Recurrence Relation

Monday, 30 May, 2022 15:18

FROM PASS WEEK 3 NOTES

1) **Recurrence Relation**

```
def func_one(n):
    if n == 0:
        return 0
    else:
        return func_one(n-1)
```

telescoping:

$T(n) = T(n-1) + c$ – recurrence relation

$T(n) = T((n-1)-1) + c + c = T(n-2) + 2c$

$T(n) = T((n-1)-2) + 2c + c = T(n-3) + 3c$

$T(n) = T(n-k) + kc$

base case:

$T(0) = a$

big-O:

$T(0) = T(n-k) + \cancel{kc}$

$0 = n-k$

$k = n$

$T(n) = T(n-n) + nc$

$= T(0) + nc$

$= a + nc$

$= O(n)$

2)

```
def func_two(n):
    if n == 1:
        return 0
    else:
        return func_two(n//2) + 10
```

telescoping:

$T(n) = T(n/2) + c$ – recurrence relation

$T(n) = T((\frac{n}{2})/2) + c + c = T(\frac{n}{4}) + 2c$

$T(n) = T((\frac{n}{4})/2) + 2c + c = T(\frac{n}{8}) + 3c$

$T(n) = T(\frac{n}{2^k}) + kc$

base case:

$T(1) = a$

big-O:

$T(1) = T(\frac{n}{2^k}) + \cancel{kc}$

$1 = \frac{n}{2^k}$

$n = 2^k$

$k = \log_2 n$

$T(n) = T(\frac{n}{2^{\log_2 n}}) + (\log_2 n)c$

$= T(1) + \log_2 n \, c$

$= a + (\log_2 n)c$

$= O(\log n)$

3)

```
def fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

$T(0) = a$

$T(1) = b$

telescoping:

$T(n) = T(n-1) + T(n-2) + c$ ✗

$T(n) < 2T(n-1) + c$ ✓ – recurrence relation

$T(n) < 2[2T(n-2) + c] + c = 4T(n-2) + 3c$

$T(n) < 2[4T(n-3) + 3c] + c = 8T(n-3) + 7c$

$T(n) < 2^k T(n-k) + (2^k - 1)c$

base case:

$T(0) = a$, $T(1) = b$

big-O:

$T(0) = 2^k T(n-k) + \cancel{(2^k-1)}c$

$0 = n-k$

$k = n$

$T(n) = 2^n a + (2^n - 1)c$

$= O(2^n)$

$T(n) = 2^k T(n-k) + 2^{k-1}c + 2^{k-2}c + \ldots + 2^{k-k}c$

$= 2^k T(n-k) + c\underbrace{(2^0 + 2^1 + 2^2 + \ldots 2^{k-1})}$

$*n = k-1$, $r = 2$ from equation $\frac{r^{n+1} - 1}{r-1} = \frac{2^{k-1+1} - 1}{2-1} = 2^k - 1$

4)

```
def func_three(n, m):
    if n == 1:
        return m
    else:
        return 3 * func_three(n//3, m)
```

$T(1) = a$

*m is not changing

telescoping.

$T(n) = T(\frac{n}{3}) + c$ – recurrence relation

$T(n) = T(\frac{n}{9}) + 2c$

$T(n) = T(\frac{n}{27}) + 3c$

$T(n) = T(\frac{n}{3^k}) + kc$

base case:

$T(1) = a$

big-O:

$T(1) = T(\frac{n}{3^k}) + \cancel{kc}$

$1 = \frac{n}{3^k}$

$n = 3^k$

$k = \log_3 n$

$T(n) = T(\frac{n}{3^{\log_3 n}}) + (\log_3 n)c$

$T(n) = a + (\log_3 n)c$

$= O(\log_3 n)$

FROM SANITY CHECK WEEK 2

The following algorithm sums up the elements of an *array* in an unusual way. Write down the recurrence relation for this function.

2 Marks

1) $T(n) = T(n-1) + c \quad ; n > 0$
$T(n) = a \quad ; n = 0$

2) $T(n) = T(n/2) + c \quad ; n > 0$
$T(n) = a \quad ; n = 1$

3) $T(n) < 2T(n-1) + c \quad ; n > 0$
$T(n) = a \quad ; n = 0$
$T(n) = b \quad ; n = 1$

4) $T(n) = T(\frac{n}{3}) + c \quad ; n > 0$
$T(n) = a \quad ; n = 1$

What is the big-O complexity of a function whose runtime is given by the following recurrence?

Please give your answer as "O(?)" where ? is the function. Do not use any symbol for multiplication, just write the terms next to each other with no spaces, e.g. "xy" for x times y.

2 Marks

$O(n)$ or $O(\log n)$ or $O(2^n)$

FROM STUDIO02 Q3

Problem 3. Find a closed form for the following recurrence relation:

$$T(n) = \begin{cases} 2T(n-1) + a, & \text{if } n > 0, \\ b, & \text{if } n = 0. \end{cases}$$

telescoping.

$T(n) = 2T(n-1) + a$

$T(n) = 2[2T(n-2) + a] + a = 4T(n-2) + 3a$

$= 2[4T(n-3) + 3a] + a = 8T(n-3) + 7a$

$T(n) = 2^k T(n-k) + (2^k - 1)a$

$T(n) = 2^k T(n-k) + 2^{k-1}a + 2^{k-2}a + \ldots + 2^{k-k}a$

$= 2^k T(n-k) + a\underbrace{(2^0 + 2^1 + 2^2 + \ldots 2^{k-1})}$

$*n = k-1$, $r = 2$ from equation $\frac{r^{n+1} - 1}{r-1} = \frac{2^{k-1+1} - 1}{2-1} = 2^k - 1$

base case:

$T(0) = b$

big-O:

$T(n) = 2^n T(n-n) + (2^n - 1)a$

$T(0) = 2^n b + (2^n - 1)a$

$= O(2^n)$

Recurrence relation:

T(1) = b

T(N) = T(N-1) + c*N

telescoping:

$T(n) = T(n-1) + nc$

$T(n) = T(n-2) + 2nc$

$T(n) = T(n-3) + 3nc$

$= T(n-k) + knc$

base case:

$T(1) = b$

big-O:

$T(1) = T(n-k) + \cancel{knc}$

$1 = n-k$

$k = n-1$

$T(n) = T(n-n+1) + (n-1)nc$

$= b + n^2c - nc$

$= O(n^2)$

$T(N) = T(N/2) + Nc \quad ; N>1$

$T(N) = a \quad ; n=1$

telescoping:

$T(n) = T(\frac{n}{2}) + nc$

$= T(\frac{n}{4}) + 2nc$

$= T(\frac{n}{8}) + 3nc$

$= T(\frac{n}{2^k}) + knc$

base case:

$T(1) = a$

big-O:

$T(1) = T(\frac{n}{2^k}) + \cancel{knc}$

$1 = \frac{n}{2^k}$

$2^k = n$

$k = \log_2 n$

$T(n) = T(1) + (\log_2 n) nc$

$= a + (\log_2 n) nc$

$= O(n \log_2 n)$

$T(N) = 2T(\frac{N}{2}) + N^2c \qquad n>1$

$T(N) = a \qquad n=1$

Telescoping:

$T(n) = 2T(\frac{n}{2}) + n^2c$

$= 2\left[2T(\frac{n}{2^2}) + (\frac{n}{2})^2 c\right] + n^2c$

$= 2^2 T(\frac{n}{2^2}) + \frac{n^2}{2}c + n^2c$

$= 2^2\left[2T(\frac{n}{2^3}) + (\frac{n}{2^2})^2 c\right] + \frac{n^2}{2}c + n^2c$

$= 2^3 T(\frac{n}{2^3}) + \frac{n^2}{2^2}c + \frac{n^2}{2^1}c + \frac{n^2}{2^0}c$

$\therefore T(n) = 2^k T(\frac{n}{2^k}) + \frac{n^2c}{2^{k-1}} + \frac{n^2c}{2^{k-2}} + \dots + \frac{n^2c}{2^{k-k}}$

$T(n) = 2^k T(\frac{n}{2^k}) + n^2c\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \dots + \frac{1}{2^{k-1}}\right) \longrightarrow \frac{1(1-(\frac{1}{2})^k)}{1-\frac{1}{2}} = 2(1-(\frac{1}{2})^k)$

Formula: $\frac{a(1-r^n)}{1-r}$

$= 2^k T(\frac{n}{2^k}) + 2n^2c(1-(\frac{1}{2})^k)$

base case:

$T(1) = a$

big-O:

$T(1) = \cancel{2^k} T(\frac{n}{2^k}) + \cancel{2n^2c(1-(\frac{1}{2})^k)}$

$1 = \frac{n}{2^k}$

$k = \log_2 n$

$T(n) = na + 2n^2c(1-(\frac{1}{2})^{\log_2 n})$

$= na + 2n^2c - 2n^2c(\frac{1}{n})$

$= na + 2n^2c - 2nc$

$= O(n^2)$

Problem 12. Find a function T that is a solution of the following recurrence relation

$$T(n) = \begin{cases} 3T\left(\frac{n}{2}\right) + n^2 & \text{if } n > 1, \\ 1 & \text{if } n = 1. \end{cases}$$

Write an asymptotic upper bound on the solution in big-O notation.

Solution

To find a closed form for T, we will use telescoping. For $n > 1$ we have

$$T(n) = 3T\left(\frac{n}{2}\right) + n^2,$$
$$T(n) = 3\left[3T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right] + n^2 = 3^2T\left(\frac{n}{4}\right) + \frac{3n^2}{2^2} + n^2,$$
$$T(n) = 3^2\left[3T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2\right] + \frac{3n}{2^2} + n^2 = 3^3T\left(\frac{n}{8}\right) + \frac{3^2n^2}{4^2} + \frac{3n^2}{2^2} + n^2.$$

Continuing the pattern, we see that the general form for $T(n)$ seems to be

$$\begin{aligned} T(n) &= 3^kT\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k} \frac{3^i n^2}{(2^i)^2}, \\ &= 3^kT\left(\frac{n}{2^k}\right) + n^2\sum_{i=0}^{k}\left(\frac{3}{4}\right)^i. \end{aligned}$$

Using the formula for the sum of the first t terms of a geometric series, we obtain

$$\begin{aligned} T(n) &= 3^kT\left(\frac{n}{2^k}\right) + n^2\left(\frac{\left(\frac{3}{4}\right)^{k+1} - 1}{\frac{-1}{4}}\right), \\ &= 3^kT\left(\frac{n}{2^k}\right) + 4n^2\left(1 - \left(\frac{3}{4}\right)^{k+1}\right), \\ &= 3^kT\left(\frac{n}{2^k}\right) + 4n^2 - 4n^2\left(\frac{3}{4}\right)^{k+1}. \end{aligned}$$

We want a closed form solution, so we need to eliminate the term $T\left(\frac{n}{2^k}\right)$. To do so, we make use of the fact that we have $T(1) = 1$. By setting $k = \log_2(n)$, $T\left(\frac{n}{2^k}\right)$ becomes $T(1)$ and hence we obtain

$$\begin{aligned} T(n) &= 3^{\log_2(n)}T\left(\frac{n}{2^{\log_2(n)}}\right) + 4n^2 - 4n^2\left(\frac{3}{4}\right)^{\log_2(n)+1} \\ &= 3^{\log_2(n)} + 4n^2 - 4n^2\left(\frac{3}{4}\right)^{\log_2(n)+1} \end{aligned}$$

Using log laws from the Week 1 studio sheet, we can write

$$\begin{aligned} T(n) &= n^{\log_2(3)} + 4n^2 - 4n^2n^{\log_2(\frac{3}{4})}\left(\frac{3}{4}\right) \\ &= n^{\log_2(3)} + 4n^2 - 3n^2n^{\log_2(\frac{3}{4})}. \\ &= n^{\log_2(3)} + n^2(4 - 3n^{\log_2(\frac{3}{4})}). \end{aligned}$$

To three decimal places, $\log_2(3) = 1.585$ and $\log_2\left(\frac{3}{4}\right) = -0.415$. Since $4 - 3n^{-0.415}$ is between 1 to 4 for $n \geq 1$, the asymptotic behaviour is

$$\begin{aligned} T(n) &= O(n^{1.585}) + O(n^2), \\ &= O(n^2). \end{aligned}$$

Space and Aux Complexity

Monday, 30 May, 2022 16:32

(Auxiliary) Space Complexity

```
def aux_one(n):
    arr = [None] * n
    for i in range(n):
        arr[i] = i * 2
    return sum(arr)
```

arr makes a list of n which has aux of O(n).
Input space is O(1)
Therefore space complexity is O(n).

```
def aux_two(arr):
    for i in range(len(arr)):
        arr[i] += 1
    return arr
```

Aux is O(1). input space is O(n) as it is inputting an arr.
Overall space complexity is O(n).

```
def aux_three(n):
    bucket = [0] * 256
    for i in range(len(arr)):
        bucket[ord(arr[i])] += 1
    return bucket
```

Input is O(1) space.
Aux is O(1) space as it is constant.
Overall space complexity: O(1)

```
def aux_four(n):
    matrix = [None] * n
    for i in range(n):
        matrix[i] = [None] * n
    return 1000
```

Aux of O(n^2) since the for loop creates another list size n.
Input of O(1)
Overall space complexity: O(n^2)

Divide and Conquer Past Year Q

Wednesday, 1 June, 2022 01:54

Problem 9. Recommender systems are widely employed nowadays to suggest new books, movies, restaurants, etc that a user is likely to enjoy based on his past ratings. One commonly used technique is collaborative filtering, in which the recommender system tries to match your preferences with those of other users, and suggests items that got high ratings from users with similar tastes. A distance measure that can be used to analyse how similar the rankings of different users are is counting the number of inversions. The counting inversions problem is the following:

- **Input:** An array V of n distinct integers.
- **Output:** The number of inversions of V, i.e., the number of pairs of indices (i, j) such that $i < j$ and $V[i] > V[j]$.

The exhaustive search algorithm for solving this problem has time complexity $O(n^2)$. Describe an algorithm with time complexity $O(n \log n)$ for solving this problem.

[Hint: Adapt a divide-and-conquer algorithm that you already studied.]

index = 1 2 3 4 5

| 3 | 5 | 1 | 2 | 4 |

$i < j$: $1 < 2$

$V[i] < V[j]$ = $3 < 5$

not an inversion

$i < j$

$V[i] > V[j]$

inversion are

(1, 3)

(1, 4)

$O(n^2)$ approach

for i in range 1...n

for j in range i+1...n

if $V[i] > V[j]$

inversion += 1

1. Use a merge sort to sort the array in decreasing order
2. Using a loop for each "i" check how many subsequent items are duplicates("x") and skip them
3. Once x are skipped, add n-(i+x) to the numbers of inversions
4. Repeat steps 2 to 3, but make the next i = i+x

Text Book page 20,21

2.3 Counting Inversions

Recommender systems are widely employed nowadays to suggest new books, movies, restaurants, etc that a user is likely to enjoy based on his past ratings. One commonly used technique is collaborative filtering, in which the recommender system tries to match your preferences with those of other users, and suggests items that got high ratings from users with similar tastes. A distance measure that can be used to analyse how similar the rankings of different users are is counting the number of inversions. The counting inversions problem is the following:

- **Input:** An array A of n distinct integers.
- **Output:** The number of inversions of A, i.e., the number of pairs of indices (i, j) such that $i < j$ and $A[i] > A[j]$.

The exhaustive search algorithm for solving this problem has time complexity $O(n^2)$, but we want to improve on that and obtain an algorithm with time complexity $O(n \log n)$ for solving this problem. Note that potentially there are $\Theta(n^2)$ inversions, so an algorithm for solving the problem with time complexity $O(n \log n)$ cannot look individually at each possible inversion.

The basic idea to obtain a $O(n \log n)$ solution is to adapt the Merge Sort algorithm to solve this problem. We split the array in the middle and invoke recursive calls on the first half and the second half. Each recursive call will count the number of inversions in that subarray *and also sort the elements of that subarray*. Getting the elements of the subarrays sorted is key to allowing us to count, during the merging procedure, in time $O(n)$ the number of "split inversions", i.e., inversions in which i belongs to the left subarray and j to the right subarray.

When we are performing the merging procedure, at each step the smallest remaining element is selected (and it will be either the first remaining element of the left subarray or of the right subarray, as the subarrays are sorted). If that smallest element comes from the left subarray, then there are no split inversions to be counted (as the index of this element is smaller than the indices of all elements in the right subarray). On the other hand, if that smallest element comes from the right subarray, then the number of split inversions should be increased by the amount of elements still to be processed in the left subarray (as all those elements have smaller indices than the selected one).

The pseudocode of the algorithm is presented below:

Algorithm 7 Sort-and-CountInv

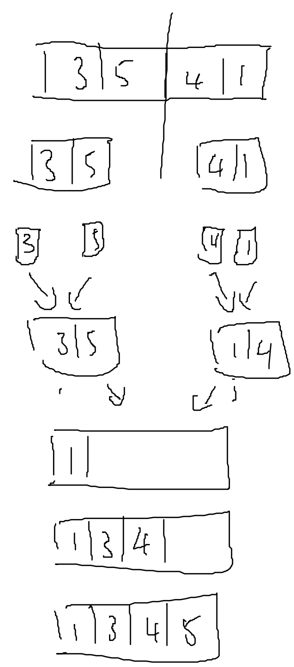
```
function SORT-AND-COUNTINV(array[lo..hi])
    if lo = hi then
        return (array[lo], 0)
    else
        mid = ⌊(lo + hi)/2⌋
        (array[lo..mid], Inv_L) = SORT-AND-COUNTINV(array[lo..mid])
        (array[mid+1..hi], Inv_H) = SORT-AND-COUNTINV(array[mid+1..hi])
        (array[lo..hi], Inv_S) = MERGE-AND-COUNTSPLITINV(array[lo..mid], array[mid+1..hi])
        Inv = Inv_L + Inv_H + Inv_S
        return (array[lo..hi], Inv)
```

What is the invariant of a merge sort? Why does merge sort works?

We are comparing the smallest from the left and from the right when merging since both of the left and the right is already sorted.

The item of the left came from the front of the list so when we compare the left side which is the front of the list it can compare with the right of the list and if i < j we can +1 to the inversion count.

inversions



+1

+2

+1

total 4 inversions.

Algorithm 8 Merge-and-CountSplitInv

```
function MERGE-AND-COUNTSPLITINV(A[i..n1], B[j..n2])
    result = empty array
    splitInversions = 0
    while i ≤ n1 or j ≤ n2 do
        if j > n2 or (i ≤ n1 and A[i] ≤ B[j]) then
            result.append(A[i])
            i += 1
        else
            result.append(B[j])
```

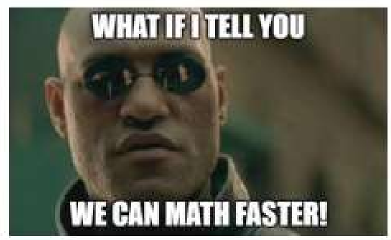
```
while i ≤ n1 or j ≤ n2 do
    if j > n2 or (i ≤ n1 and A[i] ≤ B[j]) then
        result.append(A[i])
        i += 1
    else
        result.append(B[j])
        j += 1
        splitInversions = splitInversions + n1 - i + 1
return (result, splitInversions)
```

Karatsuba

Thursday, 2 June, 2022 23:42

Karatsuba
Simple Quick Integer Multiplication

- By breaking large numbers into smaller ones
  - x = 1234 = 12 * 10^2 + 34 * 10^0
  - y = 6789 = 67 * 10^2 + 89 * 10^0
  - x * y = (12 * 10^2 + 34 * 10^0) * (67 * 10^2 + 89 * 10^0)
    = (12 * 10^2 * 67 * 10^2) + (12 * 10^2 * 89 * 10^0)
    + (34 * 10^0 * 67 * 10^2) + (34 * 10^0 * 89 * 10^0)
    = (12 * 67 * 10^4) + (12 * 89 * 10^2)
    + (34 * 67 * 10^2) + (34 * 89)
    = (12 * 67 * 10^4) + (12 * 89 + 34 * 67) * 10^2 + (34 * 89)
- And we can do even better!
  - Add small numbers, then only multiply!
  - What if I tell you we can do even better?



Karatsuba
Simple Quick Integer Multiplication

- Recall we stopped at the following
  - Therefore $x * y = x_l * y_l * 10^n + (x_l * y_r + x_r * y_l) * 10^{n/2} + x_r * y_r$
  - Gauss introduce a trick for us
    - $(x_l + x_r) * (y_l + y_r) = x_l * y_l + x_l * y_r + x_r * y_l + x_r * y_r$
    - Then we rearrange the above…
    - $x_l * y_r + x_r * y_l = (x_l + x_r) * (y_l + y_r) - x_l * y_l - x_r * y_r$

- Then we rearrange the above...
  - $x_l * y_r + x_r * y_l = \underbrace{(x_l + x_r) * (y_l + y_r)} - x_l * y_l - x_r * y_r$
  - Why are we doing this?
    - 1 multiplication instead of 2 multiplication
    - Note that it is slower to multiply than it is to add/ subtract in general

Karatsuba
In summary

- Given 2 large numbers
- Divide and conquer the large number into 2 halves
  - Smaller numbers are faster to operate on
  - Only need 3 multiplications, on smaller numbers

    We can follow Karatsuba again
    for the 3 multiplications!

- Then combine the result

Studio02 Q4

Saturday, 5 March, 2022 16:17

Problem 3. Using mathematical induction, prove the following identity:

$$\sum_{i=0}^{n} r^i := 1 + r + r^2 + r^3 + \ldots + r^n = \frac{r^{n+1}-1}{r-1}, \qquad \text{for all } n \geq 0, r \neq 1.$$

Problem 4. Using Problem 3, show that the following inequality is true for all integers $n \geq 1$

$$\sum_{i=0}^{n} \frac{1}{2^i} := 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^n} < 2.$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + n^2 c\left(\frac{1}{2^0} + \frac{1}{2^1} + \frac{1}{2^2} + \ldots + \frac{1}{2^{k-1}}\right) \longrightarrow \frac{1\left(1-\left(\frac{1}{2}\right)^k\right)}{1-\frac{1}{2}} = 2\left(1-\left(\frac{1}{2}\right)^k\right)$$

$$= 2^k T\left(\frac{n}{2^k}\right) + 2n^2 c\left(1-\left(\frac{1}{2}\right)^k\right)$$

Formula:

$$\frac{a(1-r^n)}{1-r}$$

Notice that $\sum_{i=0}^{n} \frac{1}{2^i}$ can be written as $\sum_{i=0}^{n} \left(\frac{1}{2}\right)^i$

so $r = \frac{1}{2}$ we get

$$\sum_{i=0}^{n} \left(\frac{1}{2}\right)^i = \frac{\left(\frac{1}{2}\right)^{n+1} - 1}{\frac{1}{2} - 1}$$

$$= \frac{\frac{1}{2}\left(\left(\frac{1}{2}\right)^n - 2\right)}{\frac{1}{2}(1-2)}$$

$$= \frac{\left(\frac{1}{2}\right)^n - 2}{-1}$$

$$= 2 - \left(\frac{1}{2}\right)^n$$

we have that

$$\sum_{i=0}^{n} \left(\frac{1}{2}\right)^i = 2 - \left(\frac{1}{2}\right)^n$$

Since $\left(\frac{1}{2}\right)^n > 0$ for all $n \geq 1$, we have $2 - \left(\frac{1}{2}\right)^n < 2$, and hence it is true that $\sum_{i=0}^{n} \frac{1}{2^i} < 2$ as required