```c
// Created by hfwei on 2024/10/16.

#include <stdio.h>

#define LEN_L 5
#define LEN_R 6

int L[LEN_L] = { 1, 3, 5, 7, 9 };
int R[LEN_R] = { 0, 2, 4, 6, 8, 10 };

int main(void) {
  // TODO: merge L and R into a sorted array

  return 0;
}
```

```
 1 # 4-loops
 2
 3 - `Alt + 6`: Problems on the status bar
 4 - `SonarLint` on the status bar
 5
 6 ## `game-of-life.c`
 7
 8 - play with it
 9   - [wiki](https://en.wikipedia.org/wiki/Conway%
   27s_Game_of_Life)
10   - [Demo](https://playgameoflife.com/)
11   - [Gosper_glider_gun](https://playgameoflife.com/lexicon
   /Gosper_glider_gun)
12   - [LifeWiki](https://conwaylife.com/wiki/Main_Page)
13   - [Life Lexicon Home Page](https://conwaylife.com/ref/
   lexicon/lex_home.htm)
14 - 2D-array
15   - initialization (Section 8.2.1)
16     - row-major
17     - row by row
18     - indicator
19 - extension of board
20 - how many boards?
21 - one round
22 - multiple rounds
23 - pause
24 - screen clear
25 - [ ] try a new board?
26   - [Life Lexicon Home Page](https://conwaylife.com/ref/
   lexicon/lex_home.htm)
27
28 # `merge.c`
29
30 - examples
31 - for `merge-sort.c` later
32
33 # `insertion-sort.c`
34
35 - `for` + `while` version
36 - `for` + `for` version
```

```
1 add_executable(game-of-life game-of-life.c)
2 add_executable(game-of-life-chatgpt game-of-life-chatgpt.c
  )
3
4 add_executable(insertion-sort insertion-sort.c)
5 add_executable(insertion-sort-bsearch insertion-sort-
  bsearch.c)
6 add_executable(binary-insertion-sort binary-insertion-sort
  .c)
7
8 add_executable(merge merge.c)
```

```c
// Created by hfwei on 2024/10/16.

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define SIZE 6

const int board[SIZE][SIZE] = {
    { 0 },
    { 0, 1, 1, 0, 0, 0 },
    { 0, 1, 1, 0, 0, 0 },
    { 0, 0, 0, 1, 1, 0 },
    { 0, 0, 0, 1, 1, 0 },
    { 0 }
};

//const int board[SIZE][SIZE] = {
//    [1][1] = 1, [1][2] = 1,
//    [2][1] = 1, [2][2] = 1,
//    [3][3] = 1, [3][4] = 1,
//    [4][3] = 1, [4][4] = 1
//};

int main(void) {
  // TODO: play game-of-life

  return 0;
}
```

```c
// Created by hfwei on 2024/10/16.
// Code generated by ChatGPT.

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_LEN 10000
#define RANGE 100

int main(void) {
  int numbers[MAX_LEN] = { 0 };

  int size = 0;
  scanf("%d", &size);

  srand(time(NULL));
  for (int i = 0; i < size; i++) {
    numbers[i] = rand() % RANGE;
  }

  // print the original array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  // TODO: insertion sort

  // print the sorted array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  return 0;
}
```

```c
// Created by hfwei on 2024/10/16.
// Code generated by ChatGPT.

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <unistd.h>

// Define grid dimensions
#define ROWS 20
#define COLS 40

// Function to initialize the grid randomly
void initializeGrid(int grid[ROWS][COLS]) {
  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      grid[i][j] = rand() % 2;  // 0 (dead) or 1 (alive)
    }
  }
}

// Function to print the grid
void printGrid(int grid[ROWS][COLS]) {
  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      if (grid[i][j] == 1) {
        printf("#");  // Alive cell
      } else {
        printf(" ");  // Dead cell
      }
    }
    printf("\n");
  }
  printf("\n");
}

// Function to update the grid for the next generation
void updateGrid(int grid[ROWS][COLS]) {
  int newGrid[ROWS][COLS];

  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      int neighbors = 0;

```

```c
      // Count neighbors
      for (int x = -1; x <= 1; x++) {
        for (int y = -1; y <= 1; y++) {
          if (x == 0 && y == 0) { continue; }  // Skip the current cell
          int newX = i + x;
          int newY = j + y;

          if (newX >= 0 && newX < ROWS && newY >= 0 && newY < COLS) {
            neighbors += grid[newX][newY];
          }
        }
      }

      // Apply Game of Life rules
      if (grid[i][j] == 1) {
        newGrid[i][j] = (neighbors == 2 || neighbors == 3) ? 1 : 0;
      } else {
        newGrid[i][j] = (neighbors == 3) ? 1 : 0;
      }
    }
  }

  // Update the grid
  for (int i = 0; i < ROWS; i++) {
    for (int j = 0; j < COLS; j++) {
      grid[i][j] = newGrid[i][j];
    }
  }
}

int main(void) {
  int grid[ROWS][COLS];

  // Seed the random number generator with the current time
  srand(time(NULL));

  // Initialize the grid
  initializeGrid(grid);

  // Number of generations
```

```c
  int generations = 50;

  for (int gen = 0; gen < generations; gen++) {
    system("clear");  // Use "clear" on Unix-based systems (Linux, macOS)
    printf("Generation %d:\n", gen);
    printGrid(grid);
    updateGrid(grid);
    sleep(1);  // Sleep for 100ms
  }

  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_LEN 10000
#define RANGE 10

int main() {
  int numbers[MAX_LEN] = { 0 };

  int size = 0;
  scanf("%d", &size);

  srand(time(NULL));
  for (int i = 0; i < size; i++) {
    numbers[i] = rand() % RANGE;
  }

  // print the original array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  // TODO: binary insertion sort

  // Print the sorted array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_LEN 10000
#define RANGE 10

int main() {
  int numbers[MAX_LEN] = { 0 };

  int size = 0;
  scanf("%d", &size);

  srand(time(NULL));
  for (int i = 0; i < size; i++) {
    numbers[i] = rand() % RANGE;
  }

  // print the original array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  // TODO: insertion sort with binary search

  // Print the sorted array
  for (int i = 0; i < size; i++) {
    printf("%d ", numbers[i]);
  }
  printf("\n");

  return 0;
}
```