HW 04: Pairwise and Pairwise Orthogonal Arrays

Name : Vanshaj Tyagi

Course: SSW-567

Assignment : Pairwise and Pairwise Orthogonal Arrays

Part 1: [25% of the mark]

The Software Engineering Department at Stevens Institute of Technology is launching a new website to help students plan their class schedules. We need to support the following configurations:

- Operating System: Mac iOS, Linux, and Windows 11
  - {iOS, Linux, Win11}
- Browser: Safari, Firefox, and Chrome
  - Note: Safari, Firefox, and Chrome are available on Mac OSX, Linux, and Windows 11
  - {Safari, Firefox, Chrome}
- Student Type: Undergraduate and Graduate
  - {U,G|
- Student Location: On Campus or Remote
  - {C,R}

You are required to answer the following questions:

1. What is the total number of all combinations test-cases? Show the all-combinations test-case suite.
2. Build a Pair-wise table and comment on the number of test-cases defined by your table.
3. What is the alphabet size for the given set of variables?
4. Identify the proper Orthogonal Array using S=2 and I=1 for Pair-wise OA? You can use external resources to find the required OA.
5. Populate and show the Orthogonal Array with the appropriate values for this problem.
6. Comment on the number of test-cases generated by all-combinations, Pairwise, and Pairwise-OA.

Answer Part 1 :

The Software Engineering Department at Stevens Institute of Technology is launching a new website to help students plan their class schedules. We need to support the following configurations:

Operating System: Mac iOS, Linux, and Windows 11 {iOS, Linux, Win11}

Browser: Safari, Firefox, and Chrome {Safari, Firefox, Chrome}

Student Type: Undergraduate and Graduate {U, G}

Student Location: On Campus or Remote {C, R}

Total all-combinations tests are $3 \times 3 \times 2 \times 2 = 36$, because the count equals the product of levels across factors for exhaustive testing.

All the combination are present below:

Test Case	OS	Browser	Student Type	Location
1	iOS	Safari	U	C
2	iOS	Safari	U	R
3	iOS	Safari	G	C
4	iOS	Safari	G	R
5	iOS	Firefox	U	C
6	iOS	Firefox	U	R
7	iOS	Firefox	G	C
8	iOS	Firefox	G	R
9	iOS	Chrome	U	C
10	iOS	Chrome	U	R
11	iOS	Chrome	G	C
12	iOS	Chrome	G	R
13	Linux	Safari	U	C
14	Linux	Safari	U	R
15	Linux	Safari	G	C
16	Linux	Safari	G	R
17	Linux	Firefox	U	C
18	Linux	Firefox	U	R
19	Linux	Firefox	G	C
20	Linux	Firefox	G	R
21	Linux	Chrome	U	C
22	Linux	Chrome	U	R
23	Linux	Chrome	G	C
24	Linux	Chrome	G	R
25	Win11	Safari	U	C
26	Win11	Safari	U	R
27	Win11	Safari	G	C
28	Win11	Safari	G	R
29	Win11	Firefox	U	C
30	Win11	Firefox	U	R
31	Win11	Firefox	G	C

32	Win11	Firefox	G	R
33	Win11	Chrome	U	C
34	Win11	Chrome	U	R
35	Win11	Chrome	G	C
36	Win11	Chrome	G	R

Pair-wise Table (Coverage Matrix)

To define a pairwise (or t=2) test suite, you need a minimum set of tests that ensures every combination of two variables' values is covered at least once.

The maximum number of pairs to cover is determined by the largest product of two variable sizes, which is 3×3=9 (OS-Browser, OS-Browser, Browser-Browser).

Test Case	OS	Browser	Student Type	Location
1	iOS	Safari	U	C
2	iOS	Firefox	G	R
3	iOS	Chrome	U	R
4	Linux	Safari	G	R
5	Linux	Firefox	U	C
6	Linux	Chrome	G	C
7	Win11	Safari	U	R
8	Win11	Firefox	G	C
9	Win11	Chrome	U	C

Comment on Number of Test Cases: The pairwise table successfully reduces the test cases from 36 (all combinations) to 9 tests. This represents a 75% reduction in testing effort while ensuring that every pair of factor values appears at least once across the suite, which is highly effective for finding interaction-related bugs.

Alphabet Size

The alphabet size is the maximum number of distinct values (levels) any single factor possesses.

- OS: 3 levels (iOS, Linux, Win11)
- Browser: 3 levels (Safari, Firefox, Chrome)
- Student Type: 2 levels (U, G)
- Student Location: 2 levels (C, R)

The alphabet size is 3.

Proper Orthogonal Array (S=2, I=1)

For pairwise testing, we require Strength t= 2. Since the largest alphabet size is 3 and we have N = 4 factors, the appropriate uniform Orthogonal Array is the $L_9(3^4)$ array.

- Proper Orthogonal Array: $L_9(3^4)$
    - This array has 9 runs (test cases).
    - It accommodates up to 4 factors, each with 3 levels.
    - It guarantees strength 2 (pairwise coverage), meaning every combination of 2 factor values appears at least once (index I =1).

Populated Orthogonal Array

The $L_9(3^4)$ array is populated below. The 3-level factors (OS, Browser) use all levels. The 2-level factors (Student Type, Location) are mapped to two of the three levels, with the third level mapped back to the first or second level cyclically to maintain balance.

Test Case	OS	Browser	Student Type	Location
1	iOS	Safari	U	C
2	iOS	Firefox	G	R
3	iOS	Chrome	U	R
4	Linux	Safari	G	R
5	Linux	Firefox	U	C
6	Linux	Chrome	G	C
7	Win11	Safari	U	R
8	Win11	Firefox	G	C
9	Win11	Chrome	U	C

Comparison of Test Case Numbers

Method	Number of Test Cases
All-Combinations	36
Pairwise (Manual)	9
Pairwise-OA $L_9(3^4)$	9

Comment on the Number of Test-Cases Generated:

The all-combinations approach generates 36 test cases, providing the highest coverage (every interaction) but requiring the maximum time and resources.

Both the manual pairwise construction and the systematic Pairwise-OA using the array $L_9(3^4)$ yield a minimum of 9 test cases. This demonstrates a 75% reduction in test effort compared to all-combinations. The OA approach is preferable as it mathematically guarantees the pairwise coverage and often results in a more balanced distribution of tests.

Question 2

Part 2: [25% of the mark]

You are developing and application to support bicycle stores. Your application has the following configuration variables options:
1) Sales provided on-line only or in-store only.
2) Store located in USA or Canada.
3) Payment accepted by Visa or AMEX Only (no cash).
4) Store may only sell bicycles or provide maintenance services as well.

You are required to answer the following questions:

1. Calculate the number of all-combinations test-cases.
2. What is the pairwise orthogonal array that you can use for this problem? How many test-cases does it represent?
   (You do not need to fill-in the OA, just identify the array using the format on lecture-slides 38-39)
3. If you had 7 variables with 2 values each, which array would you use? What is the number of test-cases?
   (You do not need to fill-in the OA, just identify the array using the format on lecture-slides 38-39)
4. How many test-cases does an L8 pairwise orthogonal array represent?

Answer Part 2:

The application has four configuration variables (), and each variable has two values (S = 2levels):

1. Sales (S): 2 values (online, in-store)
2. Location (L): 2 values (USA , Canada)
3. Payment (P): 2 values (Visa, AMEX)
4. Service (V): 2 values (Bicyclesonly , bicycles and maintence)
5. The total number of all-combinations (full factorial) test cases is the product of the number of values for each variable: 2 x2 x2x2 = 16

Here is the solution for Part 2 of your software testing problem, concerning the bicycle store application.

The application has four configuration variables (), and each variable has two values (levels):

1. Sales (S): 2 values ()
2. Location (L): 2 values ()
3. Payment (P): 2 values ()
4. Service (V): 2 values ()

Pairwise Orthogonal Array for 4 Variables (2 Levels Each)

For variables,N =4 all with levels,S= 3 the smallest standard Orthogonal Array (OA) that guarantees pairwise (t=2) coverage is the array $L_s(2^7)$. We use only 4 of the 7 available columns.

- Pairwise Orthogonal Array: $L_s(2^7)$.
- Number of Test-Cases: This array represents 8 test cases (runs).

Orthogonal Array for 7 Variables (2 Values Each)

- If you had variables N=7, all with S =2values each, the array to use is the array $L_s(2^7)$, as it perfectly accommodates this configuration while maintaining pairwise coverage.

- Orthogonal Array: $L_s(2^7)$.

- Number of Test-Cases: This array represents 8 test cases.

Test-Cases in an L_8 Pairwise Orthogonal Array

The notationL_t where T is the subscript, represents the number of rows or test cases in the array.

- An L_8 pairwise orthogonal array represents 8 test cases

Part 3: [25% of the mark]

Your company provides online books for various readers. The ones you need to worry about are Kindle, iPad and Zok. There are 4 different classes of books you need to worry about - textbooks (which have lots of equations), poetry (where the formatting is extremely important), graphic novels, and regular novels. They also can be ordered in three different languages, English, Spanish, and Japanese.

You are required to answer the following questions:

1. What is the total number of test cases for all-combinations?
2. What is the minimum number of tests for pairwise testing?
   (You may develop a pairwise table or use simple calculation, try to make your ideas clear for the TA).
3. You decide to use pairwise orthogonal arrays to help with your testing. Which pairwise orthogonal array should you use?
4. What do you think of mixed alphabet? Does it add additional burden? Can you use it to your advantage? Present your own opinion.

Answer Part 3:

The configuration variables and their values are:

- Reader (R): 3 values — Kindle, iPad, Zok
- Book Class (BC): 4 values — Textbooks, Poetry, Graphic Novels, Regular Novels
- Language (L): 3 values — English, Spanish, Japanese

All-combinations

- Total tests: $3 \times 4 \times 3 = 36$

Pairwise minimum

- Pair counts:
  - $R \times BC = 3 \times 4 = 12$
  - $R \times L = 3 \times 3 = 9$
  - $BC \times L = 4 \times 3 = 12$
- Minimum tests for pairwise: 12
- Rationale: The lower bound equals the largest pairwise Cartesian product, so at least 12 tests are needed; a mixed-level covering array can achieve $N = 12$.

OA selection

- Nature: Mixed-level problem $(3,4,3)$.
- Uniform OA option (standard Taguchi): use $L16(4^5)$with 16 runs (use any 3 of the 5 columns).
- Minimal option (preferred for size): a mixed-level covering array with $N = 12$, $t = 2$, levels $3^2 4^1$.

Mixed alphabet

Burden

- Standard orthogonal arrays are uniform-level, so mixed alphabets either force an oversized uniform OA (e.g., 16 runs with $L16(4^5)$) or require generating a mixed-level OA/covering array with specialized tools, adding process complexity.

Advantage

- Mixed-level covering arrays can match the true lower bound (12 here), avoiding wasted runs from oversized uniform OAs and yielding a smaller, more targeted suite aligned to real system constraints.

Part 4: [25% of the mark]

On a software testing project that you joined recently, you found the following set of conditions ready for you to use:

1. You have the following four variables,
    1. Variable A has 3 eq. classes {a_1, a_2, a_3} where: both a_1, and a_3 are invalid, but a_2 is valid.
       also, A has the following boundaries {a_{min-}, a_{min}, a_{min+}, a_{max-}, a_{max}, a_{max+}}
       where: a_{min} is a valid value for A at the border between a_1, a_2, and a_3 , and
       a_{min} is a valid value for A at the border between a_2, a_3.
    2. Variable B takes one of four values only {S for a Single, M for Married, D for Divorced, or W for Widow}
       Variable B cannot take any entry other than {S, M, D, W}. This is done using drop-down dialogue-box.
       So, we are confident there will be no invalid values for B.
    3. Variable C represents weekdays {Mo, Tu, We, Tr, Fr} and like variable B, it cannot take any invalid entry.
    4. Variable D is {Y, N}, and like variable B, it cannot take any invalid entry.
2. The management is tight on time, budget, and the development is already halfway ready.
3. The management gave you a free hand to apply, BVA, Eq. Class, DT, Pairwise, Pairwise OA, or any combination of those
   techniques based on what you see best for the company.
4. Given the time limitations, you can only design and execute a minimum of 80 test-cases or a maximum of 120 test-cases.
   The management believe that less testing would be risky, and there is no time for more testing.

Answer Part 4:

A blended test strategy using boundary value analysis for A, equivalence partitioning for A, a mixed-level pairwise covering array for A×B×C×D, decision-table checks for business rules, and targeted negative/3-way stress tests yields a balanced, high-yield suite of 100 tests within the 80–120 limit.

Problem recap

- Variables: A has three equivalence classes {a1 invalid, a2 valid, a3 invalid} and boundaries {amin−, amin, amin+, amax−, amax, amax+}.
- Variables: B={S, M, D, W}, C={Mo, Tu, We, Tr, Fr}, D={Y, N}, each constrained to valid values via UI controls.

- Constraint: Deliver between 80 and 120 executed tests, maximizing defect detection efficiency under time and budget limits.

Objectives

- Prioritize fault-prone edges in A while validating representative behavior inside each partition.
- Achieve interaction coverage efficiently across A, B, C, and D, with focused higher-order checks where risk warrants.

Boundary analysis (A)

- Focus testing on the six edges {amin−, amin, amin+, amax−, amax, amax+} to expose boundary and off-by-one errors.
- Combine each boundary with a small, diverse set of (B, C, D) tuples to surface interaction defects at edges without exploding combinations.

Equivalence classes (A)

- Exercise a1 and a3 as invalid classes and multiple mid-range representatives in a2 as valid to cover typical paths and guard against internal non-uniformity.
- Pair class representatives with stable (B, C, D) settings for clarity, then vary context in a subset to check consistent handling.

Pairwise interactions

- Use a mixed-level covering array with strength $t = 2$over A_rep(3)×B(4)×C(5)×D(2) so every value pair across any two factors appears in at least one test.
- Seed A_rep with {a1, a2, a3} and integrate a subset of boundary values into A where operationally relevant to increase edge exposure within the pairwise grid.

Decision tables

- Model key business rules and outcomes tied to combinations of B, C, and D, with special cases that involve A's validity and boundary status.
- Ensure rule completeness, conflicts, and precedence are covered beyond structural t-wise coverage.

Negative and robustness

- Validate rejection, error messaging, and stability for a1/a3 and near-boundary values (amin±, amax±), including idempotent or retry behaviors where applicable.
- Combine invalid A values with high-risk or high-traffic selections of B, C, and D to maximize practical fault yield.

3-way stress (targeted)

- Add a small, risk-based set of 3-way combinations for sensitive triads such as (A_rep, B, C) or (A_rep, C, D).
- Focus on combinations implicated by past defects, complexity hot spots, or regulatory rules.

Test count plan

Component	Rationale	Tests
A BVA edges	6 boundaries × 4 varied B/C/D samples	24
A equivalence classes	Invalid a1/a3 coverage and multiple a2 reps	8
Pairwise CA on A×B×C×D	Mixed-level covering array, t=2t=2	32
Decision-table checks	Rule completeness, conflicts, precedence	16
Negative/robustness	Invalid handling and near-boundary resilience	12
3-way stress (targeted)	Higher-order risk hot spots	8
Total	Fits 80–120 with strong coverage	100

This mix works because boundary value analysis and equivalence partitioning focus tests where faults cluster while still validating representative, everyday behavior; pairwise covering arrays provide high interaction coverage at low cost, and decision-table testing confirms business-rule correctness that pure t-wise coverage may overlook; targeted negatives and a small 3-way stress slice then hedge against higher-order and robustness defects without exceeding the execution budget. For implementation, define A_rep as {a1, a2, a3} and substitute boundary values into selected pairwise cases where operationally relevant, generate a mixed-level pairwise set of 32 tests, verify pair coverage, align the resulting configurations with decision-table scenarios, and trace each test back to its technique and requirement for auditability.