a) ① use median to partition into left and right, goes to check part with smaller size

② repeat

Question 1. The Missing Number [20 MARKS]

(a) (12 marks) Suppose you are given an unsorted array A of all integers in the range 0 to n except for one integer, denoted the *missing number*. Assume $n = 2^k - 1$. Design a $O(n)$ time Divide and Conquer algorithm to find the missing number.

Justify briefly that your algorithm is correct and runs within the required time bound. (For your reference, the Master Theorem states that a recurrence of the form $T(n) = aT(n/b) + \Theta(n^d)$ has solution $\Theta(n^d)$ if $a < b^d$, $\Theta(n^d \log n)$ if $a = b^d$, and $\Theta(n^{\log_b a})$ if $a > b^d$.)

HINT: Think about the median value in the array.

Note: For full marks in part (a), your answer *must* make use of the divide-and-conquer method. Partial marks will be given for an $O(n \log n)$ divide-and-conquer method.

(b) (8 marks) Suppose the integers in A are stored as k-bit binary numbers, i.e. each bit is 0 or 1. For example, if $k = 2$ and the array $A = [01, 00, 11]$, then the missing number is 10. Now, the only operation to examine the integers is BIT-LOOKUP(i, j), which returns the j-th bit of number $A[i]$ and costs unit time.

Design any $O(n)$ algorithm to find the missing number. Justify briefly that your algorithm is correct and runs within the required time bound.

Note: An $O(nk)$ solution will receive partial credit.

a) Divide the unsorted array into two parts. left and right.
Calculate the difference sum of the integers in each part with median value $2^{k-1} - \frac{1}{2}$, if the integers larger than median value, the difference is positive. otherwise the difference would be negative, combines together, get the total difference, then add with the median value, then will get the missing number

The recurrence form: $T(n) = 2T(\frac{n}{2}) + \Theta(1)$

By master theorem $\Rightarrow T(n) \in O(n)$

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

```
findmissing(A, k)
    median = $(2^k - 1)/2$
    mid = len(A) // 2
    left = A[0, ..., mid]
    right = A[mid+1, ... ]
    if k = 0, return A[0] - median
    if k >= 1,
        left sum = findmissing(left, k)
        right sum = find missing(right, k)
        total = left sum + right sum
    return median + total
```

b) By Finding rules, I find that for every bit of binary number, the sum of each bit should be 2^{k-1} if set is complete (no missing number)

For example $k=1$ C: [1, 0] bit 1: sum is $1 = 2^{1-1}$

$k=2$, $C = [00, 01, 10, 11]$ bit 1: sum is $2 = 2^{2-1}$

bit 2: sum is $2 = 2^{2-1}$

In such a case, we can initiate an array S with length k, each index has value 2^{k-1}, then loop through A, for each element, use BIT-LOOKUP(i, j) to find each bit, then let the corresponding index j in S minus the loop up value. So Finally, combine the array element together, we get the missing number

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Array. $[\underbrace{2^k-1, \dots, 2^k-1}_{\text{length}: k}]$

```
For each element in A
    BIT-LOOKUP(element, j)   $j \in \{1, \dots, k\}$
    Array[j] -= BIT-LOOKUP(element, j).
```

d

$\leq$

Question 2. Gas Station Problem [20 MARKS]

Consider the following **Gas Station** problem: you are driving from Toronto to Montréal and you want to fill up your gas tank as few times as possible along the way (to get there as quickly as possible). Formally, you are given input $d > 0$ (the distance you can drive with a full tank of gas), $0 = g_0 < g_1 < g_2 < \cdots < g_n < g_{n+1}$ (the locations of gas stations along the way, with g_0 representing Toronto and g_{n+1} representing Montréal) such that $g_{i+1} - g_i \leqslant d$ for $i = 0, 1, \ldots, n$ (i.e., it is possible to get from each location to the next one). You want to find a list of stations $i_1 < i_2 < \cdots < i_k$ such that $g_{i_{j+1}} - g_{i_j} \leqslant d$ for $j = 0, 1, \ldots, k$ (letting $i_0 = 0$ and $i_{k+1} = n + 1$), and k is as small as possible.

(a) (7 marks) Design a greedy algorithm to solve the gas station problem. Include a clear, concise, high-level English description of the main idea behind your algorithm, followed by the pseudo-code.

(b) (1 mark) Analyze the running time of your algorithm.

(c) (12 marks) Prove the correctness of your algorithm.

a) For each selection of stop station i_j, select the station where $g_{i_j} - d \leq 0$ and $g_{i_{j+1}} - d > 0$ (which means the last station before the gas would be empty)

```
Algorithm 1: FindStop ( $d$, $\{g_0, \cdots, g_{n+1}\}$ ).
    result = { }
    travellength = 0
    For i = 1, ..., n do
        if $g_i \leq$ travellength $< g_{i+1}$ :
            result.+ = $g_i$
            travellength = $g_i + d$
        i += 1
    return result
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

b). Since It just needs loop once through whole array,
so it is $O(n)$

c) Let i_r represents the rth iterations

Base Case: $n=0$, g_1 is Montreal
according to question. $g_1 - g_0 \leq d$
$\Rightarrow$ stop set is $\{\}$ (optimal solution)
By a) solution, result is also $\{\}$, which is correct

Inductive Hypothesis:
Assume in kth iteration, the result set. r_k is optimal
Inductive step want to prove r_{k+1} is also optimal. in i_{k+1} th iteration
Case 1: In optimal solution, g_{k+1} should not be stop station.
so in Qa) algorithm, it means, $g_{k+1} \leq travelDistance < g_{k+2}$ is not successed.
subcase 1: $g_{k+1} > travelDistance = g_i + d$, so $g_{k+1} - g_i > d$
since for each stop, $g_j - g_{j-1} \leq d$, so there must be stop stations between g_{k+1} and g_i where the car can stop here and do not need to stop at g_{k+1}.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

subcase 2: $g_{k+1} <$ travel distance $= g_i + d$
and $g_{k+2} \leq$ travel distance $= g_i + d$

If g_{k+2} is closer to the end where the tank would be empty, then the driver would prefer g_{k+2} as stop instead of g_{k+1}. so g_{k+1} is still not in the optimal solution

Case 2: g_{k+1} in optimal solution as stop station

$\Rightarrow g_{k+1} \leq$ travel Distance $< g_{k+2}$

so that means g_{k+1} is the closest station where the tank would be empty, so driver will stop at g_{k+1}, where by ~~Qa~~ algorithm, will also be included in the solution.

Question 3. Grade Maximization [20 MARKS]

It's nearing the end of the semester and you have n final projects to complete. Your goal, of course, is to maximize your total grade on these projects! For simplicity, say that you have a total of H hours (a positive integer) to work on the projects cumulatively, and you'll spend an integer nu~~mber of hours~~ on each project. To figure out how best to divide up your time, you've come up with a set of non-decreasing estimate functions $\{e_1, e_2, \dots, e_n\}$, one for each project: if you spend h hours on project ~~i (where $0 \leqslant h \leqslant H$)~~, you'll get a grade of $e_i(h)$ on that project.

The task is to come up with a dynamic programming algorithm that takes the estimate functions $\{e_1, \dots, e_n\}$ as input and outputs the maximum possible grade that you can achieve.

(a) (5 marks) Define the array(s) or table(s) that your solution would compute. Clearly explain what each entry means, and how you would compute the final answer given all the entries in your array(s) or table(s).

(b) (7 marks) Write a Bellman equation and briefly justify its correctness.

(c) (4 marks) Describe a bottom-up implementation of your Bellman equation, and analyze its worst-case running time and space complexity.

(d) (4 marks) Now, modify your algorithm to output the optimal *division* itself i.e., your algorithm should generate non-negative numbers of hours $\{h_1, \dots, h_n\}$ such that $h_1 + h_2 + \dots + h_n = H$ and your total grade $e_1(h_1) + e_2(h_2) + \dots + e_n(h_n)$ is maximum. You may define an additional array to do so, in which case, specify its definition clearly. Re-analyze the running time and space complexity of the modified algorithm.

a). Define momoization array to be 2D, first index is the total hours can be used, second index is the number of projects you have (start from 1)

The desired solution is $M[H, n]$

b). $$M[h, i] = \begin{cases} 0, & \text{if } h = 0 \text{ or } i = 0 \\ \max\{M[h, i-1],\ e_i(h_j) + M[h-h_j, i-1]\} \end{cases}$$

where j is the smallest integer that maximize e_i

CONT'D...

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

```
c). Get MaxGrade(H, [e_1, ..., e_n])
        res = 0
        threshold = [ ]
        For i=1, ..., n do:
            for j = 0, ..., H.
                if e_i(j) == e_i(j+1))
                    threshold.append(j)
                threshold.append(H).
    # threshold contains all the minimum value that
        makes e_i maximum
    M[0, n] = 0 for all n.
    For k=1, ..., H:
        M[k, 0] = 0
        M[k, 1] = threshold[0]
        For j=2, ..., n:
            M[k,j] = max[M[k,j-1], e_j(threshold[j])
                                + M[k-threshold[j], j-1]

    return M[H, n]
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 3. Grade Maximization [20 MARKS]

It's nearing the end of the semester and you have n final projects to complete. Your goal, of course, is to maximize your total grade on these projects! For simplicity, say that you have a total of H hours (a positive integer) to work on the projects cumulatively, and you'll spend an integer ~~number of hours on~~ each project. To figure out how best to divide up your time, you've come up with a set of non-decreasing estimate functions $\{e_1, e_2, \ldots, e_n\}$, one for each project: if you spend h hours on project i ~~(where $0 \leqslant h \leqslant H$)~~, you'll get a grade of $e_i(h)$ on that project.

The task is to come up with a dynamic programming algorithm that takes the estimate functions $\{e_1, \ldots, e_n\}$ as input and outputs the maximum possible grade that you can achieve.

(a) (5 marks) Define the array(s) or table(s) that your solution would compute. Clearly explain what each entry means, and how you would compute the final answer given all the entries in your array(s) or table(s).

(b) (7 marks) Write a Bellman equation and briefly justify its correctness.

(c) (4 marks) Describe a bottom-up implementation of your Bellman equation, and analyze its worst-case running time and space complexity.

(d) (4 marks) Now, modify your algorithm to output the optimal *division* itself i.e., your algorithm should generate non-negative numbers of hours $\{h_1, \ldots, h_n\}$ such that $h_1 + h_2 + \cdots + h_n = H$ and your total grade $e_1(h_1) + e_2(h_2) + \cdots + e_n(h_n)$ is maximum. You may define an additional array to do so, in which case, specify its definition clearly. Re-analyze the running time and space complexity of the modified algorithm.

a) $M[i][j]$ — i — # of hours you have
— project $1, 2, \ldots, j$

$M[H][n]$ is sol

b)
$$M[i][j] = \begin{cases} 0, & i=0 \,/\, j=0 \\ \max\{M[i][j-1],\ M[i-h][j-1] + e_j(h)\} \end{cases}$$

CONT'D...