# Assignment 5

**Total 80 points**

**Each question carries 10 marks**

**Question 1: String to Integer (atoi)**

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit charcter or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range $[-2^{31}, 2^{31} - 1]$, then clamp the integer so that it remains in the range. Specifically, integers less than $-2^{31}$ should be clamped to $-2^{31}$, and integers greater than $2^{31} - 1$ should be clamped to $2^{31} - 1$.
6. Return the integer as the final result.

**Note:**

- Only the space character ' ' is considered a whitespace character.
- **Do not ignore** any characters other than the leading whitespace or the rest of the string after the digits.

**Example 1:**

**Input:** s = "42"**Output:** 42**Explanation:** The underlined characters are what is read in, the caret is the current reader position.Step 1: "42" (no characters read because there is no leading whitespace)         ^Step 2: "42" (no characters read because there is neither a '-' nor '+') ^Step 3: "<u>42</u>" ("42" is read in)           ^The parsed integer is 42.Since 42 is in the range $[-2^{31}, 2^{31} - 1]$, the final result is 42.

**Example 2:**

**Input:** s = "   -42"**Output:** -42**Explanation:**Step 1: "__-42" (leading whitespace is read and ignored)            ^Step 2: "   -42" ('-' is read, so the result should be negative)            ^Step 3: "   -42" ("42" is read in)               ^The parsed integer is -42.Since -42 is in the range [-$2^{31}$, $2^{31}$ - 1], the final result is -42.

**Example 3:**

**Input:** s = "4193 with words"**Output:** 4193**Explanation:**Step 1: "4193 with words" (no characters read because there is no leading whitespace)         ^Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')         ^Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)            ^The parsed integer is 4193.Since 4193 is in the range [-$2^{31}$, $2^{31}$ - 1], the final result is 4193.

**Example 4:**

**Input:** s = "words and 987"**Output:** 0**Explanation:**Step 1: "words and 987" (no characters read because there is no leading whitespace)         ^Step 2: "words and 987" (no characters read because there is neither a '-' nor '+')         ^Step 3: "words and 987" (reading stops immediately because there is a non-digit 'w')         ^The parsed integer is 0 because no digits were read.Since 0 is in the range [-$2^{31}$, $2^{31}$ - 1], the final result is 0.

**Example 5:**

**Input:** s = "-91283472332"**Output:** -2147483648**Explanation:**Step 1: "-91283472332" (no characters read because there is no leading whitespace)         ^Step 2: "-91283472332" ('-' is read, so the result should be negative)         ^Step 3: "-91283472332" ("91283472332" is read in)                   ^The parsed integer is -91283472332.Since -91283472332 is less than the lower bound of the range [-$2^{31}$, $2^{31}$ - 1], the final result is clamped to -$2^{31}$ = -2147483648.

**Constraints:**

- 0 <= s.length <= 200
- s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '.'.

**Question 2: Reformat Date**

Given a date string in the form Day Month Year, where:

- Day is in the set {"1st", "2nd", "3rd", "4th", ..., "30th", "31st"}.
- Month is in the set {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"}.

- Year is in the range [1900, 2100].

Convert the date string to the format YYYY-MM-DD, where:

- YYYY denotes the 4 digit year.
- MM denotes the 2 digit month.
- DD denotes the 2 digit day.

**Example 1:**

**Input:** date = "20th Oct 2052"**Output:** "2052-10-20"

**Example 2:**

**Input:** date = "6th Jun 1933"**Output:** "1933-06-06"

**Example 3:**

**Input:** date = "26th May 1960"**Output:** "1960-05-26"

Q**uestion 3: Valid Paranthese**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

7. Open brackets must be closed by the same type of brackets.
8. Open brackets must be closed in the correct order.

**Example 1:**

**Input:** s = "()"**Output:** true

**Example 2:**

**Input:** s = "()[]{}"**Output:** true

**Example 3:**

**Input:** s = "(]"**Output:** false

**Example 4:**

**Input:** s = "([)]"**Output:** false

**Example 5:**

**Input:** s = "{[]}"**Output:** true

**Constraints:**

- 1 <= s.length <= $10^4$
- s consists of parentheses only '()[]{}'.

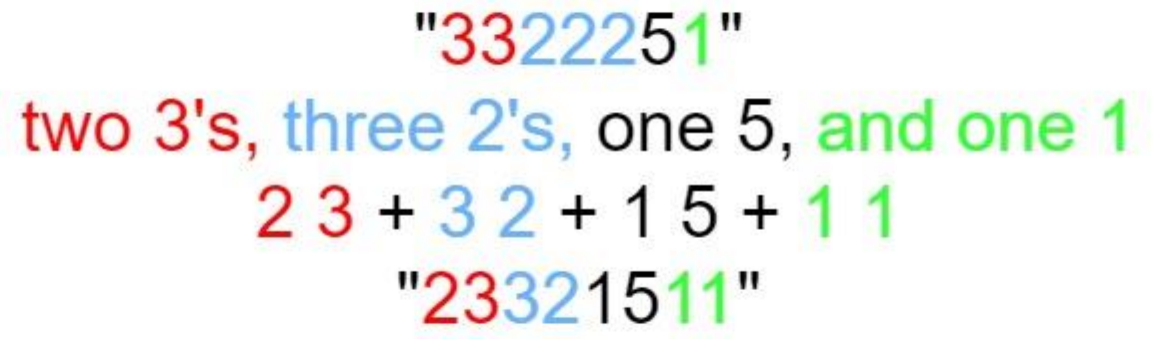
**Question 4: Count and Say**

The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

- countAndSay(1) = "1"
- countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of groups so that each group is a contiguous section all of the **same character.** Then for each group, say the number of characters, then say the character. To convert the saying into a digit string, replace the counts with a number and concatenate every saying.

For example, the saying and conversion for digit string "3322251":



Given a positive integer n, return *the* n$^{th}$ *term of the **count-and-say** sequence*.

The following are the terms from n=1 to n=10 of the count-and-say sequence:

1.     1 2.     11 3.     21 4.     1211 5.     111221  6.     312211 7.     13112221 8.     1113213211 9.     31131211131221 10.     13211311123113112211

**Example 1:**

**Input:** n = 1**Output:** "1"**Explanation:** This is the base case.

**Example 2:**

**Input:** n = 4**Output:** "1211"**Explanation:**countAndSay(1) = "1"countAndSay(2) = say "1" = one 1 = "11"countAndSay(3) = say "11" = two 1's = "21"countAndSay(4) = say "21" = one 2 + one 1 = "12" + "11" = "1211"

**Constraints:**

- 1 <= n <= 30

Question 5: **Best Time to Buy and Sell Stock**

You are given an array prices where prices[i] is the price of a given stock on the $i^{th}$ day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return 0.

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]**Output:** 5**Explanation:** Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

**Example 2:**

**Input:** prices = [7,6,4,3,1]**Output:** 0**Explanation:** In this case, no transactions are done and the max profit = 0.

**Constraints:**

- 1 <= prices.length <= $10^5$
- 0 <= prices[i] <= $10^4$

**Question 6: Find Pivot Index**

Given an array of integers nums, calculate the **pivot index** of this array.

The **pivot index** is the index where the sum of all the numbers **strictly** to the left of the index is equal to the sum of all the numbers **strictly** to the index's right.

If the index is on the left edge of the array, then the left sum is 0 because there are no elements to the left. This also applies to the right edge of the array.

Return *the **leftmost pivot index***. If no such index exists, return -1.

**Example 1:**

**Input:** nums = [1,7,3,6,5,6]**Output:** 3**Explanation:**The pivot index is 3.Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11Right sum = nums[4] + nums[5] = 5 + 6 = 11

**Example 2:**

**Input:** nums = [1,2,3]**Output:** -1**Explanation:**There is no index that satisfies the conditions in the problem statement.

**Example 3:**

**Input:** nums = [2,1,-1]**Output:** 0**Explanation:**The pivot index is 0.Left sum = 0 (no elements to the left of index 0)Right sum = nums[1] + nums[2] = 1 + -1 = 0

**Constraints:**

- 1 <= nums.length <= $10^4$
- -1000 <= nums[i] <= 1000

**Question 7: High Five**

Given a list of the scores of different students, items, where items[i] = [$\text{ID}_i$, $\text{score}_i$] represents one score from a student with $\text{ID}_i$, calculate each student's **top five average**.

Return *the answer as an array of pairs* result*, where* result[j] = [$\text{ID}_j$, $\text{topFiveAverage}_j$] *represents the student with* $\text{ID}_j$ *and their **top five average**. Sort* result *by* $\text{ID}_j$ *in **increasing order**.*

A student's **top five average** is calculated by taking the sum of their top five scores and dividing it by 5 using **integer division**.

**Example 1:**

**Input:** items = [[1,91],[1,92],[2,93],[2,97],[1,60],[2,77],[1,65],[1,87],[1,100],[2,100],[2,76]]**Output:** [[1,87],[2,88]]**Explanation:** The student with ID = 1 got scores 91, 92, 60, 65, 87, and 100. Their top five average is (100 + 92 + 91 + 87 + 65) / 5 = 87.The student with ID = 2 got scores 93, 97, 77, 100, and 76. Their top five average is (100 + 97 + 93 + 77 + 76) / 5 = 88.6, but with integer division their average converts to 88.

**Example 2:**

**Input:** items = [[1,100],[7,100],[1,100],[7,100],[1,100],[7,100],[1,100],[7,100],[1,100],[7,100]]**Output:** [[1,100],[7,100]]

**Constraints:**

- 1 <= items.length <= 1000
- items[i].length == 2
- 1 <= $ID_i$ <= 1000
- 0 <= $score_i$ <= 100
- For each $ID_i$, there will be **at least** five scores.

**Question 8: Search in Rotated Sorted Array**

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.

You must write an algorithm with O(log n) runtime complexity.

**Example 1:**

**Input:** nums = [4,5,6,7,0,1,2], target = 0**Output:** 4

**Example 2:**

**Input:** nums = [4,5,6,7,0,1,2], target = 3**Output:** -1

**Example 3:**

**Input:** nums = [1], target = 0**Output:** -1

**Constraints:**

- 1 <= nums.length <= 5000
- $-10^4$ <= nums[i] <= $10^4$
- All values of nums are **unique**.
- nums is an ascending array that is possibly rotated.
- $-10^4$ <= target <= $10^4$