Data Structure and Algorithm HW1

R11522709 機械所 碩一 石翊鵬

April 13, 2023

1. What if you became a DSA TA?

Proof of conjecture 1

Consider $f(n) \geq 0$ and $g(n) \geq 0$ that $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, where $c \in R$
To prove $\lim_{n\to\infty} \frac{f(n)}{g(n)} \geq 0$ for all $n \in R$, we can consider h(n) = 0
Since $f(n) \geq 0$ and $g(n) \geq 0$ for all $n \in R$, $\frac{f(n)}{g(n)} \geq h(n)$ for all n
therefore,

$$\lim_{n\to\infty} h(n) \geq \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$\lim_{n\to\infty} h(n) = 0 \Rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \geq 0$$

By difinition of limit, there exists $\epsilon > 0$, $n_0 > 0$ such that for all $n > n_0$,

$$|\frac{f(n)}{g(n)} - c| < \epsilon$$

$$-\epsilon < \frac{f(n)}{g(n)} - c < \epsilon$$

$$-\epsilon + c < \frac{f(n)}{g(n)} < \epsilon + c$$

In case that $0 < c < \epsilon$

$$-\epsilon + c < 0 < \frac{f(n)}{g(n)} < \epsilon + c$$

In case that $c \geq \epsilon$

$$0 \leq -\epsilon + c < \frac{f(n)}{g(n)} < \epsilon + c$$

$$0 \leq f(n) < (\epsilon + c)g(n)$$

Let $(\epsilon + c) = c_1$

$$0 \leq f(n) \leq c_1 g(n)$$

which shows $f(n) = O(g(n))$, QED

1.1 - all by myself

Consider $c_1, c_2 > 0$ for all $n > 0$
if

$$\frac{c_1}{c_2}n \geq 1$$

then

$$c_1 n^3 \geq c_2 n^2$$

let

$$n_0 = \frac{c_2}{c_1}$$

$$\frac{c_1}{c_2}n_0 = 1$$

consider $k > 0$

$$\frac{c_1}{c_2}(n_0 + k) = 1 + \frac{c_1}{c_2}k$$

where

$$\frac{c_1}{c_2}k > 0$$

that is, for all $n \geq n_0$,

$$0 \leq c_2 n^2 \leq c_1 n^3$$

therefore, if

$$0 \leq f(x) \leq c_3 n^2$$

for all $n > n_1$ where $n_1, c_3 > 0$
then

$$0 \leq f(n) \leq c_3 n^2 \leq \frac{c_3 c_1}{c_2} n^3$$

for all $n > max(n_0, n_1)$, QED

1-2 - all by myself

Let $t_m = the\ number\ of\ while\ checks$, $k = the\ index\ of\ key$, $d_n = timecostofeachlinetoexecuteonce$
Since the value of m is assigned to l in the beginning of each while loop, m will start from 1 and increase by 1 when while loop is executed once, moreover, $A[m] \leq key$ will always be satisfied.
therefore,

$$t_m = k$$

in worst case, key is not in the array, in this case

$$T(n) = d_1 n + d_2(n-1) + d_3(n-1) + d_4 * 0 + d_5(n-1) + d_6 * 0 + d_7(n-1) + d_8(n-1) + d_9$$

$$T(n) = c_1 n + c_2$$

where c_1, $c_2 > 0$

$$\lim_{n\to\infty} \frac{c_1 n + c_2}{n} = c_1$$

By conjecture 1, time complexity of the algorithm is $O(n)$

1-3

ref: `https://math.stackexchange.com/questions/925053/using-limits-to-determine-big-o-big-omega-and-big-theta#comment6149810_925053`

$$f(n) = \Theta(n^2) \iff there\ exists\ positive\ (n_0, c_1, c_2)\ such\ that\ c_1 n^2 \leq f(n) \leq c_2 n^2\ for\ all\ n \geq n_0$$

Let $c_1 = 1$, $c_2 = 10$ and $f(n) = n^2(sin(n) + 2) + n$
for all $n > 0$, $n^2 \leq f(n) \leq 10n^2$
Suppose

$$\lim_{n \to \infty} f(n) = L$$

where $L \in R$, then there exist $\epsilon > 0$, $n > 0$ such that for all $n > n_0$,

$$|f(n) - L| < \epsilon$$

Let $f(n_1) = \epsilon_1$ where $n_1 > n_0$ and $\epsilon_1 < \epsilon$

$$f(n_1 + 2\pi k) - f(n_1) = ((n_1 + 2\pi k)^2 - n_1^2)(sin(n_1) + 2) + 2\pi k$$

where $k \in n$

$$\lim_{k \to \infty} f(n_1 + 2\pi k) - f(n_1) \neq 0$$

$$\Rightarrow |f(n + 2\pi k) - L| > \epsilon$$

for some $k > k_0$ where $k_0 \in n$
That is, $\lim_{n \to \infty} \frac{f(n)}{n^2}$does not exists, therefore, the proposition is WRONG.

1-4 - all by myself

$$lg(n) = 2lg(\sqrt{n}) = 2\frac{ln(\sqrt{n})}{\ln(2)}$$

Consider

$$\lim_{n \to \infty} \frac{lg(n)}{\sqrt{n}} = \lim_{n \to \infty} \frac{2}{ln(2)} \frac{ln(\sqrt{n})}{\sqrt{n}}$$

Let $\sqrt{n} = x$

$$\lim_{n \to \infty} \frac{2}{ln(2)} \frac{ln(\sqrt{n})}{\sqrt{n}} = \lim_{x \to \infty} \frac{2}{ln(2)} \frac{ln(x)}{x}$$

By L'Hopital's rule

$$\lim_{x \to \infty} \frac{2}{ln(2)} \frac{ln(x)}{x} = \lim_{x \to \infty} \frac{2}{ln(2)} \frac{\frac{d}{dx} ln(x)}{\frac{d}{dx} x} = \lim_{x \to \infty} \frac{2}{ln(2)} \frac{\frac{1}{x}}{1} = 0 \in R$$

By conjecture 1, $lg(n) = O(\sqrt{n})$, QED

1-5 - all by myself

Consider

$$\frac{\lim_{n\to\infty}\Sigma_{i=1}^{n} i^n}{\lim_{n\to\infty}\Sigma_{i=1}^{n} n^n} = \lim_{n\to\infty}\Sigma_{i=1}^{n}(\frac{i}{n})^n$$

and

$$\lim_{n\to\infty}\int_1^n (\frac{i}{n})^n di$$

and

$$\lim_{n\to\infty}\int_0^{n-1} (\frac{i}{n})^n di$$

By definition of Σ,

$$\lim_{n\to\infty}\int_0^{n-1} (\frac{i}{n})^n di < \lim_{n\to\infty}\Sigma_{i=1}^{n}(\frac{i}{n})^n < \lim_{n\to\infty}\int_1^n (\frac{i}{n})^n di$$

$$\lim_{n\to\infty}\int_1^n (\frac{i}{n})^n di = \lim_{n\to\infty}\frac{1}{n^n}\int_1^n i^n di = \lim_{n\to\infty}\frac{1}{n^n}\ [i^n]_1^n = \lim_{n\to\infty}\frac{n^n}{n^n} - \lim_{n\to\infty}\frac{1^n}{n^n} = 1 - 0 = 1$$

$$\lim_{n\to\infty}\int_0^{n-1} (\frac{i}{n})^n di = \lim_{n\to\infty}\frac{1}{n^n}\int_0^{n-1} i^n di = \lim_{n\to\infty}\frac{1}{n^n}\ [i^n]_0^{n-1} = \lim_{n\to\infty}\frac{(n-1)^n}{n^n} - \lim_{n\to\infty}\frac{1^n}{n^n} = 1 - 0 = 1$$

By squeeze theorem,

$$\lim_{n\to\infty}\Sigma_{i=1}^{n}(\frac{i}{n})^n = 1 \in R$$

By conjecture 1, $\Sigma_{i=1}^{n} i^n = O(n^n)$

1-6 - all by myself

In second last line, since c can be positive or negative, it is possible that $\frac{1}{2^c} > 2^c$ and therefore $f(n) > \frac{1}{2^c}$
The correct proof:

$$|lg(f(n)) - lg(g(n))| = c \Rightarrow lg(\frac{f(n)}{g(n)}) = c \Rightarrow \frac{f(n)}{g(n)} = 2^c$$

$$\Rightarrow |lg(f(n)) - lg(g(n))| = c \Rightarrow lg(\frac{f(n)}{g(n)}) = c_1 \Rightarrow \frac{f(n)}{g(n)} = 2^{c_1}$$

where $c_1 = \pm c$
Take $c' = 2^c - 1$, we have $f(n) \leq c'g(n)$ for all $n > n_0$, QED

2. DSA Judge

2-1

ref: chatGPT
psuedo code:

procedure FINDLOST(A, n)
  for i from 0 to 1 **do**
    $m = n/2$
    $l = 0$
    $r = n$
    while $(r - l) > 1$ **do**
      if $A[m] < m + (2 - k)$ **then**
        $l = m$
      else
        $r = m$
      end if
    end while
  end for
  $max(r_1, r_2) + = 1$
  return r_1, r_2
end procedure

與 binary sort 相似，將 m 定為 array 的中心點，若 $A[m] > m$，則代表在 m 的前面有資料遺失，將上界定為 m，若否則將下界定為 m，反覆迭代後資料遺失的位置會位於上界 Time complexity is almost the same as binary sort in worst case, the code will be run $lg(n)$ times, the time complexity $T(n) = O(lg(n))$

2-2 - all by myself

procedure COUPLEDOUBLE(A,n)
    $isCouple = n - 1$
    for i from 0 to $n - 1$ **do**
        if $A[isCouple] == 2 * A[n - 1 - i]$ **then**
            $A[isCouple] = 0$
            $A[n - 1 - i] = 0$
            $isCouple = isCouple - 1$
            while $A[isCouple] == 0$ **do**
                $isCouple = isCouple - 1$
            end while
        end if
    end for
    if $isCouple == 0$ **then**
        $return\ True$
    else
        $return\ False$
    end if
end procedure

從 $A[n-1]$ 遍歷到 $A[0]$，$A[IsCouple]$ 代表" 下一個遍歷到的數字，如果是配對中較小的一項，應該要與其配對的大項"，如果配對成功，就將兩數歸零，因此 $A[IsCouple]$ 若遇到已配對的小項便會跳過，最後若 $IsCouple = 0$ 代表所有項皆配對成功，反之則否

The for loops will run no more than n times, so the time complexity is $O(n)$, and the space complexity is O(1)

2-3 - all by myself

```
procedure ReverselySort(root1, root2)
    if root1.ID < root2.ID then
        newNext = root1
    else
        newNext = root2
    end if
    while root1.next or root2.next != NIL do
        if root2.ID < root1.ID then
            switch(root1, root2)
        end if
        if root1.next != NIL then
            newPrev = *root1.next
            *root1.next = newNext
            newNext = root1
            root1 = newPrev
        else
            *root1.next = newNext
            newNext = root1
            while *root2.next != NIL do
                newPrev = *root2.next
                *root2.next = newNext
                newNext = root1
                root2 = newPrev
            end while
        end if
    end while
end procedure
```

比較兩個 linked list 中較小的項，先將該項原本的 $next$ 存進 $newPrev$ 避免下家遺失，並將該項的 $next$ 指向 $newNext$，再將較小的項指定為新的 $newNext$，如此不斷迭代，而如果其中一條 linked list 已經全數存數新的 list，就將剩下的 list 反轉

The while loops will run no more than n times, so the time complexity is $O(n)$, and the space complexity is O(1)

2-4 - all by myself

procedure ReverselySort($node$)
  while $node.next! = NIL$ **do**
    if $node.a_i >= 0$ **then**
      $*positiveTail.next = node$
      $positiveTail = node$
    else
      $*node.next = negativeHead$
      $negativeHead = node$
    end if
  end while
  $*positiveTail.next = negativeHead$
end procedure

從 head 遍歷至 tail，建立新的 linked list，若遇到正數則將旗下家指向暫時設定的 tail 並暫時將其設為新的 tail，若遇到負數則斷開連結並將下家指向目前遇到的最後一個負數，因為負數只會越來越大而正數只會越來越小，因此關係得以成立，最後再將最校的正數指向最大的負數
The while loops will run no more than n times, so the time complexity is $O(n)$, and the space complexity is O(1)

Problem 3 - Stack / Queue

3-1 - all by myself

$$5\ 3\ 4 \times +8\ 2\ 3 + \times 5\ 1\ 9\ 6 \times +/ - -$$

Convert the operations of the highest priorty at first, such as those with parentheses, then put the operators of the second highest priorty behind it and put the addend(subtrahend, multiplicand, dividend) at the forefront, then the thrid priorty and so on.

3-2 - all by myself

$$(5 + 2 \times 7) - ((6 \times 3 - 4) - ((4 + 8 \times 6)/4))$$

Traverse all the operators from left to right, and convert it into

$$the\ number\ in\ front\ of\ the\ front\ of\ it\ (the\ operator)\ the\ number\ in\ front\ of\ it$$

3-3 - all by myself

Yes, we can move all elements to stack 2 and sort them to the ascending order by:

Pop element 4 and push it to stack 2
Pop element 1 and push it to stack 1
Pop element 2 and push it to stack 1
Pop element 3 and push it to stack 2
Pop element 2 and push it to stack 2
Pop element 1 and push it to stack 2

3-4 - all by myself

-Divide all the elements in stack 0 into m groups in which all the elements is ascending or descending order
-Then find the smallest elements a_m of each group
-For all a_m, if there are more than one a with larger value on the top of it, it will be impossible to move all the elements to stack 2 and sort it in ascending order, otherwise, it will be possible.

3-5 - all by myself

Each time when someone wants to park a bike, ze will park zer bike in the largest space, the first one will park between 0 and 10, the second one will park between 10 and 15.5, and the third one will park between 0 and 5 or 5 and 10

3-6 - all by myself

procedure INSERTBIKE(m, n, A)
  for $i\ from\ 2\ to\ n$ **do**
    $Enqueue(Q_1, A[i] - A[i-1])$
  end for
  $Bike[0] = Q_1.head/2$
  $Enqueue(Q_2, Q_1.head/2)$
  $Dequeue(Q_1)$
  for $i\ from\ 2\ to\ m$ **do**
    if $Q_1.head > Q_2.head$ or $Q_1 = NIL$ **then**
      $Bike[i] = Bike[i-1] + Q_1.head/2$
      $Enqueue(Q_2, Q_1.head/2)$
      $Dequeue(Q_1)$
    else
      $Bike[i] = Bike[i-1] + Q_2.head/2$
      $Enqueue(Q_2, Q_2.head/2)$
      $Dequeue(Q_2)$
    end if
  end for
end procedure

Time complexity

$$T(m+n) = nd_2 + (n-1)d_3 + d_5 + d_6 + d_7 + md_8 + (m-1)d_9 + (m-1)(d_{10} + d_{11} + d_{12})$$

where d_n is the time consumed by each line
Take $c_1 = max((d_2 + d_3), (d_8 + d_9 + d_{10} + d_{11} + d_{12}))$ and $c_2 = d_5 + d_6 + d_7 - d_3 - d_9 - d_{10} - d_{11} - d_{12}$

$$T(m+n) = c_1(m+n) + c_2$$

for all $m \geq m_0$ and $n \geq n_0$, there exists $c \geq c_1 + \frac{c_2}{m+n}$ such that

$$T(m+n) \leq c(m+n)$$

$$\Rightarrow T(m+n) = O(m+n)$$

Space complexity
array A: pointer size s_1
integer m: size s_2
integer n: size s_2
array Bike: pointer size $s_1 + m\times$integer size s_2
Queue Q_1: $n\times$pointer size $s_1 + n\times$integer size s_2
Queue Q_2: $m\times$pointer size $s_1 + m\times$integer size s_2

$$S(m+n) = (2+m+n)s_1 + (2+2m+n)s_2$$

Take $c = 2 + 2m + n$
for all sets of m, n

$$c \geq 2 + m + n$$

$$\Rightarrow S(m+n) \leq c(m+n) \Rightarrow S(m+n) = O(m+n)$$

QED