-i) = O(n^2)$

Average-case analysis: O(nlgn)

Refer to: https://en.wikipedia.org/wiki/Quicksort

Array: comparison sorting (3)

(3): Insertion Sort

INSERTION-SORT (A, n) $\quad \triangleright A[1 \,..\, n]$

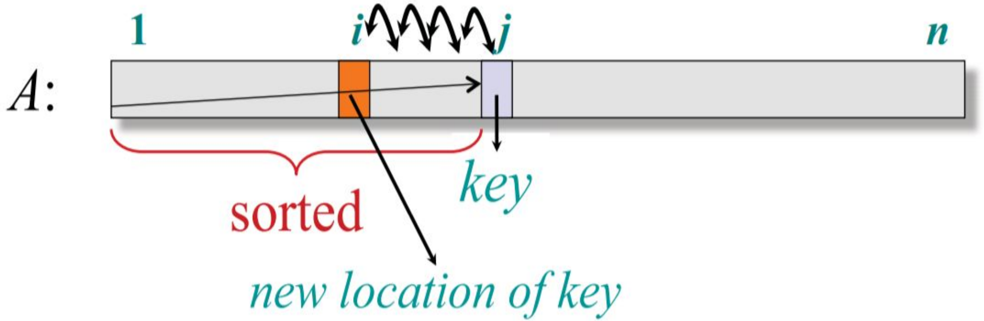
for $j \leftarrow 2$ **to** n

insert key A[j] **into the (already sorted) sub-array** A[1 .. j-1]

by pairwise key-swaps down to its right position

```python
def insert_sort(arr):
    for i in range(1,len(arr)):
        for j in range(i-1,-1,-1):
            if arr[i]>=arr[j]:
                break
            arr[i],arr[j]=arr[j],arr[i]
            i-=1
```

Illustration of iteration *j*



Time Complexity: $\Theta(n^2)$

What if using binary insertion?

Array: Example 2

Example 2: 3 sum closest

Given an array `nums` of *n* integers and an integer `target`, find three integers in `nums` such that the sum is closest to `target`. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example:

```
Given array nums = [-1, 2, 1, -4], and target = 1.

The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

```python
def threeSumClosest(self, nums, target):
    nums.sort()
    length=len(nums)
    diff=float('inf')
    for i,x in enumerate(nums[:-2]):
        l=i+1
        r=length-1
        while l<r:
            temp=x+nums[l]+nums[r]-target
            if abs(temp)<diff:
                tot=temp+target
                diff=abs(temp)
            if temp>0:
                r-=1
            elif temp<0:
                l+=1
            else:
                return tot
    return tot
```

Time complexity: $O(n^2)$
Extra space: O(1)

Array: Example 3

Example 3: Find All Duplicates in an Array

Given an array of integers, 1 ≤ a[i] ≤ *n* (*n* = size of array), some elements appear **twice** and others appear **once**.

Find all the elements that appear **twice** in this array.

Could you do it without extra space and in O(*n*) runtime?

Example:

```
Input:
[4,3,2,7,8,2,3,1]

Output:
[2,3]
```

```python
def findDuplicates(self, nums):
    """
    :type nums: List[int]
    :rtype: List[int]
    """
    res=[]
    for val in nums:
        if nums[abs(val)-1]<0:
            res.append(abs(val))
            continue
        nums[abs(val)-1]*=(-1)
    return res
```

Set

Definition: A unordered collection of unique and immutable objects, implemented by hashing.

C++:
std::set<int> s: implemented as balanced tree
std::unordered_set<int> s: implemented as hash table

Python:
s=set() : mutable
s=frozenset(): immutable

```
a=set("hello,world")
b=set([1,2,3])
c=set([a,b])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-33-b9303dec820d> in <module>()
      1 a=set("hello,world")
      2 b=set([1,2,3])
----> 3 c=set([a,b])

TypeError: unhashable type: 'set'
```

```
a=frozenset("hello,world")
b=frozenset([1,2,3])
c=set([a,b])
c
```

```
{frozenset({1, 2, 3}), frozenset({',', 'd', 'e', 'h', 'l', 'o', 'r', 'w'})}
```

More details about hashable and immutable please refer to:

https://stackoverflow.com/questions/2671376/hashable-immutable

Set: Example 4

Example 4: **3 Sum**

Given an array `nums` of *n* integers, are there elements *a*, *b*, *c* in `nums` such that *a* + *b* + *c* = 0? Find all unique triplets in the array which gives the sum of zero.

Note:

The solution set must not contain duplicate triplets.

Example:

```
Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

```python
def threeSum(self, nums):
    if len(nums) < 3:
        return []
    nums.sort()
    res = set()
    for i, a in enumerate(nums[:-2]):
        if i >= 1 and a == nums[i-1]:
            continue
        check = set()
        for b in nums[i+1:]:
            if b not in check:
                check.add(-a-b)
            else:
                res.add((a, b, -a-b))
                check.remove(b)
    return list(map(list, res))
```

Time Complexity: $O(n^2)$
Extra space: $O(n)$

Home work

- Buy and Sell Stocks

1. LC 121 Best Time to Buy and Sell Stock
2. LC 122 Best Time to Buy and Sell Stock II
3. LC 123 Best Time to Buy and Sell Stock III

- Extension of Example 3

4. LC 448 Find All Numbers Disappeared in an Array
5. LC 41 First Missing Positive

- Extension of Example 4

6. LC 18 4Sum

- Extension of Example 1

7. LC 229 Majority Element II

- Transform 2D array in-place

8. LC 73 Set Matrix Zeroes
9. LC 289 Game of Life

- Array manipulation (Huge array)

10.

You are given a list(1-indexed) of size n, initialized with zeroes. You have to perform m operations on the list and output the maximum of final values of all the n elements in the list. For every operation, you are given three integers a, b and k and you have to add value k to all the elements ranging from index a to b (both inclusive).

Consider a list a of size 3, the initial list would be $a = [0, 0, 0]$ and after performing the update $2\ 3\ 30$, the new list would be $a = [0, 30, 30]$.

Input Format

The first line will contain two integers n and m separated by a single space.
Next m lines will contain three integers a, b and k separated by a single space.
Numbers in list are numbered from 1 to n.

Constraints

- $3 \leq n \leq 10^7$
- $1 \leq m \leq 2 * 10^5$
- $1 \leq a \leq b \leq n$
- $0 \leq k \leq 10^9$

Output Format

Print in a single line the maximum value in the updated list.

Thank you!