Question 1 (2 marks)

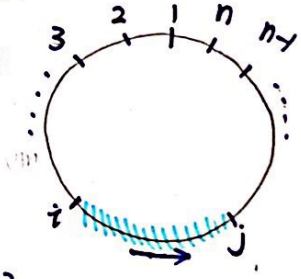
For the maximum subarray problem, we are given an array $A[1,\cdots,n]$ containing positive and negative numbers and asked to find the pair i and j with $1 \le i \le j \le n$ such that the sum $A[i]+\cdots+A[j]$ of the corresponding subarray is maximum. Modify that algorithm so that it works on circular arrays (still in $O(n\log n)$ time): i.e, it should find the pair i and j with $1 \le i, j \le n$ such that the sum of the corresponding subarray is maximum, where that sum is still $A[i]+\cdots+A[j]$ where $i \le j$, but is $A[i]+\cdots+A[n]+A[1]+\cdots+A[j]$ when $i > j$

i & j denote the indices of elements in A.

Denote sum of any subarray of A as $sumSub(A_{[i,j]})$, sum of A as $sum(A)$

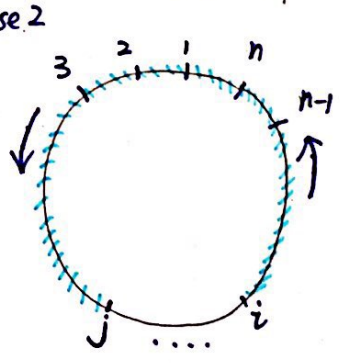
Analysis There can be two cases for the maximum sum of subarray, denoted as maxSumSub.

Case 1: The elements that contribute to the maximum sum are arranged such that $i \le j$ and $maxSumSub = \sum_{k=i}^{j} A[k]$



For example $\{-10, 2, -1, 5\} \to (2, -1, 5)$ and $2-1+5=6$

In this case, the **FIND-MAXIMUM-SUBARRAY (FMS)** algorithm will produce the result



Case 2: The elements that contribute to the maximum sum are arranged such that $i > j$ and $max\ sum = \sum_{k=i}^{n} A[k] + \sum_{k=1}^{j} A[k]$

For example $\{10, -12, 11\} \to (11, 10)$ and $11+10=21$

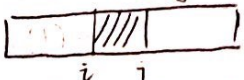
In this case, we consider the additive inverse of A. Then we find the maximum sum of any subarray of $-A$ using algorithm FMS:

$A = [A_1, A_2, \cdots, A_n]$ and $-A = [-A_1, -A_2, \cdots, -A_n] = A' = [A_1', A_2', \cdots, A_n']$

Suppose $maxSumSub(A') = \sum_{k=i}^{j} A'[k]$, this means $maxSumSub(A') = \sum_{k=i}^{j} A'[k] = A'[i] + A'[i+1] + \cdots + A'[j]$

$= (-A_i) + (-A_{i+1}) + \cdots + (-A_j) = -(A_i + A_{i+1} + \cdots + A_j)$. By definition $\sum_{k=i}^{j} A'[k] \ge sumSub(A'_{[m,n]})$

$A'_{[m,n]}$ is any subarray in A'. It follows that $-(A_i + \cdots + A_j) \ge -sumSub(A)$ Multiplying the inequality on both sides by (-1), we have $(A_i + \cdots + A_j) \le sumSub(A)$. $i \le j$.

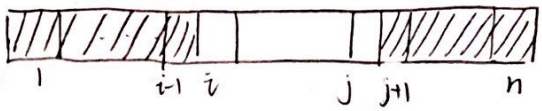


This means $\sum_{k=i}^{j} A_k$ is the minimum sum of any subarray of circular A !

Since $sum(A) = \sum_{k=1}^{n} A[k] = \sum_{k=1}^{i-1} A[k] + \sum_{k=i}^{j} A[k] + \sum_{k=j+1}^{n} A[k]$, $\sum_{k=1}^{i-1} A[k] + \sum_{k=j+1}^{n} A[k] = sum(A) - \sum_{k=i}^{j} A[k]$

When $\sum_{k=i}^{j} A[k]$ achieves its minimum value, $\sum_{k=1}^{i-1} A[k] + \sum_{k=j+1}^{n} A[k]$ achieves its maximum value

This yields the maximum sum of subarray of circular A. as shown by the shaded area on the right. Hence:



$maxSumSub(A) = sum(A) + maxSumSum(-A)$

Note that in this case it is defined that $i > j$, so we need to do an assignment

$temp \leftarrow j$

$j \leftarrow i$

$i \leftarrow temp$

Finally, we need to compare the maxSumSub returned from the two cases, take the larger one, and return the corresponding results.

The first and second algorithms are cited from the text book "Introduction to Algorithms" by Cormen et al. Note in algorithms below, index starts from 1

Algorithm 1 : FIND-MAX-CROSSING-SUBARRAY : SMCS (A, low, mid, high)

```
1  left-sum = -∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > lef-sum
6          left-sum = sum
7          max-left = i
8  right-sum = -∞
9  sum = 0
10 for j = mid+1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```

Suppose A contains n entries, $n = high - low + 1$

Then this algorithm takes $\theta(n)$ time

Algorithm 2: FIND-MAXIMUM-SUBARRAY: FMS(A, low, high)

This algorithm is modified trivially to fit into algorithm 3.

```
1  mid = ⌊(low + high)/2⌋                                          Θ(1)
2  (left-low, left-high, left-sum) = FMS(A, low, mid)              T(n/2)
3  (right-low, right-high, right-sum) = FMS(A, mid+1, high)        T(n/2)
4  (cross-low, cross-high, cross-sum) = FMCS(A, low, mid, high)    Θ(n)
5  if left-sum ≥ right-sum and left-sum ≥ cross-sum                }
6      return (left-low, left-high, left-sum)                      }
7  if right-sum ≥ left-sum and right-sum ≥ cross-sum               } Θ(1)
8      return (right-low, right-high, right-sum)                   }
9  else return (cross-low, cross-high, cross-sum)                  }
```

sum(A) returns the sum of all entries in A

Algorithm 3: MAX-SUM-CIRCULAR-SUBARRAY: MSCS(A, low, high)

```
1  if high == low                                                  } T(1) = Θ(1)
2      return (low, high, A[low]) // base case                     }
3  else (index1, index2, primerSum) = FMS(-A, low, high)           Θ(nlgn)
4      (indexLow, indexHigh, regularMax) = FMS(A, low, high)       Θ(nlgn)
5      circularMax = sum(A) + primerSum                            Θ(n)
6      if circularMax ≥ regularMax                                 }
7          return (index2, index1, circularMax)                    } Θ(1)
8      else return (indexLow, indexHigh, regularMax)               }
```

Analysis: We make the simplifying assumption that the original problem size is a power of 2, so that all subproblems have sizes of positive integer. We denote T(n) the running time of FMS on a subarray of n elements. $T(n) = \Theta(1) + 2T(\frac{n}{2}) + \Theta(n) + \Theta(1) = 2T(\frac{n}{2}) + \Theta(n)$ (as shown by green markers above) $a=2$ $b=2$ then $\log_b a = 1$ and $f(n) = \Theta(n)$. Then we can apply case 2 in Master Theorem and thus $T(n) = \Theta(n\lg n)$. Substituting $T(n)$ by $\Theta(n\lg n)$ to the analysis of algorithm 3 (as shown by orange marker above), the time complexity of algorithm Mscs(A, low, high) is $\Theta(1) + \Theta(n\lg n) + \Theta(n\lg n) + \Theta(n) + \Theta(1) = \Theta(n\lg n)$.

Question 2 (2 marks) Fill in the table from Problem 3-2 in the text:

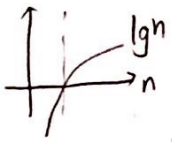
Relative Asymptotic Growth

Indicate, for each pair of expressions (A,B) in the table below, whether A is O, o, Ω, ω, or Θ of B. Assume that $k \geq 1$, $\varepsilon > 0$, and $c > 1$ are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

	A	B	O	o	Ω	ω	Θ
a.	$\lg^k n$	n^ε	yes	yes	no	no	no
b.	n^k	c^n	yes	yes	no	no	no
c.	$\sqrt{n}$	$n^{\sin n}$	no	no	no	no	no
d.	2^n	$2^{n/2}$	no	no	yes	yes	no
e.	$n^{\lg c}$	$c^{\lg n}$	yes	no	yes	no	yes
f.	$\lg(n!)$	$\lg(n^n)$	yes	no	yes	no	yes

Justifications for each cell can be found on pages 4.1 and 4.2.

Justifications for table on page 4



(a) $A = \lg^k n \quad B = n^\varepsilon$

$$\lim_{n\to\infty} \frac{\lg A}{\lg B} = \lim_{n\to\infty} \frac{\lg(\lg n)^k}{\lg(n^\varepsilon)} = \lim_{n\to\infty} \frac{k\lg(\lg n)}{\varepsilon \lg n} \overset{\lg n = x,\ \lim_{n\to\infty}\lg n=\infty}{=\!=\!=} \lim_{x\to\infty} \frac{k\lg x}{\varepsilon x} = \frac{k}{\varepsilon}\lim_{x\to\infty}\frac{\lg x}{x} = 0$$

Hence $\lg A = o(\lg B) \equiv A = o(B) \Rightarrow A = O(B)$

$\Downarrow$

$A \neq \Omega(B)$

$\Downarrow$

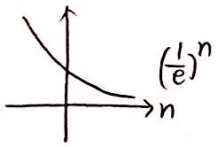
$A \neq \Theta(B),\ A \neq w(B)$

Derivative of logarithmic function: $\frac{d}{dx}\log_a x = \frac{1}{x\ln a}$

Draft: $\lim_{n\to\infty} e^{n\left(\frac{k\ln n}{n}\right)} = e^{\lim_{n\to\infty} n\frac{k\ln n}{n}} = e^{\lim_{n\to\infty} n \lim_{n\to\infty}\frac{k\ln n}{n}} = e^{\lim_{n\to\infty}\frac{nk}{n\ln 2}} = e^{\frac{k}{\ln 2}} = c$

Properties of limit: $\lim_{x\to a}(f\circ g(x)) = \left(\lim_{x\to a} f(x)\right)\left(\lim_{x\to a} g(x)\right)$

$$\lim_{x\to a} b^{f(x)} = b^{\lim_{x\to a} f(x)} \qquad \lim_{x\to a}(\log_b f(x)) = \log_b\left(\lim_{x\to a} f(x)\right)$$



(b) $A = n^k \quad B = c^n$

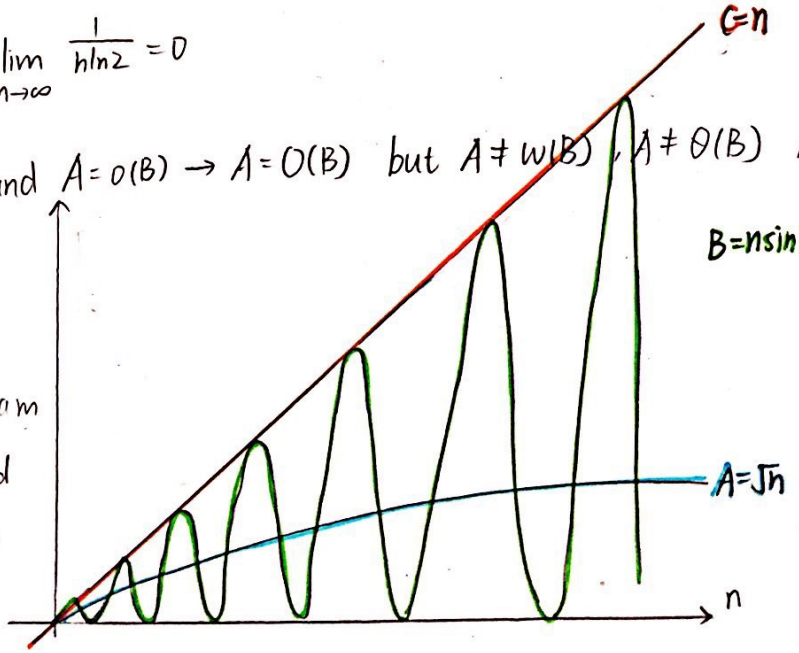
Sol: $\lim_{n\to\infty}\frac{n^k}{c^n} = \lim_{n\to\infty}\frac{e^{k\ln n}}{e^{n\ln c}} = \lim_{n\to\infty} e^{k\ln n - n\ln c} = \lim_{n\to\infty} e^{n\left(\frac{k\ln n}{n} - \ln c\right)} = \lim_{n\to\infty} e^{-\ln c\, n} = \lim_{n\to\infty}\left(\frac{1}{e}\right)^{\ln c\, n} = 0$

Note that $\lim_{n\to\infty}\frac{\ln n}{n} = \lim_{n\to\infty}\frac{1}{n\ln 2} = 0$

Hence $n^k = o(c^n)$ and $A = o(B) \to A = O(B)$ but $A \neq w(B),\ A \neq \Theta(B),\ A \neq \Omega(B)$

(c) $A = \sqrt{n},\ B = n^{\sin n}$

Sol: As shown by the diagram on the right $A \neq \Theta(B)$ and $A \neq O(B)$ and $A \neq \Omega(B) \Rightarrow A \neq o(B)$ and $A \neq w(B)$



d. $A=2^n$ $B=2^{n/2}$

Solution $\lim_{n\to\infty}\frac{A}{B}=\lim_{n\to\infty}\frac{2^n}{2^{n/2}}=\lim_{n\to\infty}2^{n/2}=\lim_{n\to\infty}\sqrt{2}^n=\infty$.

Hence $A=\omega(B)\rightarrow A=\Omega(B)$

$\Downarrow$

$A\neq\Theta(B)$

$\Downarrow$

$A\neq O(B)$

$\Downarrow$

$A\neq o(B)$

e. $A=n^{\lg c}$ $B=c^{\lg n}$

Solution: Note that $\lg A=\lg n^{\lg c}=\lg c\lg n$ Hence $\lg A=\lg B\Rightarrow A=B$

$\lg B=\lg c^{\lg n}=\lg n\lg c$

Hence $\lim_{n\to\infty}\frac{A}{B}=\lim_{n\to\infty}\frac{\lg B}{\lg A}=\lim_{n\to\infty}\frac{\lg n\lg c}{\lg c\lg n}=1$ Then $A=\Theta(B)\Rightarrow A=\Omega(B)\not\Rightarrow A\neq\omega(B)$

$\Downarrow$

$A=O(B)$

$\Downarrow$

$A\neq o(B)$

f. $A=\lg(n!)$ $B=\lg(n^n)=n\lg n$

Sol. $A=\lg(n!)=\lg n+\lg(n-1)+\cdots+\lg 2+\lg 1\leq\lg n+\lg n+\cdots+\lg n=n\lg n=B$ Hence $c=1$ and $n_0=1$

It follows that $\lg(n!)\in O(n\lg n)$ $\nearrow$ $\lg(n!)\in\Omega(\frac{n}{2}\lg\frac{n}{2})$ $\frac{n}{2}\lg\frac{n}{2}\in\Omega(n\lg n)$ (*)

$\lg(n!)=\sum_{i=1}^{n}\lg i\geq\sum_{i=n/2}^{n}\lg i\geq\sum_{i=n/2}^{n}\lg n/2=\frac{n}{2}\lg\frac{n}{2}=\frac{n}{2}(\lg n-\lg 2)=\frac{1}{2}n\lg n-\frac{1}{2}n\in\Omega(n\lg n)$

To show $\frac{1}{2}n\lg n-\frac{1}{2}n\in\Omega(n\lg n)$, we need to show there exist c and $n_0\in\mathbb{R}^+$ s.t

$\frac{1}{2}n\lg n-\frac{1}{2}n\geq cn\lg n$

$(\frac{1}{2}-c)n\lg n\geq\frac{1}{2}n\Rightarrow(\frac{1}{2}-c)\lg n\geq\frac{1}{2}\Rightarrow$ When $c=\frac{1}{3}$, $\frac{1}{2}-c=\frac{1}{6}$ $\frac{1}{6}\lg n\geq\frac{1}{2}\Rightarrow\lg n\geq 3$ $n_0=8$

Hence, $\exists\ c=\frac{1}{3}$ $n_0=8$, s.t. $\frac{1}{2}n\lg n-\frac{1}{2}n\geq cn\lg n$. Finally by (*) and transitivity, $A\in\Theta(B)$ **4.2**

Question 3 (2 marks)

Use the master method to show that Karatsuba's algorithm takes $O(n^{1.585})$ time (at least when the number of digits is a power of 2)

Solution: In class, we obtain the recurrence relation $T(n) = 3T(\frac{n}{2}) + 4n$ for Karatsuba's algorithm. Here given $T(n) = aT(n/b) + f(n)$ $a = 3$ $b = 2$ $f(n) = 4n$.

$\log_b a = \log_2 3 \simeq 1.585$ $\quad f(n) = O(n^{1.585-\varepsilon})$, where $\varepsilon = 0.2$

We can apply case 1 of the master theorem and conclude that $T(n) = \Theta(n^{1.585})$. It follows that $T(n) = O(n^{1.585})$.

Below is just a study note for Sarah.

$f(n) = 4n$ and we need to find ε such that $4n = O(n^{1.585-\varepsilon})$ (Note that we ignore constant factors in comparing asymptotic running time

① $\varepsilon > 0$ ② $1 \leq 1.585 - \varepsilon \Rightarrow \varepsilon \leq 0.585$

Hence $\varepsilon \in (0, 0.585]$

Question 4 (2 marks)

Can the master method be applied to the recurrence $T(n) = 4T(n/2) + n^2 \lg n$? Give an asymptotic upper bound for this recurrence.

Solution: Given $T(n) = aT(n/b) + f(n)$, we know $a=4$, $b=2$, $f(n) = n^2 \lg n$

$\log_b a = \log_2 4 = 2$ $f(n) = n^2 \lg n$. It is clear that case 1 and 2 do not apply. $f(n)$ is asymptotically larger than n^2, but by the **analysis** below, it is not polynomially larger than n^2, so case 3 does not apply. Hence Master theorem cannot be applied to T(n)

We now show that $T(n) = O(n^2 \lg^2 n)$. Following we prove this is true by induction on n

Suppose $T(\frac{n}{2}) = O((\frac{n}{2})^2 \lg^2(\frac{n}{2})) = O(\frac{n^2}{4}\lg^2(\frac{n}{2}))$, then $T(\frac{n}{2}) \le \frac{n^2}{4}\lg^2(\frac{n}{2})$ for sufficiently large n. Then, $T(n) = 4T(\frac{n}{2}) + n^2 \lg n \le 4\frac{n^2}{4}\lg^2(\frac{n}{2}) + n^2 \lg n = n^2(\lg n - \lg 2)^2 + n^2 \lg n = n^2(\lg n - 1)^2 + n^2 \lg n$

$= n^2(\lg^2 n - 2\lg n + 1) + n^2 \lg n = n^2 \lg^2 n - n^2 \lg n + n^2 = n^2 \lg^2 n - n^2(\lg n - 1) < n^2 \lg^2 n$. Hence $T(n) = O(n^2 \lg^2 n)$

Base Case: When $n=4$ $T(2) = 4T(1) + 2^2 \lg 2 = 4 + 4 = 8$ $n^2(\lg n)^2 = 4(\lg 2)^2 = 4$

$T(4) = 4T(2) + 4^2 \lg 4 = 4 \times 8 + 16 \times 2 = 64$ $4^2(\lg 4)^2 = 16 \times 4 = 64$

It follows by induction that $T(n) = O(n^2 \lg^2 n)$ when $n \ge 4$.

Analysis: Here we show that $f(n)$ is not polynomially larger than $n^{\log_b a} = n^2$

Proof: $f(n) = n^2 \lg n$. We need to check that $f(n)$ is polynomially larger than n^2.

Note that $\frac{f(n)}{n^2} = \frac{n^2 \lg n}{n^2} = \lg n = o(n^\varepsilon)$ ——— Δ

To show that Δ is true, we do $\lim_{n\to\infty} \frac{\lg n}{n^\varepsilon}$ L'Hopital's rule $\lim_{n\to\infty} \frac{\frac{1}{n \ln 2}}{\varepsilon n^{\varepsilon - 1}} = \lim_{n\to\infty} \frac{1}{\varepsilon n^\varepsilon \ln 2}$

$= 0$

Hence $\lg n = o(n^\varepsilon)$

This means the ratio $\frac{f(n)}{n^{\log_b a}} = \lg n$ is asymptotically less than n^ε for any positive constant ε. Consequently the recurrence falls into the gap between case 2 and case 3

Question 5 (2 marks) Please find the solution on the next page. Thank you!

Suppose our class receives more funding from the faculty and we can afford four class colours. Again, we choose them semi-democratically: each of the n student gets a vote and if some colour gets strictly more than $n/5$ votes, then it will be one of the class colours; if there are fewer than four such colours, then Travis will pick the remaining class colours (and he again makes no promises of fairness!) Notice that at most four colours can each get strictly more than $n/5$ votes.

We agree to use a version of **Misra and Gries' algorithm** for finding α-majorities.

(a) We start with everyone sitting down

(b) Travis will ask each student in turn their favourite colour:

(i) If the student's favourite colour is currently written on the board, the student will come down and stand at the front of the room (forming a group with anyone already there who chose the same colour)

(ii) If the student's favourite colour is not currently written on the board **and** there are already four groups supporting four different colours at the front, then the student will choose one person from each group to sit down

(iii) If the students favourite colour is not currently written on the board **but** there are fewer than four groups at the front, then the student will go to the front, if necessary erase some colour with no supporters at the front to defend it (so that there are then at most three colours on the board), and then write their colour and stay at the front starting their own group.

(c) The class colours are whatever is written on the board after Travis finishes asking students for colours.

For example:

blue	red	blue	green	green	black	white	blue
Alice	Bob	Charlie	Dave	Eve	Frank	Grace	Heidi

$8/5 = 1.6$

Front

Black ~~Frank~~

Green Dave ~~Eve~~

Red ~~Bob~~

Blue Alice ~~Charlie~~ Heidi

Travis declares that blue, red, green, and black are class colours. In this case, he must pick blue and green, since they each get more than 1.6 votes and he is free to choose black and red as well, just excluding white.

Now Travis gives one candy out to each student. Suppose one of the majority colour is red, but Travis still doesn't choose it, then they will console the red voters by giving them extra candies, according to the following rules:

1). If a student who votes for red never gets to stand at the front of the class, then, after the class ends and everyone agrees Travis made a mistake and should have chosen red, that student gets the candy of each of the four people they chose to sit down.

2). If a student who votes for red goes to the front of the class but is later chosen to sit down then, after the class ends, they get the candy of the person who chose them to sit down and each of the three other person chosen to sit down at the same time.

Why is this algorithm correct?

Proof: Without loss of generality, let's assume red is the majority colour. For the sake of contradiction, suppose red is the majority colour, but the algorithm is wrong and colours other than "red" are on the board at the end of the class. Then every one who voted red is seated. Then there can be two cases. **(1)** They have never stood at the front (remained sitted) **(2)** They have been to the front but were asked to sit down.

In **case (1)** When it was that student's turn, all rows at the front must be occupied by people choosing colours other than red. Then this student would ask one student from each row/colour group to sit down. As compensation, this student would get one candy from each one of those being-asked-to-sit-down students. After that, this student would have 5 candies.

In **case (2)**, the student choosing red has stood at the front before, but then was asked to sit down by another student. This means three other students must have been asked to sit down simultaneously. As a compensation, this student get 4 candies from others as compensation. After that, this student would have 5 candies.

In either case, for any student who voted for red, they would have 5 candies at the end of students in the class. By definition of the majority colour $n' > n/5$.
Let n' be the number of students who voted for red, and n be the total number.
It follows that $5n' > n$. This means, at the end there would be $5n'$ candies, exceeding the initial total number of candies n being given out. This yields a contradiction.
Therefore, the algorithm is correct: if more than $n/5$ people vote for red (or, by symmetry, for any other colour) it ends up as the majority colour.