NATIONAL UNIVERSITY OF SINGAPORE

CS1101S — PROGRAMMING METHODOLOGY

(AY2021/2022 SEMESTER 1)

FINAL ASSESSMENT

Time Allowed: **2 Hours**

INSTRUCTIONS

1. This assessment contains **22 Questions** in **9 Sections**.

2. The full score of this assessment is **100 marks**.

3. Answer **all questions**.

4. This is a **Closed-Book** assessment, but you are allowed one double-sided **A4 / foolscap / letter-sized sheet** of handwritten or printed **notes**.

5. You are allowed to use up to **4 sheets** of **blank A4 / foolscap / letter-sized** paper as **scratch paper**.

6. Where programs are required, write them in the **Source §4** language. You are allowed access to these online reference pages:

   - **Source §4 pre-declared constants and functions** at https://docs.sourceacademy.org/source_4/global.html
   - **Specification of Source §4** at https://docs.sourceacademy.org/source_4.pdf

7. In any question, unless it is specifically allowed, your answer **must not use functions given in, or written by you for, other questions**.

8. **Follow the instructions of your invigilator or the module coordinator to submit your answers**.

Section A: Orders of Growth [12 marks]

What is the **order of growth** of the **running time** for computing the result of applying the following functions to a positive integer N, using the Θ notation?

(1) [2 marks]

```
function fun(N) {
    let acc = 0;
    for (let i = 1; i <= N; i = i + 10) {
        for (let k = 1; k <= 999999; k = k + 1) {
            acc = acc + 1;
        }
    }
    return acc;
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(2) [2 marks]

```
function fun(N) {
    let acc = 0;
    for (let i = 1; i <= N; i = i + 10) { acc = acc + 1; }
    for (let k = 1; k <= 999999 * N; k = k + 10) { acc = acc + 1; }
    return acc;
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(3) [2 marks]

```
function fun(N) {
    let acc = 0;
    for (let i = 1; i <= N * N; i = i * 2) { acc = acc + 1; }
    return acc;
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(4) [3 marks]

```
function fun(N) {
    let acc = 0;
    for (let i = 1; i <= N; i = i + 10) {
        for (let k = i; k >= 1; k = k - 10) { acc = acc + 1; }
    }
    return acc;
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

(5) [3 marks]

```
function fun(N) {
    if (N <= 1) {
        return 1;
    } else {
        let acc = 0;
        for (let i = 1; i <= N; i = i + 1) { acc = acc + 1; }
        acc = acc + fun(N / 2);
        acc = acc + fun(N / 2);
        return acc;
    }
}
```

A. $\Theta(1)$
B. $\Theta(\log N)$
C. $\Theta(N)$
D. $\Theta(N \log N)$
E. $\Theta(N^2)$
F. $\Theta(N^2 \log N)$
G. $\Theta(N^3)$
H. $\Theta(2^N)$

Section B: List Processing [14 marks]

(6) [5 marks]

Complete the following function `remove_elem`, which takes as arguments a non-empty list `L`, a non-negative integer `pos`, and returns a list that is the result of removing from `L` the element at position `pos`. The value of `pos` ranges from 0 to `length(L) - 1`. Your function must not modify the input list `L`.

```
function remove_elem(L, pos) {
    return /* YOUR SOLUTION */
}
```

Example:

```
const L = list(10, 11, 12, 13);
const R = remove_elem(L, 2);
R;  // equals list(10, 11, 13)
L;  // equals list(10, 11, 12, 13)
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(7) [5 marks]

Complete the following function `d_remove_elem`, which takes as arguments a non-empty list `L`, a non-negative integer `pos`, and returns a list that is the result of removing from `L` the element at position `pos`. The value of `pos` ranges from 0 to `length(L) - 1`. Your function **must not create any new pair**, and the pairs in the result list must be from the existing pairs in the input list `L`. Your function **must not use the `set_head` function**.

```
function d_remove_elem(L, pos) {
    if (pos === 0) {
        return tail(L);
    } else {
        /* YOUR SOLUTION */
    }
}
```

Example:

```
const L = list(10, 11, 12, 13);
let R = d_remove_elem(L, 2);
R;  // equals list(10, 11, 13)
L;  // equals list(10, 11, 13)

R = d_remove_elem(L, 0);
R;  // equals list(11, 13)
L;  // equals list(10, 11, 13)
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(8) [4 marks]

Complete the following program so that at the end of the program evaluation, the value of `L` is equal to `list(9, 3, 4)`.

```
let L = list(3, 9, 4);
const p = tail(L);
const q = tail(p);

/* YOUR SOLUTION */

L;  // equals list(9, 3, 4)
```

Choose the correct sequence of statements from the following for the part that is marked ***/* YOUR SOLUTION */***.

A. `set_tail(p, L); set_tail(L, q);`

B. `set_tail(p, L); set_tail(L, q); L = q;`

C. `set_tail(L, p);  set_tail(q, L);`

D. `set_tail(L, p);  set_tail(q, L);  L = p;`

E. `set_tail(L, q);  set_tail(p, L);`

F. `set_tail(L, q);  set_tail(p, L);  L = p;`

G None of the other options is the correct answer.

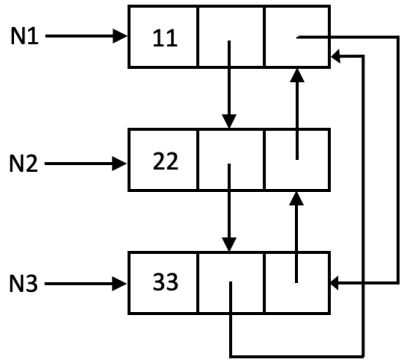
Section C: Double Rings [12 marks]

We define a ***double ring*** as a circular doubly-linked list of numbers: Each ***node*** is a 3-element array. Position 0 in the array contains the data (here numbers). Position 1 contains a pointer to the *next* node, and Position 2 contains a pointer to the *previous* node.

```
const next = 1;
const prev = 2;
```

Example 1:

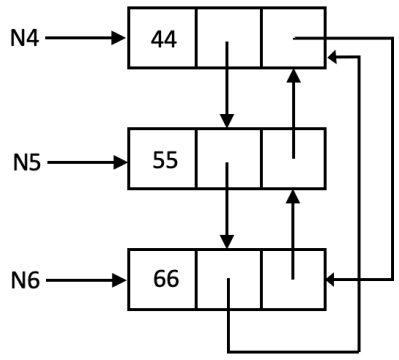
```
const N1 = [11, undefined, undefined];
const N2 = [22, undefined, undefined];
const N3 = [33, undefined, undefined];
N1[next] = N2;  N2[next] = N3;  N3[next] = N1;
N1[prev] = N3;  N2[prev] = N1;  N3[prev] = N2;
```



Note: `N1`, `N2` and `N3` are all double rings containing the numbers 11, 22, and 33 (in different order).

Example 2:

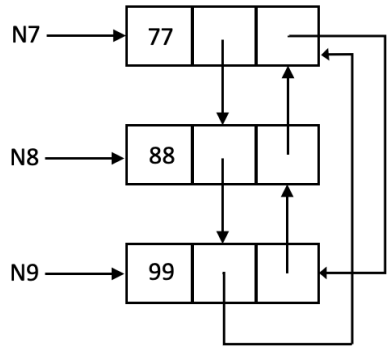
```
const N4 = [44, undefined, undefined];
const N5 = [55, undefined, undefined];
const N6 = [66, undefined, undefined];
N4[next] = N5;  N5[next] = N6;  N6[next] = N4;
N4[prev] = N6;  N5[prev] = N4;  N6[prev] = N5;
```



Note: `N4`, `N5` and `N6` are all double rings containing the numbers 44, 55, and 66 (in different order).

Example 3:

```
const N7 = [77, undefined, undefined];
const N8 = [88, undefined, undefined];
const N9 = [99, undefined, undefined];
N7[next] = N8;  N8[next] = N9;  N9[next] = N7;
N7[prev] = N9;  N8[prev] = N7;  N9[prev] = N8;
```



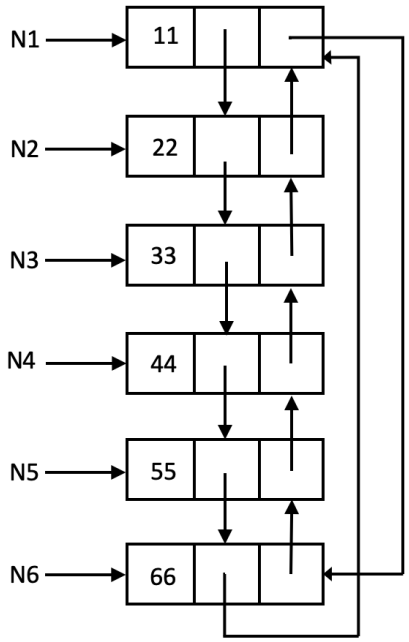
Note: `N7`, `N8` and `N9` are all double rings containing the numbers 77, 88, and 99 (in different order).

(9) [6 marks]

In this question, we program a function `adjoin_rings` that takes two double rings as arguments and destructively changes them. The function `adjoin_rings` must return `undefined`. It changes the double rings such that they both contain all numbers of the given double rings. The runtime of the function `adjoin_rings` must have an **order of growth of *O*(1)**. You can assume that each of the given double rings has at least one node, and the given double rings have no nodes in common.

Example:

The application of `adjoin_rings(N1, N4)` leads to the following structure, where `N1` and `N4` are double rings shown in Examples 1–3.



Complete the following declaration so that the function `adjoin_rings` meets the specification above.

```
function adjoin_rings(n1, n2) {
    const n1_last = n1[prev];
    const n2_last = n2[prev];

    /* YOUR SOLUTION */

}
```

Choose the correct sequence of statements from the following for the part that is marked ***/* YOUR SOLUTION */***.

A.

```
n1[prev] = n2_last;
n1_last[next] = n2;
n2[prev] = n1_last;
n2_last[next] = n1;
```

B.

```
n1[prev] = n2;
n1_last[next] = n2_last;
n2[prev] = n1;
n2_last[next] = n1_last;
```

C.

```
n1[prev] = n1_last;
n1_last[next] = n1;
n2[prev] = n2_last;
n2_last[next] = n2;
```

D.

```
for (let n = n1; n !== n1_last; n = n[next]) {
    n[next] = n[next][prev];
}
n2[prev] = n1;
n2_last[next] = n1_last;
```

E.

```
n1[prev] = n2_last;
n1_last[next] = n2;
for (let n = n2; n !== n2_last; n = n[next]) {
    n[next] = n[next][prev];
}
```

F.

```
for (let n = n1; n !== n1_last; n = n[next]) {
    n[next] = n[next][prev];
}
for (let n = n2; n !== n2_last; n = n[next]) {
    n[next] = n[next][prev];
}
```

G None of the other options is the correct answer.

(10) [6 marks]

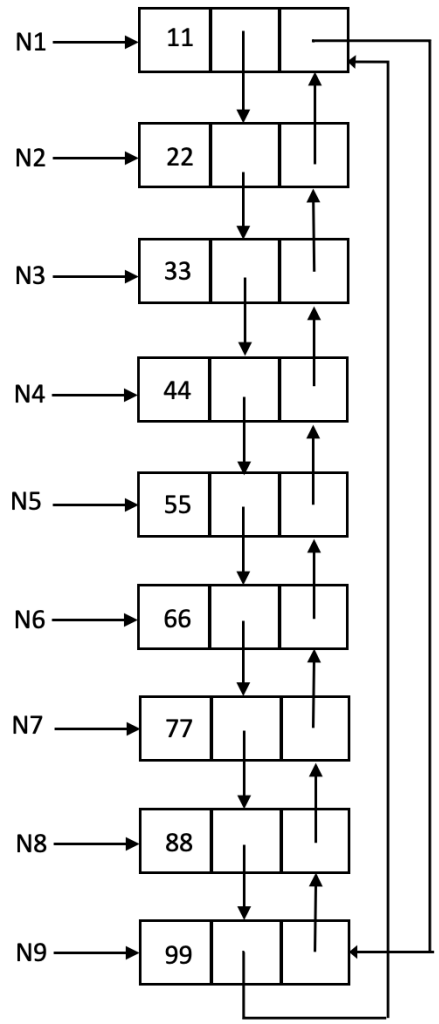
Complete the following function `adjoin_all_rings` that takes a non-empty list of double rings as argument and changes them such that they all contain all numbers of the given double rings. The function `adjoin_all_rings` must return `undefined`.

Your solution **must use the function `accumulate`** to adjoin the double rings. Your solution **must also use the function `adjoin_rings`** from the preceding question. You can assume that each of the given double rings has at least one node, and the given double rings have no nodes in common.

```
function adjoin_all_rings(ns) {
    accumulate( /* YOUR SOLUTION */ );
}
```

Example:

The application of `adjoin_all_rings(list(N1, N4, N7))` leads to the following structure, where `N1`, `N4` and `N7` are double rings shown in Examples 1–3.



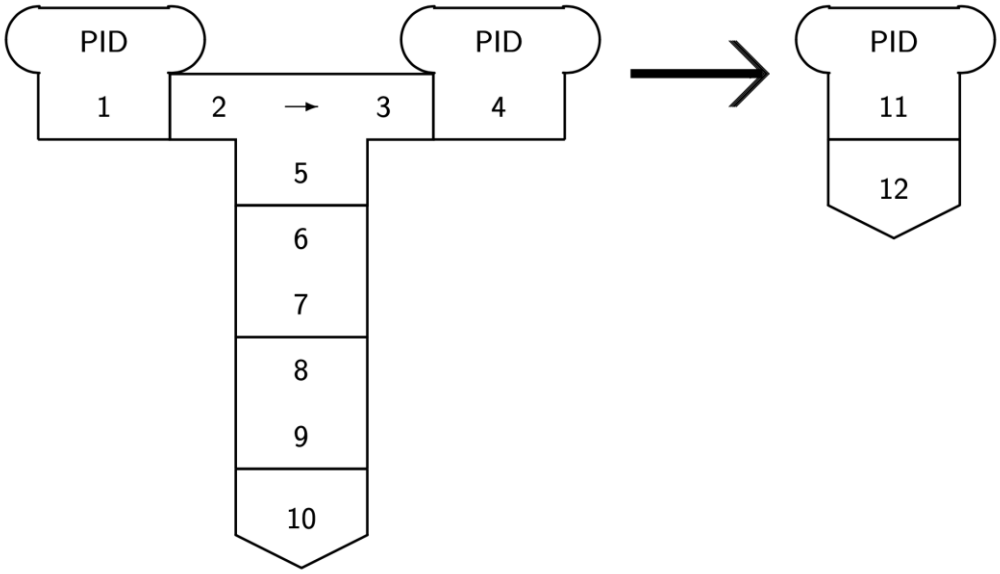
In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

Section D: The Future of PID [6 marks]

(11) [6 marks]

In a future version of the Source Academy, the Avengers are providing an interpreter for **TypeScript** in the language **JavaScript**. To program robots using the Source Academy, students are given a compiler from **Source** to **ARM8** machine code, written in **TypeScript**. Students have a browser in **x86** code and a PC with an **x86** processor. The browser can interpret programs written in **JavaScript**. The students are given a robot with an **ARM8** computer on board. The robot can be programmed via internet to run any **ARM8** program.

The following T-diagrams are supposed to depict how students can write a PID program in **Source** in the browser and run them on the robot, but the diagram is labelled with numbers instead of the actual languages.



Complete the above T-diagrams, by writing in the following spaces, the actual language for each number label, which should be either **ARM8**, **JavaScript**, **Source**, **TypeScript**, or **x86**. Make sure you **spell each word exactly** as shown in the above list.

1.		**7.**	
2.		**8.**	
3.		**9.**	
4.		**10.**	
5.		**11.**	
6.		**12.**	

Section E: Sequences of Function Applications [16 marks]

(12) [5 marks]

The following statements show example applications of the unary function `until_zero`:

```
until_zero(0);  // returns "Hello"
until_zero(82)(0);  // returns "Hello"
until_zero(1001)(339)(0);  // returns "Hello"
until_zero(56)(12)(3)(77)(67)(12)(33)(8)(38)(1)(0);  // returns "Hello"
```

An admissible use of `until_zero` is a sequence of one or more function applications as shown in these examples. The argument of the last function application is 0, and all arguments before it (if there are any) are not 0. The result of evaluating an admissible use of `until_zero` is the string `"Hello"`.

Complete the function `until_zero`, which takes a single argument `x` and produces the behavior described above.

```
function until_zero(x) {
    return /* YOUR SOLUTION */
}
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(13) [5 marks]

The following statements show example applications of the unary function `sumsum`:

```
sumsum(0);  // returns 0
sumsum(78)(0);  // returns 78
sumsum(2)(3)(1)(5)(4)(100)(0);  // returns 115
sumsum(1)(2)(1)(2)(1)(2)(1)(2)(1)(2)(1)(2)(1)(2)(0);  // returns 21
```

An admissible use of `sumsum` is a sequence of one or more function applications as shown in these examples. The argument of the last function application is 0, and all arguments before it (if there are any) are numbers that are not 0. The result of evaluating an admissible use of `sumsum` is the sum of all the arguments.

Complete the function `sumsum`, which takes a number `x` as argument and produces the behavior described above.

```
function sumsum(x) {
    /* YOUR SOLUTION */
}
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(14) [6 marks]

Complete the following function `sum_list`, which takes as argument a list of numbers, and returns the sum of all the numbers in the input list. Your function **must use the function `sumsum` from the preceding question to perform the additions**, and it **must also use the function `accumulate`** in a meaningful way. Your function must not modify the input list.

```
function sum_list(L) {
    return /* YOUR SOLUTION */
}
```

Example:

```
const L = list(10, 0, 20, 0, 0, 30, 40);
sum_list(L);  // returns 100
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

Section F: Submatrix Sums [16 marks]

We represent a ***matrix*** as a "2D array" of numbers (which is actually an array of arrays of numbers in Source). For example, the 3 x 4 matrix

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

is represented in Source as

```
[[1,  2,  3,  4],
 [5,  6,  7,  8],
 [9, 10, 11, 12]]
```

We will refer to the top row of the matrix as Row 0, and the leftmost column as Column 0.

A ***submatrix*** of a matrix **M** is a matrix consisting of only the elements within a rectangular region of **M**. In the following example matrix **M**, a 2 x 2 submatrix of **M** is shown in the shaded rectangular region.

$$\mathbf{M} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & \mathbf{6} & \mathbf{7} & 8 \\ 9 & \mathbf{10} & \mathbf{11} & 12 \end{bmatrix}$$

The ***submatrix sum*** of a submatrix is the sum of all elements within the submatrix.

(15) [5 marks]

Complete the following function `submatrix_sum`. An application `submatrix_sum(M, min_row, min_col, max_row, max_col)` to an R x C matrix `M`, and a rectangular region given by `min_row`, `min_col`, `max_row` and `max_col`, returns the submatrix sum of `M` within the rectangular region. `min_row` and `min_col` are the row and column of the top-leftmost element of the submatrix, and `max_row` and `max_col` the bottom-rightmost element. You can assume $0 \leq$ `min_row` $\leq$ `max_row` $< R$ and $0 \leq$ `min_col` $\leq$ `max_col` $< C$. Your function must not modify the input matrix `M`.

```
function submatrix_sum(M, min_row, min_col, max_row, max_col) {
    /* YOUR SOLUTION */
}
```

Example:

```
const M = [[1, 2, 3, 4],
           [2, 3, 4, 5],
           [3, 4, 5, 6]];

submatrix_sum(M, 1, 2, 1, 2));  // returns 4
submatrix_sum(M, 0, 0, 2, 3));  // returns 42
submatrix_sum(M, 1, 1, 2, 2));  // returns 16
submatrix_sum(M, 0, 1, 2, 2));  // returns 21
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(16) [6 marks]

There may be many submatrix sums we want to find for a given R x C matrix **M**, so we want to compute each submatrix sum as efficiently as possible. It is actually possible to compute each submatrix sum in $O(1)$ time (constant time), regardless of the submatrix size. This requires the matrix **M** to be pre-processed into a ***sum area table***, **S**, first. The sum area table **S** is also an R x C matrix, and its elements are derived from **M** as follows:

$$\mathbf{S}[row][col] = \sum_{r=0}^{row} \sum_{c=0}^{col} \mathbf{M}[r][c]$$

$$\text{for } 0 \leq row < R \text{ and } 0 \leq col < C.$$

Complete the following function `make_sum_area_table` that takes as argument an R x C matrix `M`, and returns the sum area table of `M`. Your function must not modify the input matrix `M`.

Your function **must compute the sum area table in *O*(*RC*) time**. If your function takes worse than $O(RC)$ time, you get **at most 2 marks** for this question.

```
function make_sum_area_table(M) {
    const ROWS = array_length(M);
    const COLS = array_length(M[0]);
    const S = []; // the sum area table

    for (let r = 0; r < ROWS; r = r + 1) { S[r] = []; }

    function fill_SAT(r, c) {
        if (r < 0 || c < 0) {
            return 0;
        } else if (S[r][c] !== undefined) {
            return S[r][c];
        } else {

            /* YOUR SOLUTION */

        }
    }

    fill_SAT(ROWS - 1, COLS - 1);
    return S;
}
```

Example:

```
const M = [[1, 2, 3, 4],
           [2, 3, 4, 5],
           [3, 4, 5, 6]];

make_sum_area_table(M);
// returns [[1,  3,  6, 10],
//          [3,  8, 15, 24],
//          [6, 15, 27, 42]]
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(17) [5 marks]

With the sum area table **S** computed from the R x C matrix **M**, we can now compute any submatrix sum of **M** in $O(1)$ time.

Complete the following function `fast_submatrix_sum`. An application `fast_submatrix_sum(S, min_row, min_col, max_row, max_col)` to a sum area table `S` of matrix **M**, and a rectangular region given by `min_row`, `min_col`, `max_row` and `max_col`, returns the submatrix sum of **M** within the rectangular region. `min_row` and `min_col` are the row and column of the top-leftmost element of the submatrix, and `max_row` and `max_col` the bottom-rightmost element. You can assume $0 \leq$ `min_row` $\leq$ `max_row` $< R$ and $0 \leq$ `min_col` $\leq$ `max_col` $< C$. Your function must not modify the input sum area table `S`.

Your function **must compute each submatrix sum in *O*(1) time**. If your function takes worse than $O(1)$ time, you get **at most 2 marks** for this question.

```
function fast_submatrix_sum(S, min_row, min_col, max_row, max_col) {
    function get_SAT_elem(r, c) {
        return (r < 0 || c < 0) ? 0 : S[r][c];
    }
    /* YOUR SOLUTION */
}
```

Example:

```
const M = [[1, 2, 3, 4],
           [2, 3, 4, 5],
           [3, 4, 5, 6]];

const S = make_sum_area_table(M);
// S is now [[1,  3,  6, 10],
//           [3,  8, 15, 24],
//           [6, 15, 27, 42]]

fast_submatrix_sum(S, 1, 2, 1, 2));  // returns 4
fast_submatrix_sum(S, 0, 0, 2, 3));  // returns 42
fast_submatrix_sum(S, 1, 1, 2, 2));  // returns 16
fast_submatrix_sum(S, 0, 1, 2, 2));  // returns 21
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

Section G: Streams [10 marks]

(18) [5 marks]

The function `hold_stream` takes in a non-empty list and returns an infinite stream which contains all elements of the list in the given order and then repeats the last element of the list.

Examples:

```
hold_stream(enum_list(1, 5))
// returns a stream containing 1, 2, 3, 4, 5, 5, 5, 5, ...

hold_stream(enum_list(1, 3))
// returns a stream containing 1, 2, 3, 3, 3, 3, 3, 3, ...

hold_stream(list(2, 4, 6))
// returns a stream containing 2, 4, 6, 6, 6, 6, 6, 6, ...

hold_stream(list(1))
// returns a stream containing 1, 1, 1, 1, 1, 1, 1, 1, ...
```

Complete the following function `hold_stream`. The runtime of `hold_stream(xs)` must have an order of growth of $O(1)$.

```
function hold_stream(xs) {
    return /* YOUR SOLUTION */
}
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(19) [5 marks]

The function `search_stream` takes as arguments a stream `xs`, a non-negative integer position `pos`, and a data item `x`. The application `search_stream(xs, pos, x)` returns `true` if `x` occurs in `xs` at a position that is less than or equal to `pos`, and `false` otherwise. We specify that the data item `x` occurs in `xs` at position `q` if and only if `stream_ref(xs, q) === x` is `true`.

Examples:

```
const ones = pair(1, () => ones);
const integers = pair(1, () => stream_map(x => x + 1, integers));
const finite_stream = enum_stream(1, 5);

search_stream(ones, 0, 1);  // returns true
search_stream(ones, 0, 2);  // returns false
search_stream(integers, 4, 4);  // returns true
search_stream(integers, 4, 5);  // returns true (corrected via announcement)
search_stream(integers, 3, 5);  // returns false
search_stream(finite_stream, 6, 10);  // returns false
```

Complete the following function `search_stream`. The runtime of `search_stream(xs, pos, x)` must have an order of growth of O(`pos`), and its space requirement must have an order of growth of $O(1)$. (**Hint:** If you use `stream_ref` or `eval_stream`, you will most likely not meet all order of growth requirements.)

```
function search_stream(xs, pos, x) {
    return /* YOUR SOLUTION */
}
```

In the following space, write your solution only for the part that is marked `/* YOUR SOLUTION */`.

Section H: Memoized Streams [8 marks]

Consider the following program:

```
function memo(fun) {
    let already_run = false;
    let result = undefined;
    return () => {
                if (!already_run) {
                    result = fun();
                    already_run = true;
                    return result;
                } else {
                    return result;
                }
            };
}

function add_streams(s1, s2) {
    return pair(head(s1) + head(s2),
                () => add_streams(stream_tail(s1),
                                  stream_tail(s2)));
}

function partial_sums_1(s) {
    return pair(head(s),
                () => add_streams(stream_tail(s),
                                  partial_sums_1(s)));
}

function partial_sums_2(s) {
    return pair(head(s),
                memo(() => add_streams(stream_tail(s),
                                       partial_sums_2(s))));
}

const ones = pair(1, () => ones);

const integers_1 = partial_sums_1(ones);

const integers_2 = partial_sums_2(ones);
```

(20) [4 marks]

Let $f(n)$ be the number of additions performed when evaluating the function application `eval_stream(integers_1, n)`. What is the order of growth of $f(n)$ in Θ notation?

A. $\Theta(1)$
B. $\Theta(\log n)$
C. $\Theta(n)$
D. $\Theta(n \log n)$
E. $\Theta(n^2)$
F. $\Theta(n^2 \log n)$
G. $\Theta(n^3)$
H. $\Theta(c^n)$, where c is a constant greater than 1

(21) [4 marks]

Let $g(n)$ be the number of additions performed when evaluating the function application `eval_stream(integers_2, n)`. What is the order of growth of $g(n)$ in Θ notation?

A. $\Theta(1)$
B. $\Theta(\log n)$
C. $\Theta(n)$
D. $\Theta(n \log n)$
E. $\Theta(n^2)$
F. $\Theta(n^2 \log n)$
G. $\Theta(n^3)$
H. $\Theta(c^n)$, where c is a constant greater than 1

Section I: Fast and Furious [6 marks]

(22) [6 marks]

The following function was given in the lectures and in the textbook:

```
const square = x => x * x;
const is_even = n => n % 2 === 0;

function fast_expt(b, n) {
    return n === 0
           ? 1
           : is_even(n)
           ? square(fast_expt(b, n / 2))
           : b * fast_expt(b, n - 1);
}
```

An application `fast_expt(b, n)`, where `b` is a number and `n` a non-negative integer, computes b^n, in words: `b` to the power of `n`. The number of multiplications performed during the evaluation of `fast_expt(b, n)` has an order of growth of $\Theta(\log n)$, and `fast_expt` gives rise to a **recursive process**.

Complete the following function `fast_and_furious_expt`, such that

- `fast_and_furious_expt(b, n)` computes b^n, and
- the number of multiplications performed by `fast_and_furious_expt(b, n)` has an order of growth of $\Theta(\log n)$, and
- the function `fast_and_furious_expt` gives rise to an **iterative process**.

```
const square = x => x * x;
const is_even = n => n % 2 === 0;

function fast_and_furious_expt(b, n) {
    function ffe(bb, nn, cont) {
        /* YOUR SOLUTION */
    }
    return ffe(b, n, x => x);
}
```

In the following space, write your solution only for the part that is marked ***/* YOUR SOLUTION */***.

(blank page)

——— END OF PAPER ———