CS4050 Design and Analysis of Algorithms

Assignment Problem Set

1_1)Problem Statement

Alice and Bob are playing a game. Alice rolls a identical b-sided dice. Bob rolls c identical d-sided dice. The sides of an n-sided die have numbers 1 through n written on them. A player's score is the sum of the numbers they rolled on their dice. The player with a strictly higher score wins. It is possible that neither player wins. You are given the ints a, b, c, and d. The players already rolled their dice. If it's not possible for Alice to win, return -1. Otherwise, assume that you don't know what numbers Alice and Bob rolled, but that you know that Alice won the game. Return the expected value of Alice's score (given the above assumption).

Definition

Class:	FixedDiceGameDiv1
Method:	getExpectation
Parameters:	int, int, int, int
Returns:	double
Method signature:	double getExpectation(int a, int b, int c, int d)

(be sure your method is public)

Notes

Your return value must have an absolute or relative error smaller than 1e-3.

Constraints

a, b, c, d will each be between 1 and 50, inclusive.

Examples

```
1)1
  2
  1
  5
Returns: 2.0
```

The only way Alice can win is if she rolls a 2. Thus, if we know Alice wins, we know she rolled a 2.

```
2)3
  1
  1
  3
Returns: 3.0
```

Alice will always roll a 3.

```
3)1
  5
1
1
Returns: 3.4999999999999996
```

Alice will not win if she rolls a 1. Thus, if we know she wins, her expected score is (2+3+4+5)/4=7/2.

```
4)2
  6
  50
  30
Returns: -1.0
```

No matter what Alice rolls, she will lose.

```
5)50
  11
  50
  50
Returns: 369.8865999182022
```

1_2)Problem Statement

In the Republic of Nlogonia there are N cities. For convenience, the cities are numbered 0 through N-1. For each two different cities i and j, there is a direct one-way road from i to j. You are given the lengths of those roads as a String[] **dist** with N elements, each with N characters. For each i and j, the character **dist**[i][j] represents the length of the road from i to j. The lengths of roads are integers between 1 and 9, inclusive, and they are represented by digits '1' through '9'. That is, for distinct i and j, **dist**[i][j] will be between '1' and '9'. For each i, **dist**[i][i] will be '0'. Note that the roads from i to j and from j to i may have different lengths. Every year on Algorithms Day (the most important holiday in Nlogonia) people travel between the cities. More precisely, for each pair of distinct cities i and j, one full bus of people travels from i to j. Each of those buses drives along a shortest path from its origin to its destination. If there are multiple shortest paths, the bus driver picks one of them arbitrarily. The roads in Nlogonia are currently limited. You are given an int **T** with the following meaning: each of the current roads is only safe if it is *guaranteed* that there will be *strictly fewer* than **T** buses driving along the road. In other words, a road is *unsafe* if it is possible that **T** or more buses will use it. The government wants to rebuild all unsafe roads before the next Algorithms Day. Return the sum of lengths of all unsafe roads.

Definition

Class:	BuildingRoutes
Method:	build
Parameters:	String[], int
Returns:	int

Method signature: int build(String[] dist, int T)
(be sure your method is public)

Constraints

- N will be between 2 and 50, inclusive.
- **dist** will contain exactly N elements.
- Each element of **dist** will contain exactly N characters.
- For each i, **dist**[i][i] will be '0'.
- For each pair of distinct i and j, **dist**[i][j] will be between '1' and '9', inclusive.
- **T** will be between 1 and 2500, inclusive.

Examples

```
1){"011",
   "101",
   "110"}
```

```
1
Returns: 6
```

As **T**=1, a road is unsafe as soon as it is possible that a bus will use it. Each of the six roads in this test case belongs to some shortest path, hence each of them is unsafe

```
2)
{"033",
 "309",
 "390"}
1
Returns: 12
```

The roads 1->2 and 2->1 (the two roads of length 9) will not be used by any bus. Only the four remaining roads are unsafe in this case.

```
3)
{"0123",
 "1023",
 "1203",
 "1230"}
2
Returns: 5
```

```
4)
{"05789654",
 "10347583",
 "65085479",
 "55602398",
 "76590934",
 "57939045",
 "12345608",
 "68647640"}
3
Returns: 40
```

1_3)Problem Statement

You have 16 bricks. Each brick has the shape of a rectangular box. You are given a int[] **height**. For each i, one of your bricks has dimensions 1 x 1 x **height**[i].You also have an opaque table. You are going to place your 16 bricks onto the table in a specific way. You are not allowed to rotate the bricks while placing them: the dimension given in **height** must always be vertical. On the table, there is a 4 x 4 grid of squares. You have to place exactly one of your bricks onto each of the squares.After you place all the bricks, we will look at the solid formed by them. We are interested in the visible surface area of the solid. Note that the bottom sides of your bricks are not a part of the visible surface, as they stand on the table. Also, adjacent bricks always touch each other and the parts where they touch are not visible.Different arrangements of bricks may lead to different visible surfaces. Return the largest possible visible surface area.

Definition

Class:	SixteenBricks
Method:	maximumSurface
Parameters:	int[]
Returns:	int

Method signature: int maximumSurface(int[] height)
(be sure your method is public)

Constraints

- **height** will contain exactly 16 elements.
- Each element of **height** will be between 1 and 100, inclusive.

Examples

```
1)
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}
Returns: 32
```

All your bricks look the same. The only solid you can construct is a 1 x 4 x 4 box. The bottom side of the box is not visible, the other five sides are. The total visible surface area is 4*4 + 4*(1*4) = 32.

```
2)
{1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2}
Returns: 64
```

In order to maximize the visible surface area, you should produce a configuration in which no two bricks with height 2 share a common face.

3) `{77, 78, 58, 34, 30, 20, 8, 71, 37, 74, 21, 45, 39, 16, 4, 59}`

```
Returns: 1798
```

1_4)Problem Statement

Shiny likes to play games. Her favorite games are games with pebbles (small stones). Today, she is playing such a game with her friend Lucy. Initially, there are N piles of stones. You are given a int[] **number** with N elements. Each element of **number** is the number of stones in one of the piles. The players take alternating turns. Shiny plays first.

In each turn, the current player must:

1. Choose a pile X that has at least two stones.
2. Split the chosen pile X into two non-empty parts A and B. (The parts can have arbitrary sizes, as long as both are non-empty.)
3. Choose two piles Y and Z. (Y and Z must be different non-empty piles other than X.)
4. Add all stones from A to the pile Y, and all stones from B to the pile Z.

For example, if the current piles are {1, 2, 50}, the current player could:

1. Choose the pile with 50 stones as X.
2. Split it into two parts with 25 stones each.
3. Choose the other two piles (the ones with 1 and 2 stones) to be Y and Z.
4. Add all stones from A to the pile Y, and all stones from B to the pile Z. At the end of the turn, there are two piles of stones: one with 26 and one with 27 stones.

The player who cannot make a valid move loses the game. Assume that both players play the game optimally. Return the String "WIN" (quotes for clarity) if Shiny wins the game, and "LOSE" if she does not.

Definition

Class:	SplitStoneGame
Method:	winOrLose
Parameters:	int[]
Returns:	String
Method signature:	String winOrLose(int[] number)

(be sure your method is public)

Constraints

- **number** will contain between 1 and 50 elements, inclusive.

- Each element of **number** will be between 1 and 50, inclusive.

Examples

```
1)
{1, 1, 1}
Returns: "LOSE"
```

Shiny can't choose a pile X that has at least two stones, so she loses.

```
2)
{2, 2}
Returns: "LOSE"
```

After Shiny chooses one of the piles as X and splits it into two piles with one stone each, she is unable to choose Y and Z, because there is only one pile left to choose from at the moment. Thus, she cannot make a valid move and therefore she loses the game.

```
3)
{1, 1, 2}
Returns: "WIN"
```

Shiny can choose the last pile as X, split it into 1+1 stone, and add those stones to the other two piles. This is a valid move that produces two piles with two stones each, and it is now Lucy's turn. As we saw in Example 1, Lucy now has no valid move left, thus she loses the game and Shiny is the winner.

```
4)
{1, 2, 3, 4, 3, 2, 2, 4, 3, 1, 4, 4, 1, 2, 4, 4, 1, 4, 3, 1,
4, 2, 1}
Returns: "WIN"
```

```
5)
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 3, 1, 9, 1, 3, 1,
1, 1, 1, 1}
Returns: "LOSE"
```

1_5)Problem Statement

Marco likes strings. In particular, he likes strings that have a lot of palindromic substrings. For example, he really likes the string "aaa" because it has 6 palindromic substrings: "a" occurs three times, "aa" occurs twice, and "aaa" occurs once.Right now, Marco has a string `S` composed of lowercase letters and question marks. You are to reconstruct `S` from the given String[]s **S1** and **S2** as follows:

1. Concatenate all elements of **S1** to make a string `A`.
2. Concatenate all elements of **S2** to make a string `B`.
3. Finally, concatenate `A` and `B` to get `S`.

Marco is going to replace every question mark in `S` with a random lowercase letter (`'a' - 'z'`). Return the expected number of palindromic substrings in the resulting string.

Definition

Class:	PalindromicSubstringsDiv1
Method:	expectedPalindromes
Parameters:	String[], String[]
Returns:	double
Method signature:	double expectedPalindromes(String[] S1, String[] S2)

(be sure your method is public)

Notes

- For each question mark, the letter used to replace it is chosen uniformly at random. That is, the probability of choosing any particular letter is 1/26. All random choices are mutually independent.
- A palindromic string is a string that reads the same forwards and backwards.
- Your return value must have an absolute or a relative error of less than 1e-9.

Constraints

- **S1** and **S2** will contain no more than 50 elements.
- Each element of **S1** and **S2** will contain no more than 50 characters.
- `S` will contain at least 1 character.

- `S` will contain only lowercase letters (`'a'` - `'z'`) and question marks (`'?'`).

Examples

1)

```
{"a","a",""}
{"a"}
Returns: 6.0
```

This is the example given in the statement.

2)

```
{"z??"}
{}
Returns: 3.115384615384615
```

There are 26^2 = 676 equally likely possibilities for the letters used to replace the question marks. Here are all possible outcomes:

- The string "zzz" has 6 palindromic substrings.
- Each of the 25 strings "zaz", "zbz", ..., "zyz" has 4 palindromic substrings.
- Each of the 25 strings "zza", "zzb", ..., "zzy" has 4 palindromic substrings.
- Each of the 25 strings "zaa", "zbb", ..., "zyy" has 4 palindromic substrings.
- Each of the remaining 600 possible strings only has the 3 single-letter palindromic substrings.

The expected number of palindromic substrings can be computed simply as the average over all 676 possible cases. Hence, the correct return value is (6 + 75*4 + 600*3) / 676.

3)

```
{"ab","c"}
{"??","a?"}
Returns: 7.315088757396449
```

4)

```
{}
{"?"}
Returns: 1.0
```

5)

```
{"ab?def","?"}
{"f??a"}
Returns: 12.545971779699588
```

1_6) Problem Statement

You have an array with N elements. Initially, each element is 0. You can perform the following operations:

- *Increment operation*: Choose one element of the array and increment the value by one.
- *Doubling operation*: Double the value of each element.

You are given a int[] **desiredArray** containing N elements. Compute and return the smallest possible number of operations needed to change the array from all zeros to **desiredArray**.

Definition

Class:	IncrementAndDoubling
Method:	getMin
Parameters:	int[]
Returns:	int

Method signature: int getMin(int[] desiredArray)
(be sure your method is public)

Constraints

- **desiredArray** will contain between 1 and 50 elements, inclusive.
- Each element of **desiredArray** will be between 0 and 1,000, inclusive.

Examples

1)

```
{2, 1}
Returns: 3
```

One of the optimal solutions is to apply increment operations to element 0 twice and then to element 1 once. Total number of operations is 3.

```
2) {16, 16, 16}
Returns: 7
```

The optimum solution looks as follows. First, apply an increment operation to each element. Then apply the doubling operation four times. Total number of operations is 3+4=7.

3)

```
{100}
Returns: 9
```

4)

```
{0, 0, 1, 0, 1}
Returns: 2
```

Some elements in **desiredArray** may be zeros.

5)

```
{123, 234, 345, 456, 567, 789}
```

```
Returns: 40
```

6)

```
{7,5,8,1,8,6,6,5,3,5,5,2,8,9,9,4,6,9,4,4,1,9,9,2,8,4,7,4,8,8,6,3,9,4,
3,4,5,1,9,8,3,8,3,7,9,3,8,4,4,7}
```

```
Returns: 84
```

1_7)Problem Statement

You have N balls, where N is odd. The balls are numbered from 0 to N-1. In that order, they are arranged into a row going from the left to the right. In addition to the number, each ball has either the word "left" or the word "right" written on it. For simplicity, we will use the character '<' instead of "left", and the character '>' instead of "right". You are given the labels on all balls as the String **label**. For each i, character i of **label** represents the word on ball i.

You will now repeat the following procedure:

1. Choose a ball that is not at either end of the row of balls.
2. If the chosen ball has the label '<', remove the chosen ball and also the ball immediately to the left of it. Otherwise, remove the chosen ball and also the ball to the right of it.
3. Without reordering the remaining balls, push them together to get rid of the gap created in the previous step.

The process ends when only one ball remains in the row. That ball is called the survivor. Note that the numbers on the balls do not change during the process.Find all possible survivors. Your method must return a String containing exactly N characters. If ball i can be the survivor, character i of the return value must be 'o' (lowercase oh). Otherwise, the corresponding character must be '.' (a period).

Definition

Class:	BallRemoval
Method:	canLeave
Parameters:	String
Returns:	String
Method signature:	String canLeave(String label)

(be sure your method is public)

Constraints

- **label** will contain between 3 and 49 characters, inclusive.
- **label** will contain an odd number of characters.
- Each character of **label** will be either '>' or '<'.

Examples

1)

```
"<<>"
```

```
Returns: "..o"
```

Initially, you have three balls. Since you cannot choose balls at the ends of the row, you have to choose ball 1. As its label is '<', you remove balls 0 and 1.

Hence the only possible survivor is ball 2.

2)

```
">>><<"
Returns: "o...o"
```

If you choose ball 2 or ball 3 first, you have to choose ball 1 next, and the survivor will be ball 0. If you choose ball 1 first, you have to choose ball 3 next, and the survivor will be ball 4.

3)

```
"<<><<"
Returns: "....o"
```

4)

```
"<><<><>"
Returns: "o.....o"
```

5)

```
">>><<<>>>>><<<>"
Returns: "o.....o.o.....o"
```

1_8)Problem Statement

For a given string S of length n an inversion is a pair of integers (i, j) such that 0 <= i < j <= n-1 and S[i] > S[j]. (That is, the character at 0-based index i is greater than the character at 0-based index j.) For example, the string "abcab" has 3 inversions: (1, 3), (2, 3), and (2, 4).
Given are ints **n** and **minInv**, and a String **minStr**. We will consider all strings that are permutations of the first **n** lowercase English letters. That is, these strings have length **n** and contain each of the first **n** letters exactly once. Out of these strings, return the lexicographically smallest string R with the following two properties:

- The number of inversions in R is at least **minInv**.
- The string R is not lexicographically smaller than **minStr**.

If there is no such string, return an empty String instead.

Definition

Class:	StrIIRec
Method:	recovstr
Parameters:	int, int, String
Returns:	String
Method signature:	String recovstr(int n, int minInv, String minStr)

(be sure your method is public)

Notes

- A String A is lexicographically smaller than a String B if A is a prefix of B or A contains a smaller character at the first position where the Strings differ.

Constraints

- **n** will be between 1 and 20, inclusive.
- **minInv** will be between 0 and **n***(**n**-1)/2, inclusive.
- **minStr** will contain between 1 and **n** characters, inclusive.
- Each character in **minStr** will be one of the first **n** lowercase Latin letters.
- All characters in **minStr** will be unique.

Examples

1)

```
2
1
```

```
"ab"
Returns: "ba"
```

You must find the lexicographically smallest String that has at least 1 inversion and is not lexicographically smaller than "ab".

2)

```
9
1
"efcdgab"
Returns: "efcdgabhi"
```

3)

```
11
55
"debgikjfc"
Returns: "kjihgfedcba"
```

"kjihgfedcba" is the only String that has at least 55 inversions.

4)

```
15
0
"e"
Returns: "eabcdfghijklmno"
```

5)

```
9
20
"fcdebiha"
Returns: "fcdehigba"
```

1_9)Problem Statement

Elly and Kriss play a game. The game is played on a single row that consists of N cells; we will call it the board. The cells of the board are numbered 0 through N-1, from the left to the right. Each cell of the board is either empty or occupied by a single checker. The girls take alternating turns, until one of them cannot make a move. The girl who is unable to make a move loses the game.

In each move the current player selects a cell containing a checker and performs one of the following two types of moves:

- A step, in which the checker is moved one cell to the right. The step can only be made if the target cell is empty.
- A jump, in which the checker jumps three cells to the right. The jump can only be made if the target cell is empty and the cells it jumped over contain two other checkers.

Once a checker reaches the rightmost cell, it disappears immediately and no longer plays any role in the game. The initial layout of the board will be given as a String **board**. The i-th character of **board** will be '.' (a period) if the i-th cell is empty at the beginning, and it will be 'o' (lowercase letter o) if the i-th cell initially contains a checker. Assume that both girls play optimally. Return "YES" (quotes for clarity) if the first player wins the game and "NO" otherwise.

Definition

Class:	EllysCheckers
Method:	getWinner
Parameters:	String
Returns:	String
Method signature:	String getWinner(String board)

(be sure your method is public)

Notes

- If there is a checker on the rightmost cell in the beginning of the game, it disappears instantly (before the first move is made), as if it were never there.
- The rules of the game ensure that each cell contains at most one checker at any time, and that no checker can jump beyond the last cell.

Constraints

- **board** will contain between 1 and 20 characters, inclusive.
- Each character of **board** will be either '.' or 'o'.

Examples

1)

```
".o..."
Returns: "YES"
```

With only one checker it is pretty obvious who will win.

2)

```
"..o..o"
Returns: "YES"
```

Don't forget to ignore checkers on the rightmost cell.

3)

```
".o...ooo..oo.."
Returns: "NO"
```

Here one can jump the checker from cell 5 to cell 8.

4)

```
"......o.ooo.o......"
Returns: "YES"
```

5)

```
".o..o...o....o.....o"
Returns: "NO"
```

1_10)Problem Statement

Michael loves listening to music from his cell phone while travelling by train. He currently has **N** songs in his cell phone. During one trip he has the time to listen to **P** songs. So his cell phone creates a playlist of **P** (not necessarily different) songs according to the following rules:

- Each song has to be played at least once.
- At least **M** songs have to be played between any two occurrences of the same song. (This ensures that the playlist is not playing the same song too often.)

Michael wonders how many different playlists his cell phone can create. You are given the ints **N**, **M**, and **P**. Let X be the number of valid playlists. Since X can be too large, your method must compute and return the value (X modulo 1,000,000,007).

Definition

Class:	NoRepeatPlaylist
Method:	numPlaylists
Parameters:	int, int, int
Returns:	int
Method signature:	int numPlaylists(int N, int M, int P)

(be sure your method is public)

Notes

- Two playlists A and B are different if for some i between 1 and **P**, inclusive, the i-th song in A is different from the i-th song in B.

Constraints

- **N** will be between 1 and 100, inclusive.
- **M** will be between 0 and **N**, inclusive.
- **P** will be between **N** and 100, inclusive.

Examples

1)

1

0

3

Returns: 1

You have only 1 song which can be played as often as you want.

So the only valid playlist is: {song1, song1, song1}.

2)

1

1

3

Returns: 0

Now is the same scenario as in Example 0, but the song cannot be played 2 times in a row.

Thus there is no valid playlist.

3)

2

0

3

Returns: 6

Now you have 2 songs and you can play them as often as you want.

Just remember that playlists {song1, song1, song1} and {song2, song2, song2} are not valid, because each song must be played at least once.

4)

4

0

4

Returns: 24

You have time to play each song exactly once. So there are 4! possible playlists.

5)

2

1

4

Returns: 2

The only two possibilities are {song1, song2, song1, song2} and {song2, song1, song2, song1}.

6)

50

5

100

Returns: 222288991

1_11)Problem Statement

Camomile and her twin sister Romashka are playing a game. At the beginning of the game, they are given a word. Then, starting with Camomile, they take alternate turns, and on each turn, the player erases one letter from the word. That letter must be at a position greater than or equal to the position of the letter erased on the previous turn (on the first turn, the player can erase any letter). Letter positions are numbered consecutively from left to right and are renumbered from scratch after each turn. For example, if the word is "topcoder" and a player erases the letter 'c', the word would become "topoder", and on the next turn, the other player could only erase the letters 'r', 'e', 'd' or the second 'o'. When a player erases the last letter, the game ends. If the word at the end of the game is lexicographically greater than the word at the beginning, Camomile wins. Otherwise, Romashka wins. You are given a String **word**, which is the word given at the beginning of the game. Assuming that Romashka and Camomile both play optimally, return "Romashka" if Romashka will win or "Camomile" if Camomile will win (all quotes for clarity).

Definition

Class:	SistersErasingLetters
Method:	whoWins
Parameters:	String
Returns:	String
Method signature:	String whoWins(String word)

(be sure your method is public)

Notes

- A string X is defined as smaller than a string Y if and only if X is a prefix of Y or X has a smaller character than Y at the first position where they differ.

Constraints

- **word** will contain between 1 and 50 characters, inclusive.
- Each character in **word** will be a lowercase letter ('a'-'z').

Examples

1)

"topcoder"

Returns: "Camomile"

1. Camomile starts by erasing the letter 'c', leaving "topoder".

2. It doesn't matter what Romashka erases on her turn. The possible outcomes of this turn are "topder", "topoer", "topodr" and "topode". In the last outcome, Romashka deletes the last letter and ends the game, but loses.

3. If Romashka doesn't end the game, then Camomile will end it by erasing the last letter. The possible outcomes are "topde", "topoe" and "topod", all of which are lexicographically greater than "topcoder".

2)

"program"

Returns: "Romashka"

Camomile can't win here.

3)

"abcd"

Returns: "Camomile"

Here, Camomile can only win if she starts by erasing the letter 'a'.

4)

"abc"

Returns: "Romashka"

Note that the empty string is lexicographically smaller than any other string.

1_12)Problem Statement

Little Dazdraperma likes to travel a lot. One day she made a route in an **N**-dimensional space. In this space each point is represented by **N** coordinates. The coordinates are indexed from 1 to **N**, inclusive. She started from the origin, i.e., a point where each coordinate is 0. Then she did several moves of the following type:

- First she chose a coordinate index, i.e., a number between 1 and **N**, inclusive.
- Then she jumped to a point where the coordinate with the chosen index is either increased or decreased by 1 and all other coordinates remain the same.

Now Dazdraperma wonders whether she has ever visited the same point twice. You will be given a int[] **coords** and a String **moves** representing her route. The i-th element of **coords** is the coordinate index she has chosen during her i-th move. If the coordinate with this index was increased during the i-th move, the i-th character of **moves** will be '+', and it will be '-' if this coordinate was decreased. Return "VALID" if all points of her route were unique, including the first and the last points, and return "NOT VALID" otherwise. Two points A and B in **N**-dimensional space are different if there's an index i such that A's coordinate with index i and B's coordinate with index i are different.

Definition

Class:	RouteIntersection
Method:	isValid
Parameters:	int, int[], String
Returns:	String
Method signature:	String isValid(int N, int[] coords, String moves)

(be sure your method is public)

Constraints

- **N** will be between 1 and 1000000000 (10^9), inclusive.
- **coords** will contain between 1 and 50 elements, inclusive.
- Each element of **coords** will be between 1 and **N**, inclusive.
- **moves** will contain the same number of characters as the number of elements in **coords**.
- Each character in **moves** will be either '+' or '-'.

Examples

1)

```
1
{1}
```

"+"

Returns: "VALID"

Dazdraperma starts at (0) and then jumps to (1). The answer is "VALID".

2)

2

{1,2,1,2}

"++--"

Returns: "NOT VALID"

The route goes through 5 points: (0,0) -> (1,0) -> (1,1) -> (0,1) -> (0,0). The point (0,0) was visited twice.

3)

3

{1,2,3,1,2}

"+++--"

Returns: "VALID"

(0,0,0) -> (1,0,0) -> (1,1,0) -> (1,1,1) -> (0,1,1) -> (0,0,1).

4)

344447

{132,51717,628,344447,628,51717,344447,2}

"+-++-+--"

Returns: "NOT VALID"

The repeated point doesn't have to be the first or the last point in the route.

5)

1

{1,1}

"+-"

Returns: "NOT VALID"

6)

990630

{833196,524568,361663,108056,28026,824639,269315,440977,440977,76548,
988451,242440,948414,130873,773990,765458,130873,28026,853121,553636,
581069,82254,735536,833196,898562,898562,940783,988451,540613,317306,
623194,940783,571384,988451,108056,514374,97664}

"--+---+-+++-+-+---++-++-+---+-+--+-++"

Returns: "NOT VALID"

1_13)Problem Statement

You are given a int[] **permutation** containing a permutation of the first n positive integers (1 through n), and you want to sort them in ascending order. To do this, you will perform a series of swaps. For each swap, you consider all pairs (i, j) such that i < j and **permutation[i]** > **permutation[j]**. Among all those pairs, you choose one randomly and swap **permutation[i]** and **permutation[j]**. Each pair has the same probability of being chosen. Return the expected number of swaps needed to sort **permutation** in ascending order.

Definition

Class:	RandomSort
Method:	getExpected
Parameters:	int[]
Returns:	double

Method signature: double getExpected(int[] permutation)

(be sure your method is public)

Notes

- The returned value must be accurate to within a relative or absolute value of 1E-9.

Constraints

- **permutation** will contain between 1 and 8 elements, inclusive.
- **permutation** will contain each integer between 1 and n, inclusive, exactly once, where n is the number of elements in **permutation**.

Examples

1)

{1,3,2}

Returns: 1.0

Exactly one swap is needed.

2)

{4,3,2,1}

Returns: 4.066666666666666

In the first step, any two elements can be swapped.

3)

{1}

Returns: 0.0

This permutation is already sorted, so there's no need to perform any swaps.

4)

{2,5,1,6,3,4}

Returns: 5.666666666666666