Function: decimalToBinary

Input:

- **arr**: 2D array to store binary representation

- **decimalNumber**: decimal number to convert to binary

- **numPlaces**: number of bits in the binary representation

Return value:

- None (results are stored in the array arr)

Algorithm:

- Initialize an array binary of size numPlaces to store the binary representation.
- Loop from i = 0 to numPlaces - 1:

    a. Set binary[i] to the least significant bit of decimalNumber (decimalNumber & 1).

    b. Right shift decimalNumber by 1 (decimalNumber >>= 1).

- Initialize a variable count to 0.
- Loop from i = 0 to numPlaces - 1:

    a. If binary[i] is 1, increment count.

- If count is greater than “combinationCount”:

    a. Loop from i = numPlaces - 1 to 0:

        i. Store binary[i] in the array arr[ctr][i].

- End of function.

Function: compareArticles (for sorting the articles)

Input:

- **a**: pointer to the first Article structure

- **b**: pointer to the second Article structure

Output:

- Returns -1 if the cost of the first article is less than the cost of the second article.

- Returns 1 if the cost of the first article is greater than the cost of the second article.

- Returns 0 if the costs of both articles are equal.

Algorithm:

- Cast the void pointers a and b to pointers of type struct Article.
- 2. Compare the cost of the two articles.

    a. If the cost of structA is less than the cost of structB, return -1.

    b. If the cost of structA is greater than the cost of structB, return 1.

    c. If the costs are equal, return 0.

End of algorithm.

Function: main

Input:

- **argc**: number of command-line arguments

- **argv**: array of command-line arguments

Output:

- Returns 0 on successful completion

Algorithm:

- Check if the correct number of command-line arguments is provided.

  a. If not, print usage information and exit.

- Initialize a timer to measure the computation time.
- Extract the input file name from command-line arguments.
- Open the input file for reading.

  a. If the file cannot be opened, print an error message and exit.

- Read the number of articles, reporters, required clicks, and article details from the input file.
- Sort the articles using the `qsort` function based on a comparison function (`compareArticles`).
- Check for duplicate articles with the same type and reporter, and adjust the number of articles accordingly.
- Calculate the number of combinations and allocate memory for the `output` and `arr` arrays.
- Find the combinationCount by getting the number of unique reporters
- If combinationCount < 4, Then combinationCount = 4 (Type of articles = 4)
- Iterate over all the remaining combinations:

  a. Convert the current combination index to binary using the `decimalToBinary` function.

  b. If the count of set bits in the binary representation is less than or equal to 3, continue to the next iteration.

  c. Check constraints using the `checkConstraints` function.

  d. Free memory for the current combination.

- Stop the timer and calculate the computation time in hours, minutes, and seconds.
- Extract the base file name from the input file name and create the output file name.
- Open the output file for writing.

  a. If the file cannot be opened, print an error message and exit.

- Print the selected articles to the output file along with the total cost, total clicks, and computation time.
- Close the output file.
- Print a message indicating the successful creation of the output file.
- Return 0 to indicate successful completion.

End of algorithm.

Time Taken to get the Optimal Solution:

Files	**P1**	**P2**
Ex11.txt	2 min 17 seconds	59 seconds
Ex12.txt	4 min 31 seconds	2 min 3 seconds
Ex13.txt	8 min 24 seconds	4 min 9 seconds