# 2016 APL Problem Solving Competition – Phase 2 Problem Descriptions

## Overview

This year's Phase II problems are divided into three sets, representing three categories – Bioinformatics, Finance and General Computing. Each set has three problems; a problem can be broken down into multiple tasks.

Three grand prizes will be awarded, one in each of the categories.
**To be considered for a grand prize, you must solve all of the problems in a category.**

You may submit an entry that completes more than one category.  However, you can only win once – for example, if your entries in all three categories are judged to be the best in each category, the judging committee will select the entry they deem the best and declare you the winner in that category while other competitors will be selected as the winners of the other categories.

Good Luck and Happy Problem Solving!

**Note:**
Some of the examples are displayed using the user command setting `]boxing on` to more clearly depict the structure of the displayed data.

```
      ('Dyalog' 'APL')(4 4⍴⍳16) 5
  Dyalog  APL     1  2  3  4  5
                  5  6  7  8
                  9 10 11 12
                 13 14 15 16

      ]boxing on
Was OFF

      ('Dyalog' 'APL')(4 4⍴⍳16) 5
┌────────────┬───────────┬─┐
│┌──────┬───┐│ 1  2  3  4│5│
││Dyalog│APL││ 5  6  7  8│ │
│└──────┴───┘│ 9 10 11 12│ │
│            │13 14 15 16│ │
└────────────┴───────────┴─┘
```

## Description of the Contest2016 Template Files

Two template files are available for download from the contest website. Which file you use depends on how you choose to implement your problem solutions.

---

### If you use Dyalog APL, you should use the template workspace `Contest2016.DWS`

The Contest2016 workspace contains:

- Three namespaces, one for each of the problem categories. Each namespace will contain:
  - o stubs for all of the functions described in the problem descriptions. The function stubs are implemented as traditional APL functions (tradfns) but you can change the implementation to dynamic functions (dfns) if you want to do so. Either form is acceptable.
  - o any sample data elements mentioned in the problem descriptions.

  Any sub-functions that you develop as a part of your solution should be co-located in the category namespace.

  The namespaces are:

  - o `#.bio` – for the bioinformatics problem set
  - o `#.fin` – for the finance problem set
  - o `#.gen` – for the general computing problem set

- **`#.SubmitMe`** – a function used to package your solution for submission.

**Make sure you save your work using the `)SAVE` system command!**

Once you have developed and are ready to submit your solutions, run the **`#.SubmitMe`** function, enter the requested information and click the **Save** button. **`#.SubmitMe`** will create a file called **Contest2016.dyalog** which will contain any code or data you placed in the **`#.bio`**, **`#.fin`**, and **`#.gen`** namespaces. You will then need to upload the **Contest2016.dyalog** file using the contest website.

---

### If you use some other APL system, you can use the template script file `Contest2016.dyalog`

This file contains the correct structure for submission. You can populate it with your code, but do not change the namespace structure. Once you have developed your solution, edit the variable definitions as indicated at the top of the file and upload the file using the contest website. If you use some other APL system to develop your application, **it will still need to execute under Dyalog APL**; it is recommended that your solution use APL features that are common between your APL system and Dyalog.

## Set 1: Bioinformatics Problems

The Bioinformatics problems presented here are based on problems presented on the website http://rosalind.info. The descriptions have been adapted to be more suitable for APL syntax and this competition.

Please note that your solutions should perform on arguments as described in the "**Given:"** section of the problem descriptions on rosalind.info. For example, in "Translating RNA into Protein" it states:

"**Given:** An RNA string s corresponding to a strand of mRNA (of length at most 10 kbp)."

This means your solution should work with arguments up to 10,000 base pairs.

### Bioinformatics Problem 1 (1 task)

#### *Task 1 – Comparing Spectra with the Spectral Convolution*

Suppose you have two mass spectra, and you want to check if they both were obtained from the same protein; you will need some notion of spectra similarity.

The complete description of this task can be found at http://rosalind.info/problems/conv/.

Write an APL function named **maxMult** which:

- takes vectors of positive real numbers representing multisets as its left and right arguments
- returns a 2-element vector of
    - [1] the largest multiplicity of the inputs
    - [2] the absolute value of the number maximizing the shared masses of the inputs

Example:

```
    s1←186.07931 287.12699 548.20532 580.18077 681.22845 706.27446 782.27613
968.35544 968.35544
    s2←101.04768 158.06914 202.09536 318.09979 419.14747 463.17369

    s1 maxMult s2
3 85.0163
```

## Bioinformatics Problem 2 (1 task)

### *Task 1 – Interleaving Two Motifs*

In this problem, we are given two different motifs and we wish to interleave them together in such a way as to produce a shortest possible genetic string in which both motifs occur as subsequences.

The complete description of this problem can be found at http://rosalind.info/problems/scsp/.

Write an APL function named `interleave` which:

- takes character vectors representing DNA strings as its left and right arguments
- returns a character vector representing a shortest common supersequence of the arguments

Example:

```
      'ATCTGAT' interleave 'TGCATA'
ATGCATGAT
```

Example Explained:

The result represents the shortest sequence in which the two arguments are presented completely and in order.

```
Left argument:    AT C TGAT
Result:           ATGCATGAT
Right argument:    TGCAT A
```

## Bioinformatics Problem 3 **(1 task)**

### *Task 1 – Sorting by Reversals*

Perhaps the most common type of genome rearrangement is an inversion, which flips an entire interval of DNA found on the same chromosome. In this problem we would like to determine the minimum number of inversions that have occurred on the evolutionary path between two chromosomes.

The complete description of this problem can be found at [http://rosalind.info/problems/rear/](http://rosalind.info/problems/rear/).

Write an APL function named `reversals` which:

- takes an integer vector left right argument representing $\pi$ as described on Rosalind.info
- takes an integer vector left argument representing $\sigma$ as described on Rosalind.info
- returns a 2-element vector with elements of

  [1] an integer representing the reversal distance between $\pi$ and $\sigma$

  [2] a vector of 2-element integer vectors representing the collection of reversals sorting $\pi$ into $\sigma$

Example:

```
      1 2 3 4 5 6 7 8 9 10 reversals 1 8 9 3 2 7 6 5 4 10
┌─┬─────────┐
│2│┌───┬───┐│
│ ││4 9│2 5││
│ │└───┴───┘│
└─┴─────────┘
```

Example Explained:

To perform the conversion, two reversals are necessary – the first from index positions 4-9, and the second from index positions 2-5:

```
1 2 3 4 5 6 7 8 9 10
      └─────┬─────┘
         reverse
      ┌─────┴─────┐
1 2 3 9 8 7 6 5 4 10
  └──┬──┘
  reverse
  ┌──┴──┐
1 8 9 3 2 7 6 5 4 10
```