Problem A
3 Partition

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

You are given $3 \cdot N$ numbers $A_1, A_2, \ldots, A_{3N}$ $(1 \leq A_i \leq 3)$.
You need to partition these $3N$ integers into N triplets, each of size 3. Each of the triplets must have the same sum.

Formally, let $B_i = [C_{(i,1)}, C_{(i,2)}, C_{(i,3)}]$ be the i-th triplet, then $C_{(i,1)} + C_{(i,2)} + C_{(i,3)} = C_{(j,1)} + C_{(j,2)} + C_{(j,3)}$ for all $1 \leq i, j \leq N$.

Does a valid partition exist?
Print `Yes` if it is possible, or `No` otherwise. You do not need to construct the triplets themselves.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains N - the number of triplets needed.
  - The second line of each test case contains $3 \cdot N$ integers - $A_1, A_2, \ldots, A_{3N}$.

Output Format

For each test case, output on a new line `Yes` or `No` depending on whether it is possible.

It is allowed to print each character in either case, i.e. `yes`, `YES` and `yEs` will all be accepted as a positive response.

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 10^5$
- $1 \leq A_i \leq 3$
- The sum of N over all test cases does not exceed 10^5

Samples

Sample Input 1

```
5
2
3 2 2 2 2 1
2
1 1 1 1 1 2
1
1 1 2
2
1 1 1 2 2 3
2
1 1 1 1 2 3
```

Sample Output 1

```
Yes
No
Yes
Yes
No
```

Sample Explanation

Test Case 1 : We can form the 2 triplets as $[1, 2, 3]$ and $[2, 2, 2]$. Both sum to exactly 6.

Test Case 2 : There exists no valid triplet partition.

Problem B
Cache Optimization

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

Cache optimizations can significantly impact program performance, often making a program run faster than anticipated. To illustrate this, let us consider a simplified model of a cache.

Suppose the cache has a size K. This means the cache can store the memory locations of up to K variables. If fewer than K variables have been accessed, the cache contains all of them only - in particular, the cache is empty before any variables are accessed.
When a new variable is accessed beyond the K-variable limit, the least recently used variable is removed from the cache to make space for the new one.

In this task, we simulate the behavior of a CPU cache to determine the total CPU cycles required for memory access.

- If the memory location of a variable is present in the cache, it takes 1 CPU cycle to access it.
- If the memory location is not in the cache, it takes 2 CPU cycles to retrieve it.

You have a list of N variables you will use in that order. Some of the variables might be repeated. You would like to compute the total number of CPU cycles the program will use. You may assume that the only time-consuming part is the memory accesses.

Further, the exact value of K is not known to you. So, we require you to solve the problem for each $K = 1, 2, \ldots, N$.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains N - the number of variables.
  - The second line contains N integers, $A_1, A_2, \ldots, A_N$ - the IDs of the N variables.

Output Format

For each test case, output N space-separated integers - the total CPU cycles taken for $K = 1, 2, \ldots, N$ in that order.

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq A_i \leq N$
- The sum of N over all test cases does not exceed $2 \cdot 10^5$

Samples

Sample Input 1

```
3
2
1 1
3
3 2 1
5
1 2 1 3 2
```

Sample Output 1

```
3 3
6 6 6
10 9 8 8 8
```

Sample Explanation

Test Case 3 : Let's solve the problem for $K = 2$.

- Initially, your cache is empty.
- You query for the location of the 1st variable. This is not present in your cache, so it takes 2 CPU cycles. Your cache is now $\{1\}$.
- You query for the location of the 2nd variable. This is not present in your cache, so it takes 2 CPU cycles. Your cache is now $\{2, 1\}$, ordered in descending order of recency.
- You query for the location of the 1st variable. This is present in your cache, so it takes 1 CPU cycles. Your cache is now $\{1, 2\}$, ordered in descending order of recency.
- You query for the location of the 3rd variable. This is not present in your cache, so it takes 2 CPU cycles. Further, this gets added to your cache, however it must remove one variable otherwise you exceed the $K = 2$ limit. 2 gets removed from cache as 1 is more recent. Thus, your cache is $\{3, 1\}$, ordered in descending order of recency.
- You query for the location of the 2nd variable. This is not present in your cache, so it takes 2 CPU cycles. Your cache gets updated to $\{2, 3\}$.

Thus, the total is $2 + 2 + 1 + 2 + 2 = 9$.

Problem C
Composite Madness

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

You really like composite[†] numbers. Therefore, you call a number N *good* if and only if it can be decomposed into K ($K \geq 1$) composite numbers $A_1, A_2, \ldots, A_K$ satisfying the following conditions:

- $A_1 + A_2 + \ldots + A_K = N$
- Each A_i is composite.

Given a number N, check if it is *good*.

[†] An integer X is said to be composite if and only if there exists some integer Y such that Y divides X and $1 < Y < X$. Notably, 1 is not composite.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- The first and only line of input of each test case contains a single integer N.

Output Format

For each test case, output `Yes` if N is *good*, otherwise `No`.

It is allowed to output each character in either lower or upper case. For example, `YES`, `yes` and `yEs` will all be taken as positive responses.

Constraints

- $1 \leq T \leq 1000$
- $1 \leq N \leq 10^9$

Samples

Sample Input 1

```
3
52
4
5
```

Sample Output 1

```
Yes
Yes
No
```

Sample Explanation

Test case 1: We can decompose 52 into $(25 + 27)$, both of which are composite. Thus, 52 is a *good* number.

Test case 2: We decompose 4 into (4) only, which is composite. Thus, 4 is a *good* number.

Problem D
Delete if Equal

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

You are given a binary string S of size N. You can do the following operation as many times as you want:

- Choose an index i satisfying $1 < i < |S|$ and $S_{i-1} = S_{i+1}$, and delete S_i from the string. Then, concatenate the 2 parts $S[1, i-1]$ and $S[i+1, |S|]$ to form the new string S. Note that the length of S reduces by 1 after each operation.

Find the minimum possible length of the final string after performing any number of operations.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains N - the length of the binary string S.
  - The second line contains S - a binary string of size N.

Output Format

For each test case, output on a new line the minimum possible length of the final string.

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 2 \cdot 10^5$
- $S_i \in \{0, 1\}$
- $|S| = N$
- The sum of N over all test cases does not exceed $2 \cdot 10^5$

Samples

Sample Input 1

```
6
3
101
4
0010
1
1
4
0101
5
11011
5
01001
```

Sample Output 1

```
2
2
1
3
2
3
```

Sample Explanation

Test case 1: We choose $i = 2$ in the first and only operation, and since $S_1 = S_3$, we delete S_2. The final string is 11. Note that no more operations are possible because it is impossible to pick i satisfying $1 < i < |S|$.

Test case 2: We perform the following sequence of operations:

- $i = 3$: $S_2 = S_4 = 0$ so this is valid. We delete S_3 from the string to get 000.
- $i = 2$: After the first operation, $S_1 = S_3 = 0$ so this is valid. We delete S_2 from the string to get 00.

Thus, our answer is 2.

Problem E
Expected Rain

Time limit: 2 seconds
Memory limit: 1.5 Gigabytes

Problem Description

There are N buildings on a 1-D line. The i-th building spans from x-coordinate $(i-1)$ to x-coordinate i, and has a height of H_i.

When rain falls, water gets collected between the buildings. There are no buildings to the left of $x = 0$ or to the right of $x = N$, and so no water gets collected there.

Let $f(H)$ denote the amount of rain collected.

Unfortunately, you only remember the N heights, but not the order in which they were present. Find the expected value of $f(H)$ over all possible permutations of H.

The expected value must be computed modulo 998244353.
Formally, it can be proven that the answer can be represented as a rational number $\frac{P}{Q}$ where $Q \not\equiv 0 \pmod{998244353}$. Find the unique integer R in $[0, 998244353)$ such that $R \cdot Q \equiv P \pmod{998244353}$.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains a single integer N - the number of buildings.
  - The second line of each test case contains N space-separated integers - $H_1, H_2, \ldots, H_N$.

Output Format

For each test case, output on a new line the expected value of the collected rain modulo (998244353).

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 5 \cdot 10^5$
- $1 \leq H_i \leq N$
- The sum of N over all test cases does not exceed $5 \cdot 10^5$

Samples

Sample Input 1

```
4
2
1 2
3
3 2 1
4
1 1 2 2
5
1 2 3 4 5
```

Sample Output 1

```
0
332748118
665496236
798595485
```

Sample Explanation

Test case 1: There are 2 possible rearrangements. Both have no collected rain. Hence the answer is 0.

Test case 2: There are 6 permutations possible. $[2, 1, 3]$ and $[3, 1, 2]$ both have collected rain of 1 unit, and the other 4 have 0 collected rain. Hence, the answer is $\frac{2}{6} \equiv 332748118 \pmod{998244353}$.

Problem F
Flip on Cycle

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

Consider a grid graph with $N \cdot M$ nodes. Let's represent each node by a pair of numbers (i, j) - the row number i and the column number j. Two nodes (i_1, j_1) and (i_2, j_2) are said to be adjacent if and only if they share an edge, formally $|i_2 - i_1| + |j_2 - j_1| = 1$.

You have an $N \times M$ matrix A, where each $A_{i,j}$ is a binary number, i.e., 0 or 1.
You would like to convert A to another binary matrix B of the same dimensions.

To do this, you can use the following operation at most 500 times:

- Choose a simple cycle in the graph. Flip the values of each cell in the cycle.
  Formally, for all nodes (X, Y) in the cycle replace $A_{X,Y}$ by $1 - A_{X,Y}$.

If it is possible to convert the matrix A into B, output a possible sequence of operations of length at most 500. Otherwise, output -1.
In particular, note that you **do not have to minimize the number of operations used**.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of multiple lines of input.
  - The first line of each test case contains two space-separated integers N and M — the dimensions of the matrix.
  - The i-th of the next N lines each contain M integers - $A_{i,1}, A_{i,2}, \ldots, A_{i,M}$.
  - The i-th of the next N lines each contain M integers - $B_{i,1}, B_{i,2}, \ldots, B_{i,M}$.

Output Format

For each test case, output the following:

- If we can convert A to B, first output op ($0 \leq op \leq 500$), the number of operations used. Then output the details of the op operations. For each operation:
  - First, output K ($3 \leq K \leq N \cdot M$), the number of nodes present on the simple cycle chosen in this operation.
  - Then output a sequence of $2 \cdot K$ integers $i_1, j_1, i_2, j_2, \ldots, i_K, j_K$ where (i_l, j_l) represents a node in the cycle. These K nodes must form a simple cycle in this order.
- If we cannot convert A to B, output -1 on a new line.

Constraints

- $1 \leq T \leq 500$
- $2 \leq N, M \leq 100$
- $N \cdot M \leq 200$
- $0 \leq A_{i,j}, B_{i,j} \leq 1$
- The sum of $N \cdot M$ over all test cases does not exceed 2000.

Samples

Sample Input 1

```
3
2 3
1 0 1
0 1 0
0 1 0
1 0 1
2 2
0 0
0 0
0 1
0 1
2 2
0 0
0 0
0 0
0 0
```

Sample Output 1

```
1
6
1 1 1 2 1 3 2 3 2 2 2 1
-1
0
```

Sample Explanation

Test Case 1 : We simply choose a cycle going through all 6 nodes in the grid. This flips each and every cell in the grid, and converts A to B.

Test Case 2 : It can be proven to be impossible.

Test Case 3 : $A = B$ already, so no operations are needed.

Problem G
Infinite Operations

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

Alice has N boxes, numbered 1 to N, where the i-th box contains A_i stones.

Alice can perform the following two-step operation:

- Pick two different boxes, say X and Y, such that the number of stones in box X and box Y **are not equal**.
- Then, move exactly 1 stone from the box with more stones to the box with lesser stones. For example, if $A_X > A_Y$, she will remove 1 stone from box X and place that stone in box Y.

Is it possible for her to choose operations such that she is able to continue doing this indefinitely?

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of input contains N - the number of boxes.
  - The second line of input contains N integers - $A_1, A_2, \ldots, A_N$, the number of stones in each box.

Output Format

For each test case, output on a new line `Yes` if Alice can continue doing operations infinitely, otherwise output `No`.

It is allowed to print each character in either case - `YES`, `yes`, and `yEs` will all be considered positive responses.

Constraints

- $1 \leq T \leq 100$
- $2 \leq N \leq 100$
- $1 \leq A_i \leq 100$

Samples

Sample Input 1

```
2
2
1 2
3
1 1 3
```

Sample Output 1

```
Yes
Yes
```

Sample Explanation

Test Case 1: Alice can always choose boxes 1 and 2, and alternate between the states $[1, 2]$ and $[2, 1]$.

Problem H
Keep it Simple (and Bipartite) Stupid!

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

Alice and Bob have been given a **simple**[†] **bipartite**[‡] graph G with N nodes and M edges. They play a game on this graph.

Alternatively, starting from Alice, each player will add exactly 1 edge to the graph G. The graph must remain bipartite and simple after each edge addition.

The player unable to move loses, and the other player is declared the winner. Predict the winner of the game under optimal play from both players.

[†] A graph is said to be **simple** if there is at most one edge between any 2 pairs of nodes, and there is no edge between a node and itself. (No multi edges and no self edges.)

[‡] A graph is said to be **bipartite** if the nodes can be partitioned into 2 sets S and T such that there is no edge between 2 nodes in S, and similarly, there is no edge between 2 nodes in T.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of multiple lines of input.
  - The first line of each test case contains two space-separated integers N and M — the number of vertices and edges of the graph, respectively.
  - The next M lines describe the edges. The i-th of these M lines contains two space-separated integers u and v, denoting an edge between u and v.

Output Format

For each test case, output on a new line `Alice` or `Bob`, denoting the winner of the game.

It is allowed to print each character in either upper or lower case. `aLiCe`, `ALICE` and `alice` will all be accepted.

Constraints

- $1 \leq T \leq 10^5$
- $1 \leq N \leq 2 \cdot 10^5$
- $0 \leq M \leq 2 \cdot 10^5$
- $1 \leq u, v \leq N$ and $u \neq v$
- Each unordered pair (u, v) occurs at most once.
- The given graph is bipartite.
- The sum of N and the sum of M both do not exceed $2 \cdot 10^5$.

Samples

Sample Input 1

```
3
3 2
1 2
2 3
2 0
3 1
1 2
```

Sample Output 1

```
Bob
Alice
Alice
```

Sample Explanation

Test case 1: On the first turn itself, Alice has no move. The only possible edge to add while keeping the graph simple is $(1, 3)$, but this edge breaks the bipartite condition. Thus, she loses.

Test case 2: The given graph has 2 nodes and 0 edges. Alice adds the edge $(1, 2)$ on her first turn. Now, it's Bob's turn, but there are no edges to add while keeping the graph simple.

Problem I
Majority Partition

Time limit: 1 second
Memory limit: 1.5 Gigabytes

Problem Description

An integer X is the **majority** of sequence B if it occurs **strictly more** than $\frac{|B|}{2}$ times, where $|B|$ represents the length of B.
A sequence B is considered *awesome* if it contains an element X which is a **majority**.

You are given a sequence A of length N. You must partition A into several subsequences such that:

- Each element is included in exactly one subsequence.
  Note that elements in a subsequence are not required to be adjacent.
- Every subsequence is *awesome*.

Output the minimum number of subsequences that you must partition A into. It can be shown it is always possible to partition A into some number of subsequences such that both conditions are satisfied.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains N - the size of the array.
  - The second line contains N integers - $A_1, A_2, \ldots, A_N$.

Output Format

For each test case, output on a new line the minimum number of subsequences that you must partition A into.

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq A_i \leq N$
- The sum of N over all test cases does not exceed $2 \cdot 10^5$.

Samples

Sample Input 1

```
3
4
1 2 3 4
4
1 1 1 2
6
1 1 2 2 3 3
```

Sample Output 1

```
4
1
2
```

Sample Explanation

Test Case 1: The optimal and only valid division is [1], [2], [3] and [4].

Test Case 2: The whole array can be taken as one sequence as 1 is a majority element.

Problem J
Make Equal Under Mod

Time limit: 2 seconds
Memory limit: 1.5 Gigabytes

Problem Description

You have an array A consisting of N integers. You can do the following operation as many times as you want:

- Choose i and X satisfying $1 \leq i \leq N$ and $1 \leq X \leq 10^7$, and change A_i to $A_i \bmod X$.

It can be proven that it is possible to make all elements of A equal.
Let $f(A)$ denote the maximum possible value of A_1 after all the elements of A have been made equal.

As usual, you have to solve a harder version of this task.
Given an array A of N integers, find the sum of all answers over all subarrays of A, i.e.

$$\sum_{L=1}^{N} \sum_{R=L}^{N} f([A_L, A_{L+1}, \ldots, A_R])$$

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case consists of two lines of input.
  - The first line of each test case contains N - the length of the array A.
  - The second line contains N integers - $A_1, A_2, \ldots, A_N$.

Output Format

For each test case, on a new line output the sum of f over all subarrays of A.

Constraints

- $1 \leq T \leq 10^4$
- $1 \leq N \leq 2 \cdot 10^5$
- $1 \leq A_i \leq 10^7$
- The sum of N does not exceed $2 \cdot 10^5$.

Samples

Sample Input 1

```
5
2
1 2
3
2 1 3
3
4 4 3
5
2 3 1 4 2
5
3 7 7 3 3
```

Sample Output 1

```
3
7
17
15
57
```

Sample Explanation

Test Case 1: There are 3 subarrays of A:

- $[1]$: No operations needed. Array elements are all already equal. $f([1]) = 1$.
- $[2]$: No operations needed. Array elements are all already equal. $f([2]) = 2$.
- $[1, 2]$: First choose $i = 1, X = 1$, and then $i = 2, X = 1$. This changes the array to $[0, 0]$ and it can be proven to be optimal. $f([1, 2]) = 0$.

Thus, the sum of f over all subarrays is 3.

Problem K
Max Mod

Time limit: 1.5 seconds
Memory limit: 1.5 Gigabytes

Problem Description

You have an array A of N integers. Initially, all $A_i = 0$. You are also given a **prime** M.

Handle updates and queries of the following format:

1. Given X, Y: add $X + (i-1) \cdot Y$ to A_i for all $1 \leq i \leq N$.
2. Given L, R: find the maximum value of $(A_i \bmod M)$ across all $L \leq i \leq R$.

Input Format

- The first line of each test case contains 3 integers - N, M and Q.
- The next Q lines each contain 3 integers in one of the 2 following formats:
  - $T = 1, X, Y$: representing an update with parameters X and Y.
  - $T = 2, L, R$: representing a query with parameters L and R.

Output Format

For each test case, output on a new line the maximum value of $A_i \pmod M$ for $L \leq i \leq R$ for each query.

Constraints

- $1 \leq N, Q \leq 5 \cdot 10^5$
- $1 \leq M \leq 10^9$
- M is prime
- $1 \leq T \leq 2$
- $1 \leq X, Y \leq 10^9$
- $1 \leq L \leq R \leq N$

Samples

Sample Input 1

```
5 37 6
1 2 7
1 9 13
2 2 4
1 29 3
2 1 3
2 3 5
```

Sample Output 1

```
34
26
35
```

Sample Explanation

Test Case 1: The array changes in the first 3 steps as follows:

- **Update 1**: The array is updated to $[2, 9, 16, 23, 30]$.
- **Update 2**: The array is updated to $[11, 31, 51, 71, 91]$.
- **Query 1**: $A_2 \pmod{37} = 14$, $A_3 \pmod{37} = 34$, $A_4 \pmod{37} = 17$. Thus, the maximum is 34.

Problem L
Ticket Revenue Maximization

Time limit: 4 seconds
Memory limit: 1.5 Gigabytes

Problem Description

There is going to be a tournament with 128 teams numbered $1, 2, \ldots, 128$. For each $i < j$, the team j is stronger than the team i, and in a match between them, the team j always wins. For each i, the team i has P_i supporters. No two teams have any common supporters.

There will be 7 rounds in the tournament. In each round, the teams will be divided into pairs, such that each team belongs to **exactly one** pair. Each pair will play a match, with the loser being eliminated and the winner proceeding to the next round (if any).
Formally,

- In the first round, the 128 teams will be paired into 64 matches. Hence, 64 teams will be eliminated, and the rest will move to the next round.
- In the second round, the 64 winning teams from the first round will be paired into 32 matches.
- In the third round, the 32 winning teams from the second round will be paired into 16 matches.
- and so on, until the final (seventh) round, where the two remaining teams will play a match.

Naturally, the later rounds are more valuable. Hence, the ticket of the $r^{\texttt{th}}$ round costs r. A match between teams i and j in the $r^{\texttt{th}}$ round will therefore create a revenue of $r \cdot (P_i + P_j)$, as each supporter of any of the two teams will buy the tickets for the match, each costing r.

You are in-charge of the pairings in all the rounds, and your objective is to select the pairings in such a way that the total revenue generated through the tournament is maximized. Find this maximum revenue.

Input Format

- The first line of input will contain a single integer T, denoting the number of test cases.
- Each test case contains a single line of input which has 128 integers - $P_1, P_2, \ldots, P_{128}$.

Output Format

For each test case, output on a new line the maximum possible revenue.

Constraints

- $1 \leq T \leq 5$
- $1 \leq P_i \leq 10^9$

Samples

Sample Input 1

```
2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
128 127 126 125 124 123 122 121 120 119 118 117 116 115 114 113 112 111
110 109 108 107 106 105 104 103 102 101 100 99 98 97 96 95 94 93 92 91
90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67
66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43
42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

Sample Output 1

```
494
30318
```

Note

The sample input 1 in this document has been split into multiple lines to fit them on the page. In reality, all the 128 numbers are present on a single line in the tests.

Sample Explanation

Test case 1: All matches have revenue of $r \cdot 2$ where r is the round number, since all $P_i = 1$.

- Round 1: 64 matches, revenue $64 \cdot 2 = 128$
- Round 2: 32 matches, revenue $32 \cdot 4 = 128$
- Round 3: 16 matches, revenue $16 \cdot 6 = 96$
- Round 4: 8 matches, revenue $8 \cdot 8 = 64$
- Round 5: 4 matches, revenue $4 \cdot 10 = 40$
- Round 6: 2 matches, revenue $2 \cdot 12 = 24$
- Round 7: 1 match, revenue $1 \cdot 14 = 14$

The total revenue is $128 + 128 + 96 + 64 + 40 + 24 + 14 = 494$.