Programming Challenge

Welcome to the DNAnexus programming challenge! There are four questions, arranged in order of difficulty. We suggest that you do the problems in the order in which they are given. You have two hours to complete as many of the problems as you can, and send us your code.

We hope you'll find this to be a fun, challenging test of your skills. We understand that some people find these problems to be a bit dry, and maybe too simple. If you think so, great! We're looking for awesome coders like you. Even if you find these problems to be easy, your solutions still give us a lot of insight about your strengths, weaknesses, and coding style.

If you find anything confusing, just try to solve the problem as best you can. There isn't necessarily a single "right" answer – we are interested in seeing how you approach solving these types of problems. Feel free to send us any feedback you have about these problems and suggestions for making it clearer!

We are interested in your ability to write clean, correct code to solve the problem at hand. You may write the code in the language of your choice (please, nothing too exotic!), but it should be running code, not pseudocode. You're free to use standard library functions of the language you're using, but don't use a function that directly solves the problem (use your judgement). Please send us your original source code, preferably with comments explaining what it's doing. Although it's not necessary, if you also created any test stubs and test data, feel free to send that as well, and don't forget the piece of code you're proud of. Good luck!

Problem 1: DNA sequence conversion

DNA is a long molecule that lies inside the nucleus of a cell, and it can be thought of as a very long string consisting of characters in the alphabet {A, C, G, T}. DNA sequencing is the technology that enables reading from DNA molecules and converting them to strings on the output. We are interested in a technology that works in the following way: the DNA molecules in the input are fragmented into pieces of equal length L; each piece is then sequenced by the technology, and its content is encoded in the output. The particular encoding used in the output is the following:

- The file contains multiple consecutive entries, one per piece.
- Each piece is represented by L consecutive bytes (1 byte = 8 bits).
- The first two (most significant) bits of each byte encode the DNA letter:
    - 00 represents A
    - 01 represents C

    - 10 represents G
    - 11 represents T
- The last six (least significant) bits of each byte encode the confidence that the readout was correct, also known as the quality score. It is represented as an unsigned 6-bit integer in the range 0 to 63.

Write a program that takes as input an encoded file as well as the number L, and converts it to a text file of the following format (known as the FASTQ format):

- Each piece is represented by four lines:
    - The first line contains the word `@READ_` followed by the piece index. The first piece has an index of 1, so its first line would be `@READ_1`
    - The second line contains L characters in the {A,C,G,T} alphabet, representing the DNA sequence of the piece.
    - The third line contains the word `+READ_` followed by the piece index (e.g., `+READ_1`).
    - The fourth line contains L characters, representing the quality scores of the piece. Each score is represented as an ASCII character in the range 33-96, by adding 33 to the original score. For example, if the original score is 0, it should be represented by the ASCII character 33 (“!”)

Example input (for L=2) shown here in binary:
`00000000 11100000 11000001 01111111`

Example output:

```
@READ_1
AT
+READ_1
!A
@READ_2
TC
+READ2
"`
```

Please note that input shall be a binary sequence of bytes -- not a string of ‘0’ and ‘1’ characters. An example input file can be downloaded from https://classic.dnanexus.com/programming_challenges/dna_conversion_samples/input. This example file has 1680 bytes. One can generate different output with different L values. The example output for L = 7, 15, and 80 can also be downloaded from the following locations:

L = 7: https://classic.dnanexus.com/programming_challenges/dna_conversion_samples/output7
L = 15:
https://classic.dnanexus.com/programming_challenges/dna_conversion_samples/output15
L = 80:
https://classic.dnanexus.com/programming_challenges/dna_conversion_samples/output80

Problem 2: Search

You are given an array **A** of **n** integers. The elements of **A** are sorted in ascending order, and are not necessarily unique. You are also given a target integer **x**. Implement a search algorithm which finds a location **j** in the array such that all elements in the range **A[0], …, A[j - 1]** are strictly less than **x**, and all elements in the range **A[j], …, A[n - 1]** are greater than or equal to **x**. (Note that **x** itself need not appear in **A** for these conditions to be satisfied.) If no suitable location is found, return **-1**. Your solution should have runtime **O(log n)**.

Problem 3: Random line from a file

You are given a very, very large plain text file where each line contains a plain text string. The file has at most 1 billion lines; lines may have different lengths, but each line has at most 1000 characters. Your goal is to write a program that will print an arbitrary line from the file. Your program will be run many times (although you don't know exactly how many times it will be run in advance), and you don't know in advance which lines might be selected. Thus, your solution should be optimized to minimize the runtime for each additional execution. The first execution of the program may take longer than subsequent runs, and you may use additional disk storage to improve performance.

Your program should take two command-line arguments: the path of the file from which to print lines, and the index of the line you want to print. Your program should write the line to standard output.

Sample input/output:

input_file.txt:

```
apple
banana
orange
lemon
```

```
$ random_line input_file.txt 3
```

on stderr: `Writing index to input_file.txt.idx... done.`

```
lemon
```

```
$ random_line input_file.txt 2
orange
$ random_line input_file.txt 0
apple
```

Problem 4: Algorithm

You are given a sequence of **N** numbers **s[1], s[2], ..., s[N]**. A contiguous subsequence is defined as follows: for any **i** and **j** with **1 <= i <= j <= N**, the sequence of numbers **s[i], s[i+1], …, s[j]**. Write a program that takes a sequence of numbers and finds the contiguous subsequence with the largest sum and outputs **i** and **j**. If there are multiple such subsequences, return the **i** and **j** for the shortest subsequence which occurs first in the original sequence. Try to use the most efficient algorithm, and describe the time and space complexity of your solution.