- Please submit before the submission deadline.
- Assignment submitted one (1) day after the assignment deadline will be accepted with 20% deduction on corresponding assignment grade.
- Assignment submitted more than one (1) day late will not be accepted.

Problem 1: Greedy algorithm (25 points)

Let N be a positive integer. Given $2N$ ordered reals $p_1 < p_2 < \cdots < p_{2N}$ and $M(\leq N)$, you are tasked with getting $\min_{\{i_j\}} \sum_{1\leq j\leq M}(p_{i_j+1} - p_{i_j})$, where $1 \leq i_j < 2N$ and $|i_j - i_k| \geq 2$ for every $j, k \in \{1, \ldots, M\}, j \neq k$.
Design an algorithm that calculates the minimum value with a time complexity of $O(N \log N)$.

Solution. Even if (p_i, p_{i+1}) has the smallest difference $p_{i+1} - p_i$ value, it does not mean that (p_i, p_{i+1}) must be used to calculate the minimum value. (counter example: $N = 3, M = 2, p_1 = 0, p_2 = 2, p_3 = 3, p_4 = 5, p_5 = 200, p_6 = 300$. Although $p_3 - p_2$ is the smallest, the sum of $p_2 - p_1$ and $p_4 - p_3$ gives the minimum value.)

However, if it is not used, its two adjacent pairs must be used together. Because, if only one of its adjacent pair is used to calculate the minimum value, replacing the pair into (p_i, p_{i+1}) can make the sum smaller, so the sum cannot be the minimum. Else if both of the pairs are not used, still (p_i, p_{i+1}) can replace a pair, making the sum smaller. **(Criteria 1)**

Since the pair with the smallest difference is used or its two adjacent pairs are used together, we can design the following algorithm by repeating this process.

Algorithm 1 Solution for Problem 1

Input: $N, M, p_1, \ldots, p_{2N}$
Output: $\min_{\{i_j\}} \sum_{1\leq j\leq M}(p_{i_j+1} - p_{i_j})$

`left`$[i] \leftarrow i - 1$ for all $i \in \{1, \ldots, 2N\}$ ▷ $O(N)$
`right`$[i] \leftarrow i + 1$ for all $i \in \{1, \ldots, 2N\}$ ▷ $O(N)$
`diff`$[i] \leftarrow p_{i+1} - p_i$ for all $i \in \{1, \ldots, 2N - 1\}$ ▷ $O(N)$
$m \leftarrow 0$ ▷ Counts how many pairs are added
`ret` $\leftarrow 0$ ▷ Initialize the return value as 0
$H \leftarrow$ makequeue(key : $p_{i+1} - p_i$, value : $(p_{i+1} - p_i, i, i + 1)$ for all $i \in \{1, \ldots, 2N - 1\}$) ▷ $O(N \log N)$
while $m < M$ **do** ▷ Iterate $O(N)$ times
  $(v, l, r) \leftarrow$ deletemin(H) ▷ $O(\log N)$
  if $l =$ `right`$[l]$ and $r =$ `left`$[r]$ **then** ▷ If reflects the latest status
    `ret` $\leftarrow$ `ret` $+ v$
    $m \leftarrow m + 1$
    $l' \leftarrow$ `left`$[l]$
    $r' \leftarrow$ `right`$[r]$
    `diff`$[l'] \leftarrow$ `diff`$[l'] +$ `diff`$[r] - v$
    `left`$[r'] \leftarrow l'$
    `right`$[l'] \leftarrow r'$
    if $1 \leq l'$ and $r' \leq 2N$ **then**
      add(H, key : `diff`$[l']$, value : (`diff`$[l'], l', r'$)) ▷ $O(\log N)$
return `ret`

Grading Criteria:
(+10 points) Correct algorithm with optimal time complexity.
(+5 points) Correctly working with suboptimal time complexity.
(+5 points) Identifying the **Criteria 1**.
(+5 points) Used greedy algorithm.

Problem 2: Huffman Encoding (25 points)

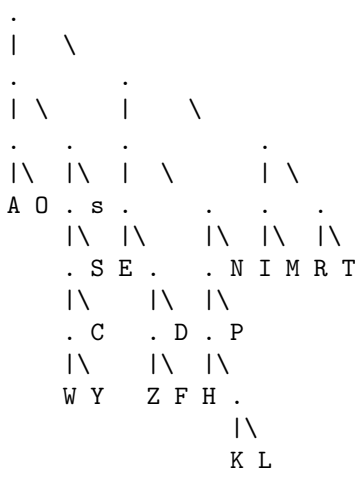
Apply Huffman encoding to the sentence provided below.

HARDWARE LIMITATIONS AND MEMORY CONSTRAINTS MAKE DATA COMPRESSION METHODS IMPORTANT FOR PERFORMANCE OPTIMIZATION

(a) (5 points) Complete the frequency table of the characters (including the space character).

Char.	' '	A	C	D	E	F	H	I	K	L	M	N	O	P	R	S	T	W	Y	Z
Freq.	12	11	3	4	7	2	2	9	1	1	9	8	11	4	9	6	10	1	1	1

(b) (5 points) Construct a Huffman tree.



(s denotes the space character.)

(c) (5 points) Generate the binary codes for each character.

Char.	' '	A	C	D	E	F	H	I	K	L
Code.	011	000	01001	10011	1000	100101	101000	1100	1010010	1010011
Char.	M	N	O	P	R	S	T	W	Y	Z
Code.	1101	1011	001	10101	1110	0101	1111	010000	010001	100100

(d) (10 points) Calculate the compression ratio ($\frac{\text{\# original bits - \# encoded bits}}{\text{\# original bits}}$) compared to the original sentence. (Note that each character takes 8 bits.)

Bits for characters: $8 \times 112 = 896$, bits for encoded one: 445. $\rightarrow \frac{896-445}{896} = 50.33\%$.

Grading Criteria:
(Full credit) Correct answer. The answer may be different from the solution here in (b) and (c).
(+5 points) Correct answer based from the previous results.

Problem 3: Dynamic Programming (50 points)

(a) (25 points) You are tasked with calculating the number of binary trees with $N(\geq 1)$ nodes and a height $H(\geq 1)$.

A binary tree consisting of only the root node is considered to have a height of 1. Also, the left and right subtrees of each node are ordered and distinct. For instance, with $N = 2$ and $H = 2$, the number of binary trees is 2, not 1.

Design an algorithm with a memory usage of $O(NH)$ and a time complexity of $O(N^2H)$.

Solution. A binary tree with N nodes and height that H is uniquely determined by the left and right subtrees of the root node, and the larger height of the left and right subtrees is $H - 1$. However, it's not guaranteed that both subtrees have height $H - 1$, meaning the subproblems differ from the original problem definition. To apply dynamic programming, we can slightly modify the problem to instead *find the number of binary trees with N nodes and a height* ***at most*** H. **(Criteria 1)** Then, the left and right subtrees has at most $H - 1$.

Since the total nodes of the left and right subtrees are $N - 1$, it suffices to sum all possible cases making the sum of nodes be $N - 1$.

Algorithm 2 Solution for Problem 3-a

Input: N, H
Output: # binary trees with N nodes and a height H
$\texttt{num}[n, h] \leftarrow 0$ for all $n \leq N, h \leq H$ ▷ $O(NH)$ memory
for all $h = 1, \ldots, H$ **do** ▷ $O(H)$ time
  for all $n = 0, \ldots, \min(N, 2^h - 1)$ **do** ▷ $O(N)$ time
    if $n = 0$ or $n = 2^h - 1$ **then**
      $\texttt{num}[n, h] = 1$
    else
      for all $i = 0, \ldots, n - 1$ **do**
        $\texttt{num}[n, h] = \texttt{num}[n, h] + \texttt{num}[i, h - 1] \times \texttt{num}[n - 1 - i, h - 1]$ ▷ $O(N)$ time
return $\texttt{num}[N, H] - \texttt{num}[N, H - 1]$

The time complexity of this algorithm is $O(N^2H)$.

Grading Criteria:
(Full credit) The algorithm correctly yields the values.
(+10 points) Identifying the $\texttt{num}[n, h]$ formula.
(+10 points) Identifying the **Criteria 1**.
(−10 points) Incorrect time complexity.

(b) (25 points) Given $M(\in \mathbb{N})$, you are tasked with calculating the number of non-increasing sequences $p_1 \geq p_2 \geq p_3 \geq \dots$ of integers such that $\sum_i p_i = M$ and $p_i = \sum_j \mathbb{1}\{p_j \geq i\}$ where i is a positive integer, and $\mathbb{1}$ is an indicator function.

Design an algorithm that computes the number of sequences with a time complexity of $O(M^{3/2})$.

Solution. Let k be the maximal i such that $p_i \geq i$. $(M \geq 1 \Rightarrow k \geq 1)$ Then, for $m \leq k$,

$$p_m = \sum_j \mathbb{1}\{p_j \geq m\} = \sum_{j \leq k} \mathbb{1}\{p_j \geq m\} + \sum_{j>k} \mathbb{1}\{p_j \geq m\}.$$

If $j \geq k$, then $p_j \geq p_k \geq k \geq m$. The above one becomes $p_m = k + \sum_{j>k} \mathbb{1}\{p_j \geq m\}$. Which means, $p_m (m \geq k)$ is uniquely determined by $p_j (j > k)$.

In addition, to utilize the fact $\sum_i p_i = M$, let us calculate $\sum_{1 \leq m \leq k} p_m$.

$$\sum_{1 \leq m \leq k} p_m = \sum_{1 \leq i \leq k} (k + \sum_{j>k} \mathbb{1}\{p_j \geq m\}) = k^2 + \sum_{1 \leq m \leq k} \sum_{j>k} \mathbb{1}\{p_j \geq m\}.$$

Since $p_{k+1} < k+1$, we know $p_j \leq p_{k+1} \leq k$ for $j > k$. Thus the above one can be summarized as follows:

$$\sum_{1 \leq m \leq k} p_m = k^2 + \sum_{j>k} \sum_{1 \leq m \leq k} \mathbb{1}\{p_j \geq m\} = k^2 + \sum_{1 \leq m \leq k} \sum_{j>k} \mathbb{1}\{p_j \geq m\} = k^2 + \sum_{j>k} p_j.$$

Then we can utilize $\sum_i p_i = M$ as follows:

$$M = \sum_i p_i = \sum_{1 \leq i \leq k} p_i + \sum_{i>k} p_i = k^2 + 2\sum_{i>k} p_j \Rightarrow 2\sum_{i>k} p_j = M - k^2.$$

So, it suffices to find the number of possible non-increasing p_i's $(i > k)$ such that $p_{k+1} <= k$ and $2\sum_{i>k} p_j = M - k^2$.

Simply speaking, we need to know the number of non-increasing non-negative integer sequence q_i such that $q_1 \leq K$ and $\sum_i q_i = N$ for specific $K \geq 1$ and $N \geq 0$. **(Criteria 1)** Since $k^2 \leq M$, the problem suffices to be solved for $K \leq \sqrt{M}$.

For given N and K, if q_1 is set to be K, then the next values have to satisfy $q_2 \leq K$ and $\sum_{i \geq 2} q_i = N - K$, and this is identical to the problem with $N - K$ and K. Else if q_1 is smaller than K, then it becomes the problem with N and $K - 1$. **(Criteria 2)**

Algorithm 3 Solution for Problem 3-b

Input: M
Output: # $\{p_i\}$
$\texttt{num}[n, k] \leftarrow 0$ for all $n \leq M, k \leq \lfloor\sqrt{M}\rfloor$ ▷ $O(M^{3/2})$ memory
for all $n = 0, \dots, M$ **do** ▷ $O(M)$ time
  for all $k = 1, \dots, \lfloor\sqrt{M}\rfloor$ **do** ▷ $O(M^{1/2})$ time
    if $n \leq 1$ or $k = 1$ **then**
      $\texttt{num}[n, k] = 1$
    else if $k \leq n$ **then**
      $\texttt{num}[n, k] = \texttt{num}[n, k-1] + \texttt{num}[n-k, k]$
    else
      $\texttt{num}[n, k] = \texttt{num}[n, k-1]$
$\texttt{ret} \leftarrow 0$ ▷ Initialize the return value as 0
for all $i = 1, \dots, \lfloor\sqrt{M}\rfloor$ **do** ▷ $O(M^{1/2})$ time
  if $M - i^2 \equiv 0 \mod 2$ **then**
    $\texttt{ret} \leftarrow \texttt{ret} + \texttt{num}[\frac{M-i^2}{2}, i]$
return ret

The time complexity of this algorithm is $O(M^{3/2})$.

Grading Criteria:
(Full credit) The algorithm correctly yields the values.
(+10 points) Identifying the **Criteria 2**.
(+10 points) Identifying the **Criteria 1**.
(−10 points) Incorrect time complexity.