Kmad 5568@uni.sydney.edu

Warm-up

Problem 1. Sort the following functions in increasing order of asymptotic growth

$$n, n^3, n\log n, n^n, \frac{3^n}{n^2}, n!, \sqrt{n}, 2^n$$

remember this

$n^n > n! > \frac{3^n}{n^2} > 2^n > n^3 > n\log n > n > \sqrt{n}$

Problem 2. Sort the following functions in increasing order of asymptotic growth

$$\log\log n, \log n!, 2^{\log\log n}, n^{\frac{1}{\log n}}$$

$\log n = x$

$\log n! > 2^{\log\log n} > \log\log x > n^{\frac{1}{\log n}}$

Problem 3. Consider the following pseudocode fragment.

```
1: function PRINTFIVE(n)
2:     for i ← 1; i ≤ 5; i++ do
3:         Print a single character 'n'
```

$\sum_{i=1}^{5} O(1) = 5 \cdot O(1) = O(1)$ since the loop time is constant

Using the O-notation, upperbound the running time of PRINTFIVE.

Problem 4. Consider the following pseudocode fragment.

```
1: function STARS(n)
2:     for i ← 1; i ≤ n; i++ do
3:         Print '*' i times
```

$O(i)$ } $O(n)$

a) Using the O-notation, upperbound the running time of STARS.

b) Using the Ω-notation, lowerbound the running time of STARS to show that your upperbound is in fact asymptotically tight.

Problem 5. Recall the problem we covered in lecture: Given an array A with n entries, find $0 \le i \le j < n$ maximizing $A[i] + \cdots + A[j]$.

Prove that the following algorithm is incorrect: Compute the array B as described in the lectures. Find i minimizing $B[i]$, find j maximizing $B[j+1]$, return (i, j).

Come up with the smallest example possible where the proposed algorithm fails.

Problem solving

Problem 6. Given an array A consisting of n integers, we want to compute the upper triangle matrix C where

$$C[i][j] = \frac{A[i] + A[i+1] + \cdots + A[j]}{j - i + 1}$$

for $0 \le i \le j < n$. Consider the following algorithm for computing C:

```
1: function SUMMING_UP(A)
2:     C ← new matrix of size(A) by size(A)
3:     for i ← 0; i < n; i++ do
4:         for j ← i; j < n; j++ do
5:             Compute average of entries A[i : j]
6:             Store result in C[i, j]
7:     return C
```

a) Using the O-notation, upperbound the running time of SUMMING-UP.

b) Using the Ω-notation, lowerbound the running time of SUMMING-UP.

Problem 7. Come up with a more efficient algorithm for computing the above matrix $C[i][j] = \frac{A[i]+A[i+1]+\cdots+A[j]}{j-i+1}$ for $0 \leq i \leq j < n$. Your algorithm should run in $O(n^2)$ time.

Problem 8. Give a formal proof of the transitivity of the O-notation. That is, for function f, g, and h show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

Problem 9. Using O-notation, show that the follow snippet of code runs in $O(n^2)$ time. As an extra challenge, it can be shown that it actually runs in $O(n)$ time.

```
1: function ALGORITHM(n)
2:     j ← 0
3:     for i ← 0; i < n; i++ do
4:         while j ≥ 1 and [condition that can be checked in O(1) time] do
5:             j ← j − 1
6:         j ← j + 1
7:     return j
```

Problem 10. Given a string (which you can see as an array of characters) of length n, determine whether the string contains k identical consecutive characters. You can assume that $2 \leq k \leq n$. For example, when we take $k = 3$, the string "abaaab" contains three consecutive a's and thus we should return true, while the string "abaaba" doesn't contain three consecutive characters that are the same.

Your task is to design an algorithm that always correctly returns whether the input string contains k identical consecutive characters. Your solution should run in $O(n)$ time. In particular, k isn't a constant and your running time shouldn't depend on it.

Problem 11. Given an array with n integer values, we would like to know if there are any duplicates in the array. Design an algorithm for this task and analyze its time complexity.

Problem 4. Consider the following pseudocode fragment.

```
1: function STARS(n)
2:     for i ← 1; i ≤ n; i++ do
3:         Print '*' i times
```

O(i) } O(n)

a) Using the O-notation, upperbound the running time of STARS.

b) Using the Ω-notation, lowerbound the running time of STARS to show that your upperbound is in fact asymptotically tight.

(a) $\sum_{j=1}^{n} j \leq \sum_{j=1}^{n} n = n^2 = O(n^2)$

(b) $\sum_{j=1}^{n} j \geq \sum_{j=\frac{n}{2}+1}^{n} \frac{n}{2} = \frac{n^2}{4} = \Omega(n^2)$

Problem 5. Recall the problem we covered in lecture: Given an array A with n entries, find $0 \le i \le j < n$ maximizing $A[i] + \cdots + A[j]$.

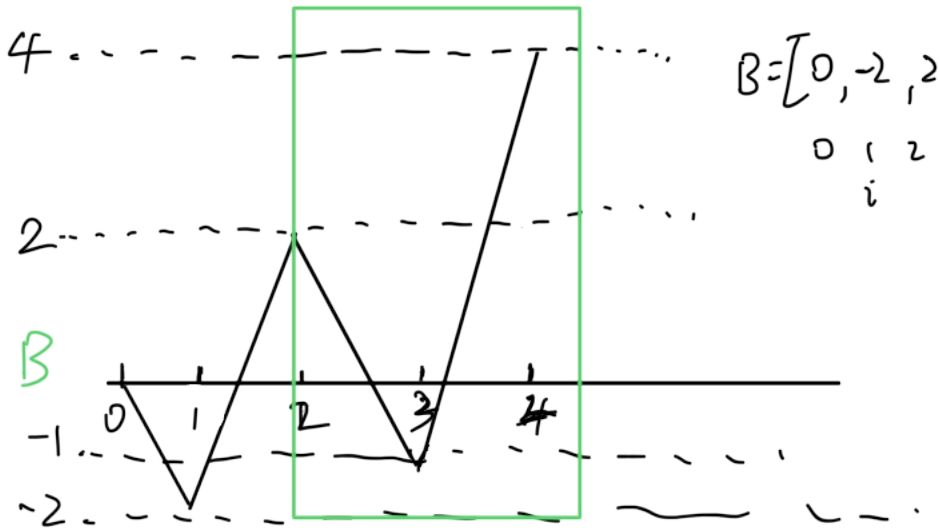
Prove that the following algorithm is incorrect: Compute the array B as described in the lectures. Find i minimizing $B[i]$, find j maximizing $B[j+1]$, return (i, j).

Come up with the smallest example possible where the proposed algorithm fails.

assume $k = j+1$

Remember $B[i] = B[i-1] + A[i-1]$. except for $i=0$, since $B[0] = 0$

$[-2, 4, -3, 5] = A$ ✗



$B = [0, -2, 2, -1, 4]$

0 1 2 3 4

i j k

✓

类似于 $[1, -2, 3]$，即在最低点后立马碰最大值

不能成立，因为只包含导致最大值的数会们到最大的 $A[i] \cdots A[j]$

目的是为了让 i 和 j 都处于最低点

✓ $[0, 1, -1]$ 因为 $i \le j$

$A = [A_1, \cdots A_i, \cdots, A_j, \cdots A_n]$

Problem 6. Given an array A consisting of n integers, we want to compute the upper triangle matrix C where

$$C[i][j] = \frac{A[i] + A[i+1] + \cdots + A[j]}{j - i + 1}$$

for $0 \le i \le j < n$. Consider the following algorithm for computing C:

i: row number
j = col number
i≤j means row number less than col number


```
1: function SUMMING_UP(A)
2:     C ← new matrix of size(A) by size(A)
3:     for i ← 0; i < n; i++ do
4:         for j ← i; j < n; j++ do
5:             Compute average of entries A[i : j]
6:             Store result in C[i, j]
7:     return C
```

i从第0行到第n行

j从第i列到第n列

因为j要从i取到n(根据题目)

$O(j-i)$ $\Omega(1)$ $\Theta(n)$ $\Theta(j-i+1)$

↳ since need to sum from A[i] to A[j]

P6 (a)

$$\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j-i+1)$$

$$= \sum_{i=0}^{n-1} \sum_{j=i}^{(n-1)-i+i} (k+1) \quad \# \ k = j-i$$

$$= \sum_{i=0}^{n-1} \sum_{k=0}^{n-i-1} k+1$$

$j - i = k$

For each i, the inner sum is

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

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

$$= \Theta(n^2)$$

Include outer sum notation, we have

$$\sum_{i=0}^{n-1} \Theta(n^2) = \Theta(n^3)$$

(b)
- we take $i < \frac{1}{4}n$, $j > \frac{3}{4}n$. There are $\frac{3n^2}{4 \cdot 4}$ possibilities, which is $\Omega\left(\frac{3n^2}{16}\right) = \Omega(n^2)$ (for outer 2 "for loops")
- line 5 will execute $\Omega(n/2)$ times, since calculate $[\frac{1}{4}n, \frac{3}{4}n]$ would traverse $\frac{1}{2}n$
- combine we have $\Omega(n^3)$

P7. I guess we need to simplify the "avg part"?

Put it out of for loop,

$\sum_{k=0}^{n} A_k$ restore all A_k in an Array X

Then $\sum_{k=i}^{j} A_k$ Equivalent to $X_j - X_i$

which is $O(1)$

for . . .

for . . .

Then $\sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \Theta(1) = \Theta(n^2)$

Problem 8. Give a formal proof of the transitivity of the O-notation. That is, for function f, g, and h show that if $f(n) = O(g(n))$ and $g(n) = O(h(n))$ then $f(n) = O(h(n))$.

$f = O(g)$ means since there exists $n_0 > 0$ and $c > 0$ such that $f(n) \leq c g(n)$ for all $n \geq n_0$

$g = O(h)$ means since there exists $n_0' > 0$ and $c' > 0$ such that $g(n) \leq c' h(n)$

so $f(n) \leq c \cdot g(n) \leq c c' h(n)$

clearly $f = O(h)$

Problem 9. Using O-notation, show that the follow snippet of code runs in $O(n^2)$ time. As an extra challenge, it can be shown that it actually runs in $O(n)$ time.

```
1: function ALGORITHM(n)
2:     j ← 0
3:     for i ← 0; i < n; i++ do
4:         while j ≥ 1 and [condition that can be checked in O(1) time] do
5:             j ← j − 1
6:         j ← j + 1
7:     return j
```

only execute when i is odd number

- $\sum_{i=0}^{n} O(\frac{n}{2}) = O(\frac{n^2}{2}) \leadsto O(n^2)$

- it can be runs in O(n) since we can use "if n%2=0 and [...]" instead of while loop

Problem 10. Given a string (which you can see as an array of characters) of length n, determine whether the string contains k identical consecutive characters. You can assume that $2 \leq k \leq n$. For example, when we take $k = 3$, the string "abaaab" contains three consecutive a's and thus we should return true, while the string "abaaba" doesn't contain three consecutive characters that are the same.

Your task is to design an algorithm that always correctly returns whether the input string contains k identical consecutive characters. Your solution should run in $O(n)$ time. In particular, k isn't a constant and your running time shouldn't depend on it.

```
last = ?
count = 0
for i <- 1, i<n, i++
    if S[i] == last:
        count = count + 1
        if count == k
            return True
    else
        last = S[i]
        count = 0
return False
```

Problem 11. Given an array with n integer values, we would like to know if there are any duplicates in the array. Design an algorithm for this task and analyze its time complexity.