Tarea programada 2
Segundo Avance

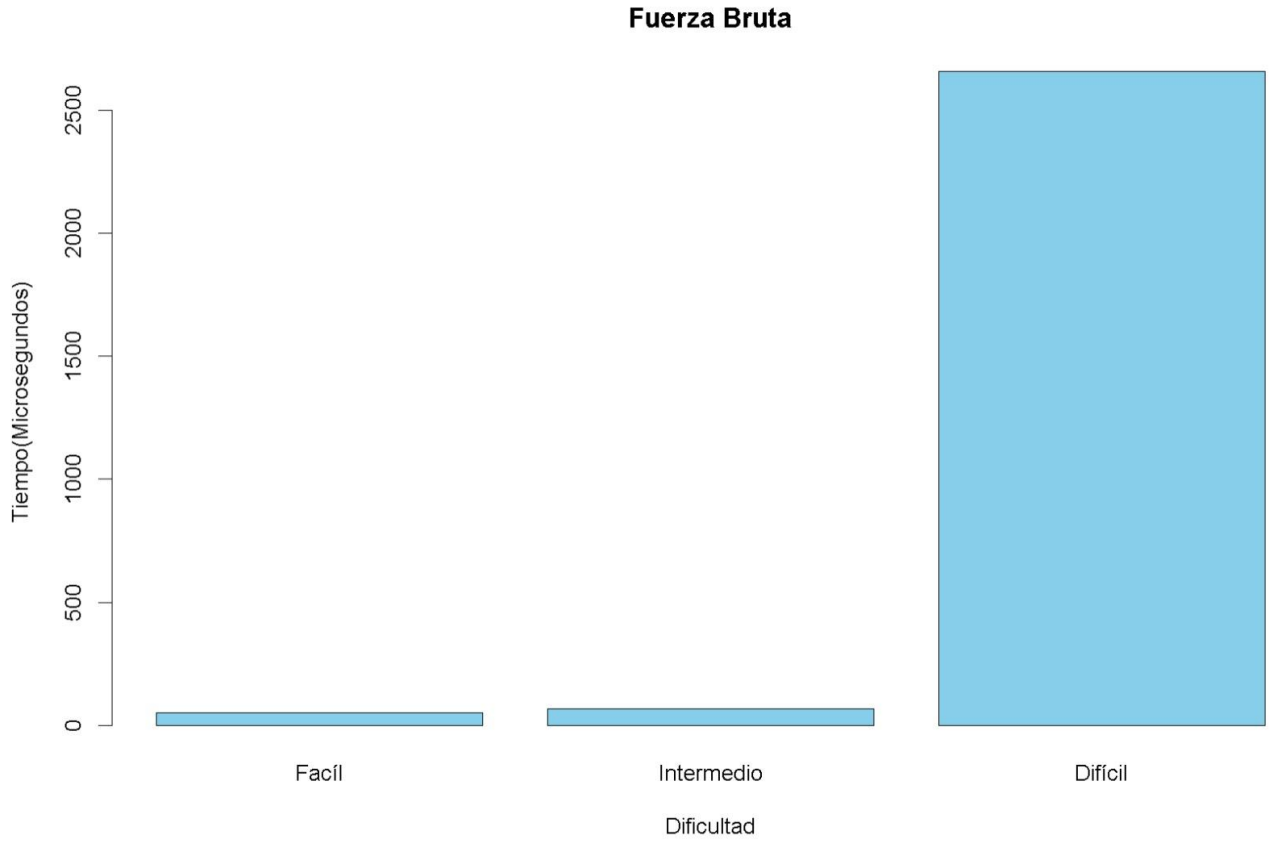
Nuggets

Fabian Vega Meza
Sebastian Venegas Brenes
Randy Steven Robles Vega

Fuerza bruta

```
40 bool Solver::solveBruteForce(Sudoku &sudoku, size_t row, size_t col) {
41   if (row == sudoku.size - 1 && col == sudoku.size) {
42     return true; // Tablero completado
43   }
44
45   if (col == sudoku.size) {
46     row++;
47     col = 0;
48   }
49
50   if (sudoku.matrix[row][col] != 0) {
51     return solveBruteForce(sudoku, row, col + 1); // Casilla ya llena, pasar a la siguiente
52   }
53
54   for (size_t num = 1; num <= sudoku.size; num++) {
55     if (sudoku.esValidoFuerzaBruta(row, col, num)) {
56       sudoku.matrix[row][col] = num;
57       if (solveBruteForce(sudoku, row, col + 1)) {
58         return true; // Intentar rellenar la siguiente casilla
59       }
60       sudoku.matrix[row][col] = 0; // Si no se puede, restaurar y probar otro número
61     }
62   }
63   return false; // No se puede resolver con estos números, retroceder
64 }
```

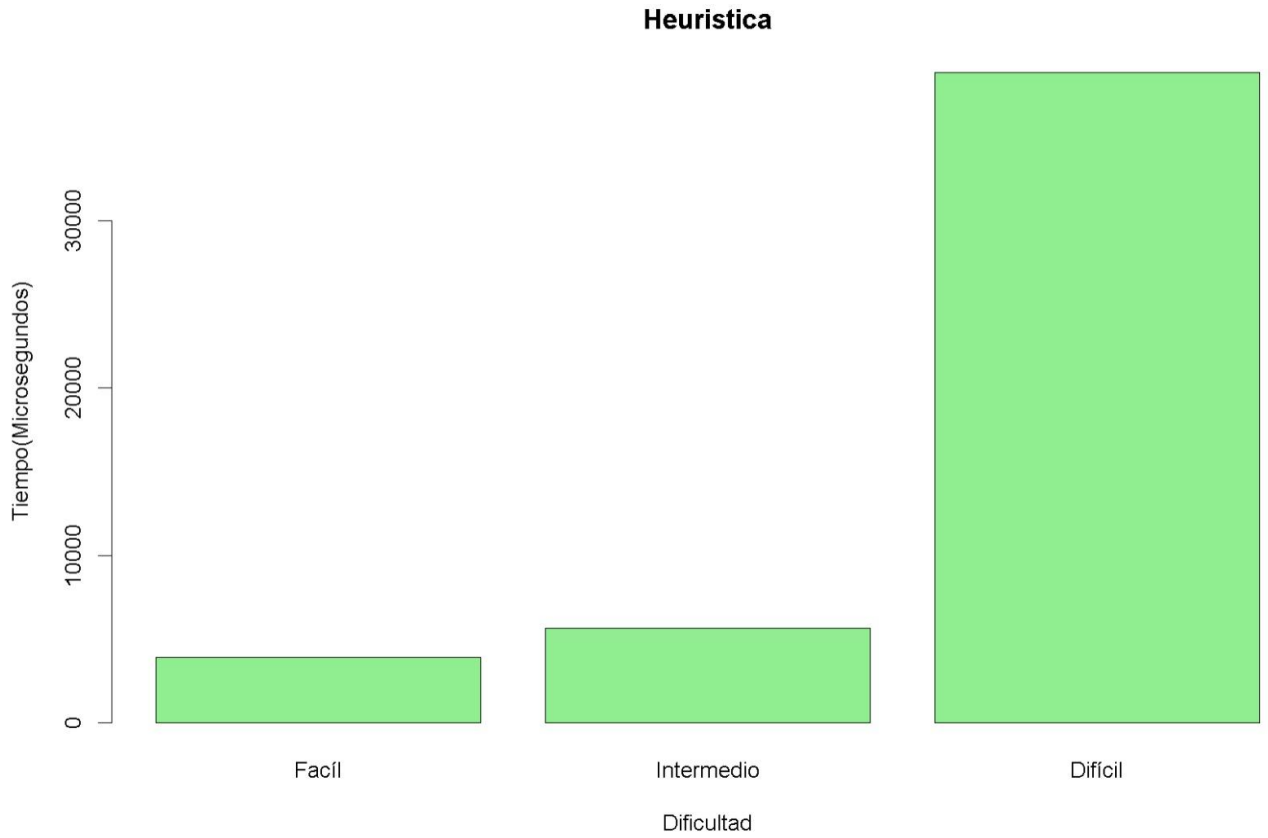
Fuerza bruta



Heurística

```
120  bool Solver::reduceCandidates(std::vector<std::vector<std::unordered_set<int>>>& candidates) {
121    bool isReduced = false;
122    for (size_t i = 0; i < this->sudoku.size; ++i) {
123      for (size_t j = 0; j < this->sudoku.size; ++j) {
124        if (candidates[i][j].size() == 1) {
125          int value = *(candidates[i][j].begin());
126          if (!eliminateCandidates(candidates, i, j, value)) {
127            return false;
128          }
129          isReduced = true;
130        }
131      }
132    }
133    return isReduced;
134  }
```

Heurística


Heuristica
Tiempo(Microsegundos)
0
10000
20000
30000
Facíl
Intermedio
Difícil
Dificultad

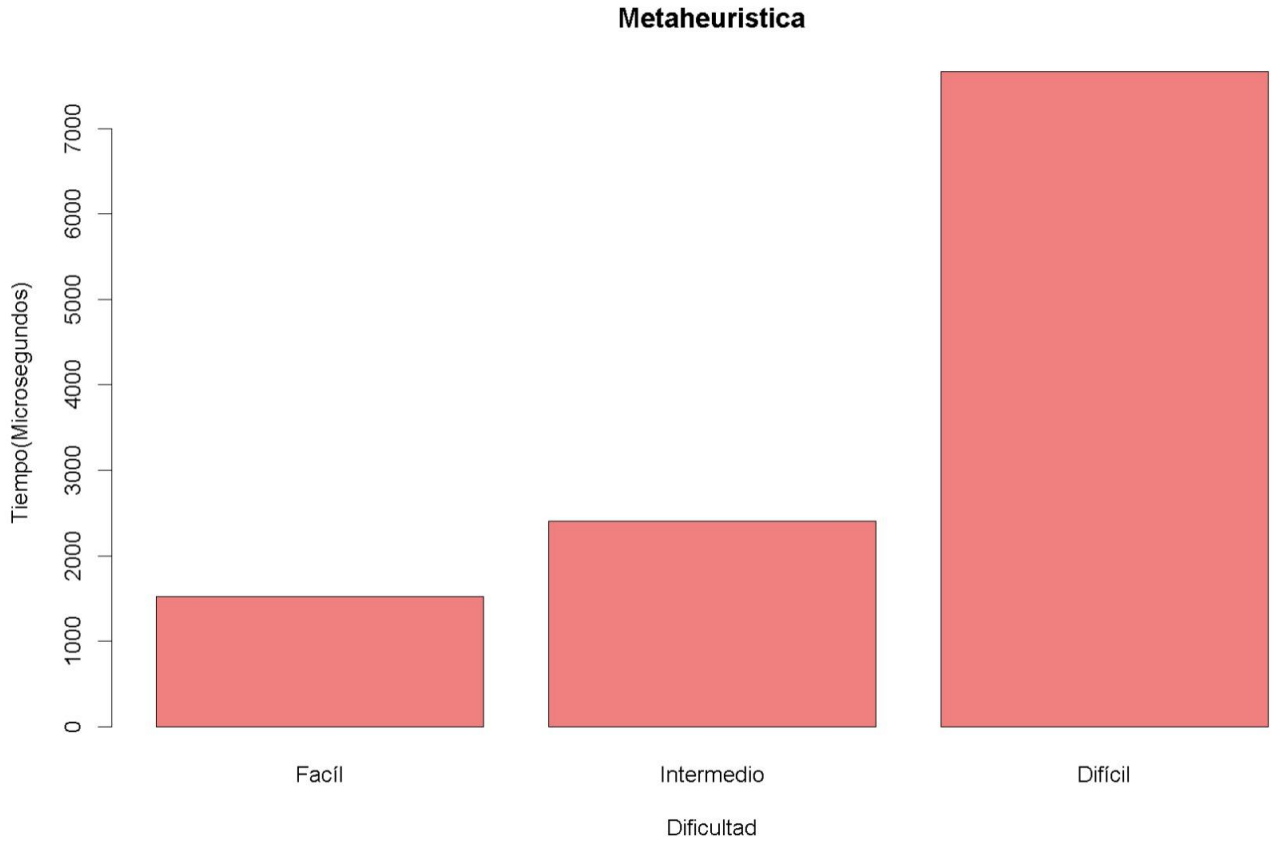
Meta-Heurística - Algoritmo genético

```
3   GeneticAlgorithm::GeneticAlgorithm(Sudoku &matriz, size_t tamano) : populationSize(tamano) {
4       std::cout << "Initializing Genetic Algorithm\n";
5       sudoku = matriz;
6       size_t vaciosNum = 0 ;
7       // Almacena todos los espacios vacios
8       for (size_t i = 0; i < tamano; i++) {
9           for (size_t j = 0; j < tamano; j++) {
10              if (sudoku.matrix[i][j] == 0) {
11                  vacios.push_back(std::make_pair(i, j));
12                  vaciosNum++;
13              }
14          }
15      }
16      population.resize(100);
17      // Initialize population
18      // Inicializa la poblacion con vectores de posibles soluciones
19      for (auto &cromosoma : population) {
20          cromosoma.solution.clear(); // Clear the existing vector content
21          Sudoku prueba = sudoku;
22          for (size_t i = 0; i < vaciosNum; ++i) {
23              //size_t randomNumber = prueba.randomGenerator(prueba.size);
24              size_t randomNumber = prueba.randomGenerator(tamano);
25              while (randomNumber> this->populationSize)
26              {
27                 randomNumber = sudoku.randomGenerator(sudoku.size);
28              }
29              for (size_t h = 1; h <= tamano; h++){
30                  if (prueba.esValido(vacios[i].first,vacios[i].second, h))
31                  {
32                      prueba.matrix[vacios[i].first][vacios[i].second] = h;
33                      randomNumber = h;
34                      break;
```

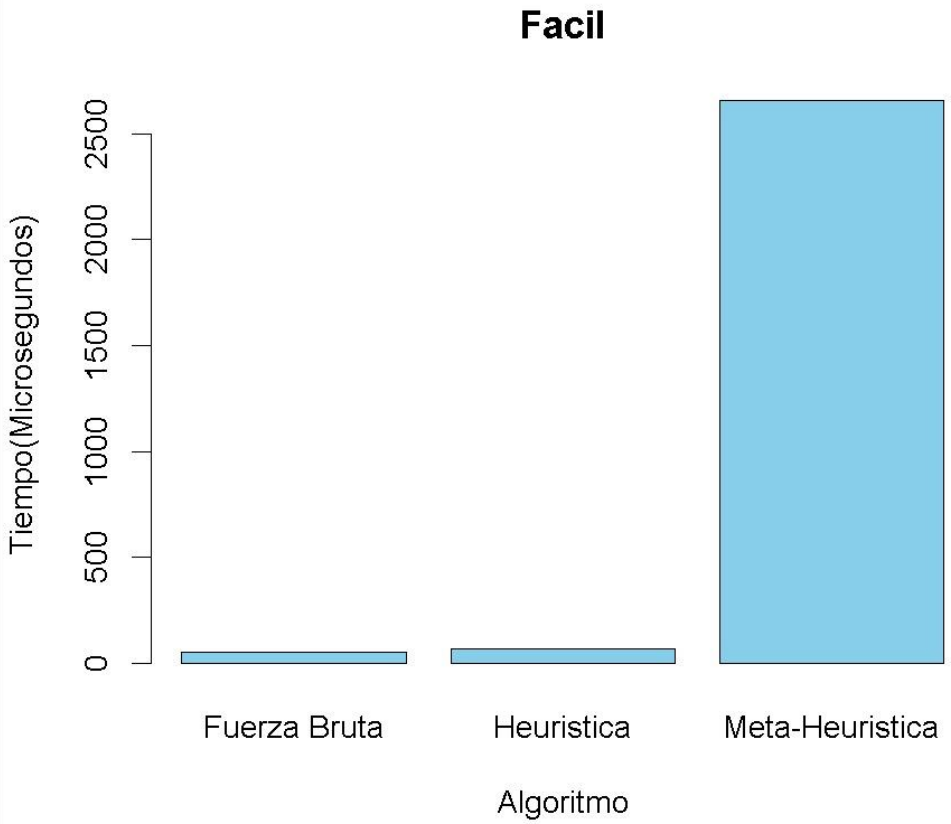
Meta-Heurística - Algoritmo genético

```
51      while (encontrado == false && contador < 20000) {
52          //std::cout << "Iteration: " << contador << "\n";
53          // Combina padres y crea hijos
54          mezcla();
55          // Les da ranking a los que no tienen
56          size_t a = rank();
57          if (a != 300) {
58              encontrado = true;
59              solucionFinal = a;
60          }
61          // Elimina los peores cromosomas
62          truncar();
63          // Muta las soluciones
64          mutate();
65          ++contador;
66      }
67      if (contador > 19998){
68          size_t lowestRank = 10000;
69          Sudoku mejor;
70
71          for (size_t index = 0; index < population.size(); ++index) {
72              Sudoku prueba = sudoku;
73
74              for (size_t i = 0; i < population[index].solution.size(); ++i) {
75                  size_t row = vacios[i].first;
76                  size_t col = vacios[i].second;
77                  prueba.matrix[row][col] = population[index].solution[i];
78              }
79
80              size_t duplicados = prueba.contarDuplicados();
81              population[index].ranking = duplicados;
82
```

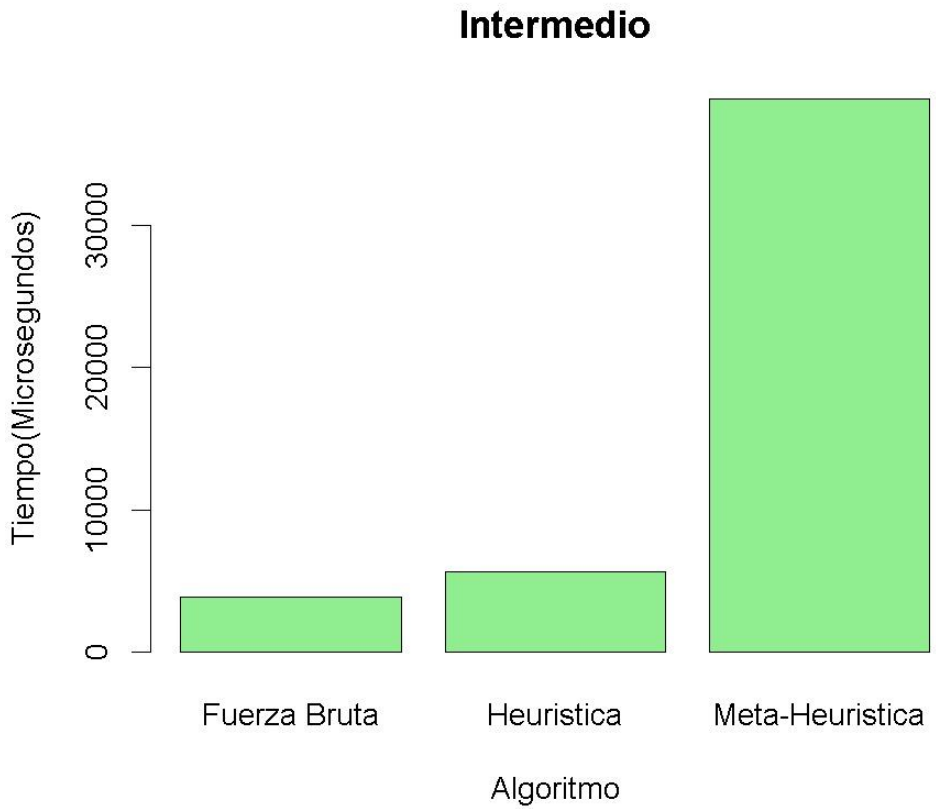
Meta-Heurística - Algoritmo genético



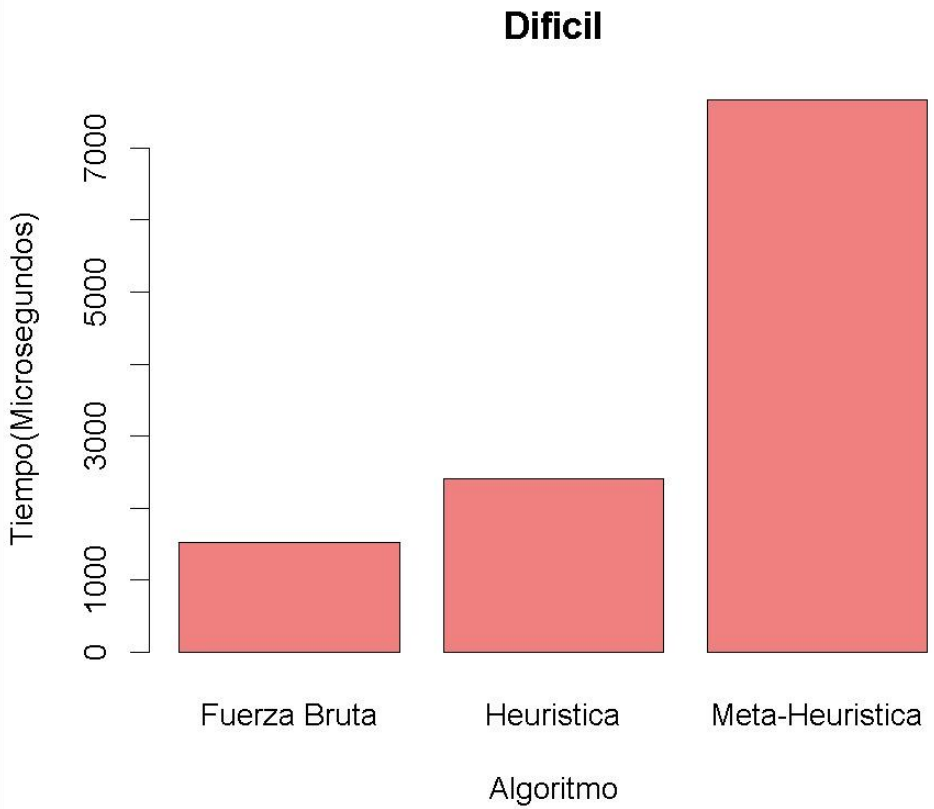
Dificultad


Facil
Tiempo(Microsegundos)
0
500
1000
1500
2000
2500
Fuerza Bruta
Heuristica
Meta-Heuristica
Algoritmo

Dificultad


Intermedio
Tiempo(Microsegundos)
0
10000
20000
30000
Fuerza Bruta
Heuristica
Meta-Heuristica
Algoritmo

Dificultad


Dificil
Tiempo(Microsegundos)
0
1000
3000
5000
7000
Fuerza Bruta
Heuristica
Meta-Heuristica
Algoritmo

Gracias