The Merkle-Hellman Cryptosystem

ALGORITMOS E ESTRUTURAS DE DADOS

Prof. Tomás Silva

André Butuc - 103530 – 60%

Gonçalo Silva - 103668 - 40%

Índice

Introdução

The Merkle-Hellman Cryptosystem

Este trabalho prático desenvolvido no âmbito da cadeira Algoritmos e Estruturas de Dados tem como objetivo a implementação e o estudo de inúmeras técnicas de resolução do sistema de criptografia de Merkle e Hellman.

O criptosistema de Merkle-Hellman é aquilo que se chama um sistema de chave pública, ou seja, são usadas duas chaves, uma delas sendo pública usada para a criptografia e uma privada usada na descriptografia.

Este sistema é baseado no problema de somas de um subconjunto, sendo este um caso do mais complexo knapsack problem.

Seja dado um conjunto de números inteiros e um número s, existe um subconjunto em que a soma dos seus elementos dê s? Este é o nosso problema geral: verificar se existe este subconjunto, identificá-lo e traduzi-lo numa sequência binária que denota o peso que cada número inteiro do conjunto inicial tem na soma.

Tendo como dados iniciais, n (o número de elementos do conjunto inicial), p (o vetor que contém os números inteiros do conjunto inicial) e desired_sum (a soma que desejamos encontrar) iremos percorrer os dois ficheiros com os problemas: “103530.h” e “103668.h”.

Os ficheiros “.h” estão construídos de forma a terem um intervalo de valores de “n=10” a “n=57”, onde em cada “n” há um vetor p com “n” números inteiros e em cada “n” temos “20 desired_sum’s” às quais teremos que encontrar a sequência binária solução.

Material adaptado

- Código relacionado com o algoritmo de ordenação “Quicksort” (swap, partition, quicksort) foi adaptado do site “GeeksforGeeks”, presente na seguinte hiperligação: https://www.geeksforgeeks.org/quick-sort/

- Código relacionado com o algoritmo de ordenação “Merge Sort” (insertion_sort e merge_sort) foi adaptado dos slides teóricos de AED do professor Tomás Silva.

Abordagem ao Problema

Na tentativa de resolução deste problema foram testadas diversas técnicas computacionais com o intuito de desenvolver algoritmos capazes de resolver o problema em causa, avaliando a eficiência dos mesmos.

Foram desenvolvidos 7 algoritmos:

- Iterative Brute Force ("iterative_brute_force")

- Recursive Brute Force ("recursive_brute_force")

- Clever Recursive Brute Force ("clever_recursive_brute_force")

- Horowitz and Sahni ("horowitz_sahni")
    - QuickSort version
    - MergeSort version

- Schroeppel and Shamir ("shroeppel_shamir")
    - QuickSort version
    - MergeSort version

Iterative Brute Force

Complexidade Computacional: O$(n2^n)$

A implementação "Iterative Brute Force" consiste em percorrer iterativamente todas as possibilidades de combinação dos "n" elementos do conjunto "p", logo a complexidade computacional desta abordagem reduz-se a

$$O(2^n).$$

Em cada iteração é calculada a "test_soma", sendo esta incrementada por "p[bit]" dentro de um loop que percorre o valor de "bit" de 0 a n-1, que corresponde ao índice de cada elemento de "p". No entanto, esta incrementação só ocorre se o valor de "bit" pertence à combinação binária de "comb" (podendo esta propriedade ser observada se a posição em que o valor bit a "um" se encontra em "bit" corresponde à posição em que um dos bits a "um" se encontra em "comb").

As operações bitwise em C, facilitam bastante a extensão de operandos que não têm a mesma dimensão, permitindo fazer cálculos em base binária que se enquadram com a natureza do problema.

Esta última abordagem contribui com um acréscimo de complexidade de O(n) em cada iteração de 2^n possibilidades de combinações binárias.

Totalizando as complexidades computacionais das abordagens internas ao algoritmo teremos de facto como complexidade final

$$O(n2^n)$$

Gráficos Iterative Brute Force

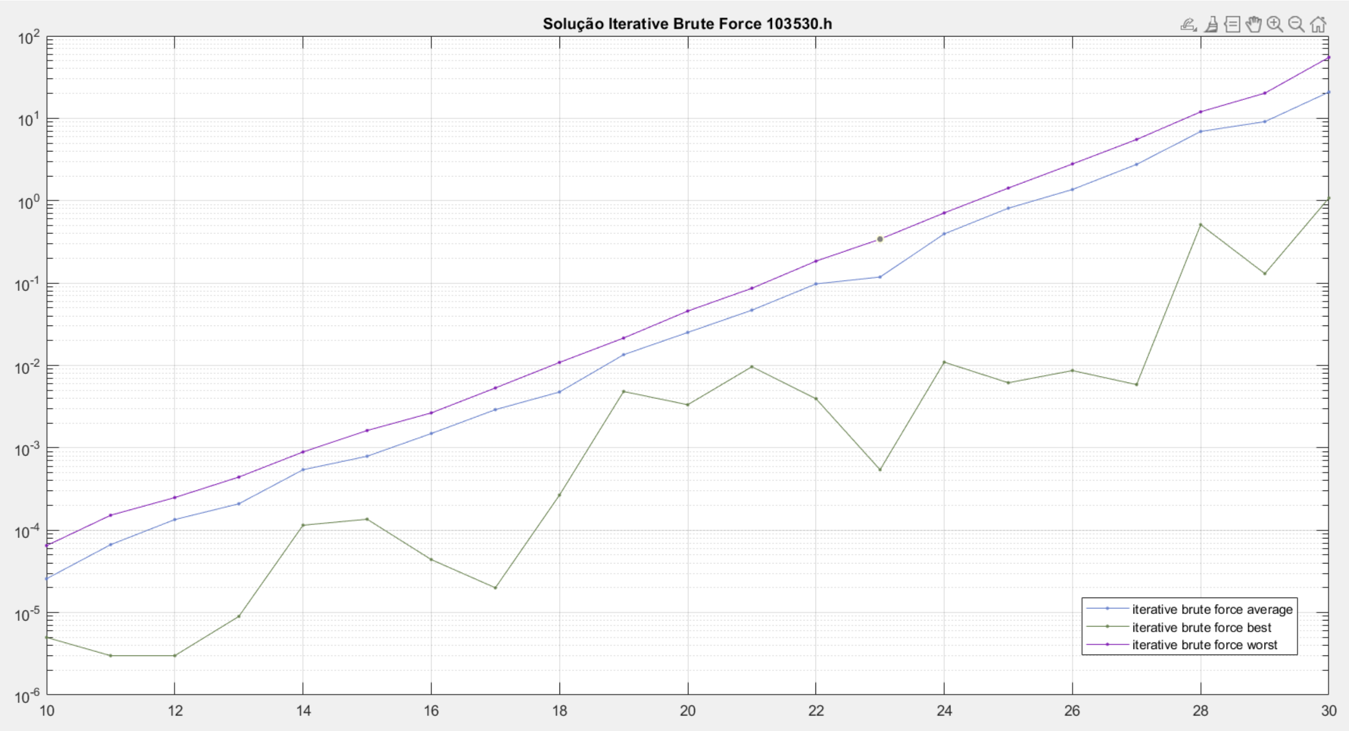

Figura 1 - Iterative Brute Force 103530.h

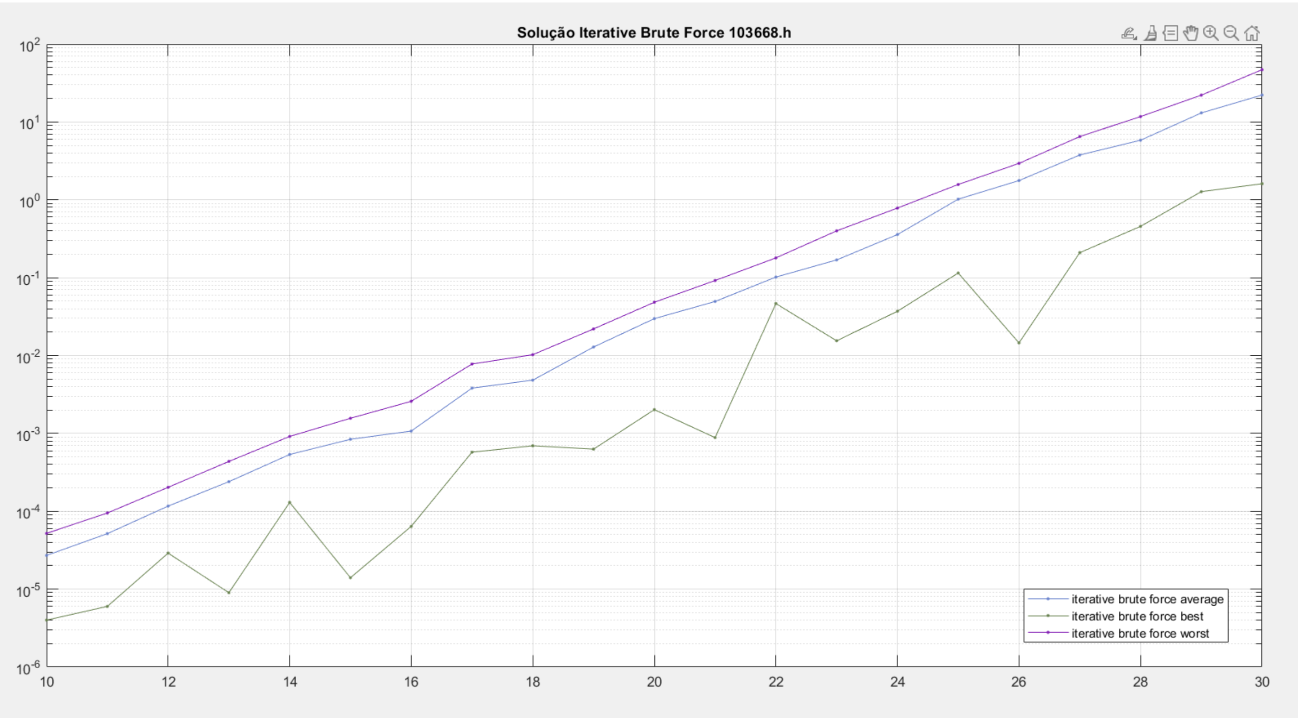

Figura 2 - Iterative Brute Force 103668.h

Recursive Brute Force

Complexidade computacional: $O(2^n)$

O algoritmo recursive_brute_force, como o nome indica, encontra a solução recursivamente. Em cada valor de "p" há um "branching": o bit do valor de "p" fica a 1 ou fica a 0. Em cada ramo, a soma parcial ("partial_soma") é incrementada (no ramo do bit igual a 1) ou mantém-se igual (no ramo do bit igual a 0). De seguida é incrementado o índice do array "p" em cada ramo e é chamada de novo a função "recursive_brute_force" em cada ramo. Esta abordagem permite criar uma "árvore" com todas as possibilidades de combinações binárias que mapeiam a "presença" ou "não presença" do elemento a ser avaliado de "p" na soma parcial. Devido ao facto da função ser chamada duas vezes de forma recursiva e pelo facto de no pior caso esta o ser chamada "n" vezes, a complexidade computacional do algoritmo é totalizado a $O(2^n)$.

As condições de paragem da função recursiva são:

- if(partial_soma == desired_soma), que significa que foi encontrada a solução. O retorno do valor 1 irá desencadear o desenvolvimento das sucessivas chamadas recursivas prévias da função. Sendo que o bit da "folha" que desencadeou a terminação da função com sucesso irá ser armazenado no array "b" e assim sucessivamente até à raiz;
- if(current_index == n), que significa, caso verdadeiro, que a recursividade chegou ao fim do array "p" e como já não há mais elementos ou combinações a somar e como ainda não foi encontrada a solução, retorna o valor 0 que por sua vez indica que não foi encontrada a solução-soma.

Gráficos Recursive Brute Force

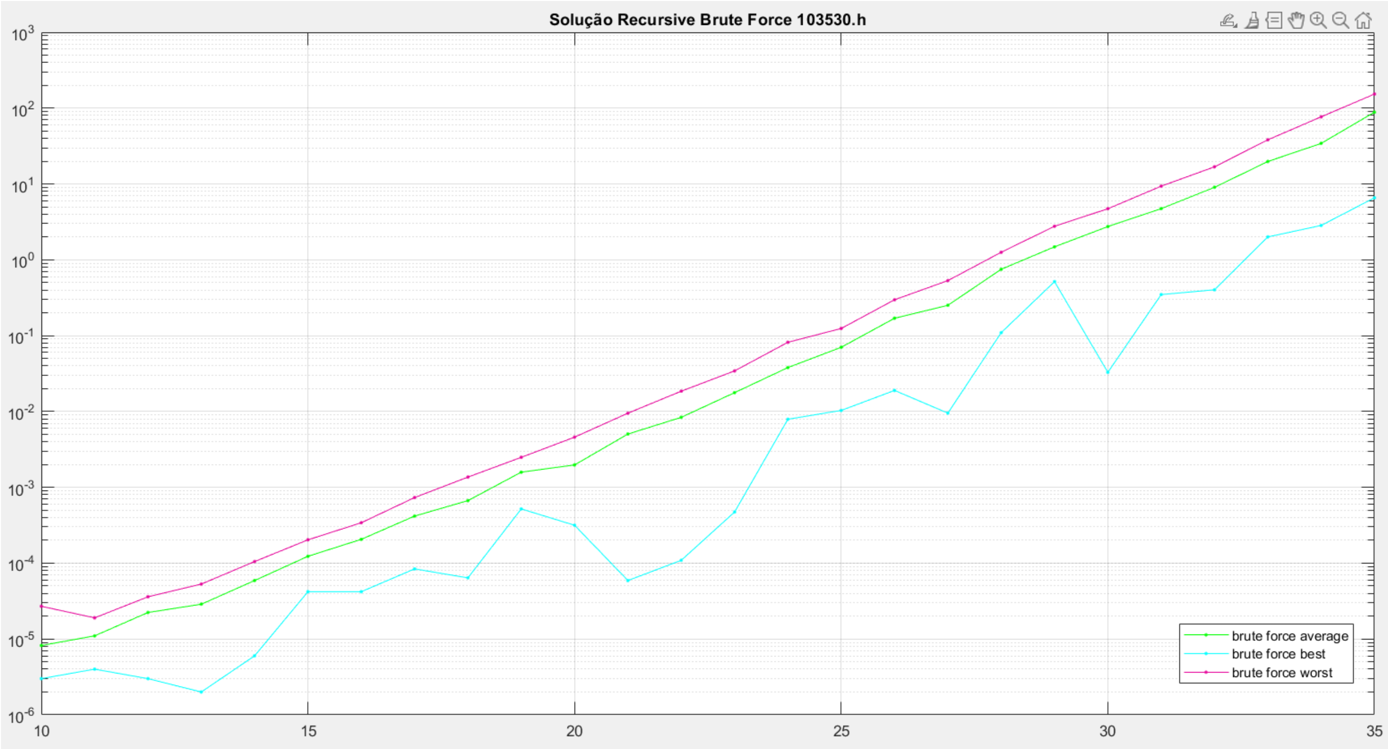

Figura 3 - Recursive Brute Force 103530.h

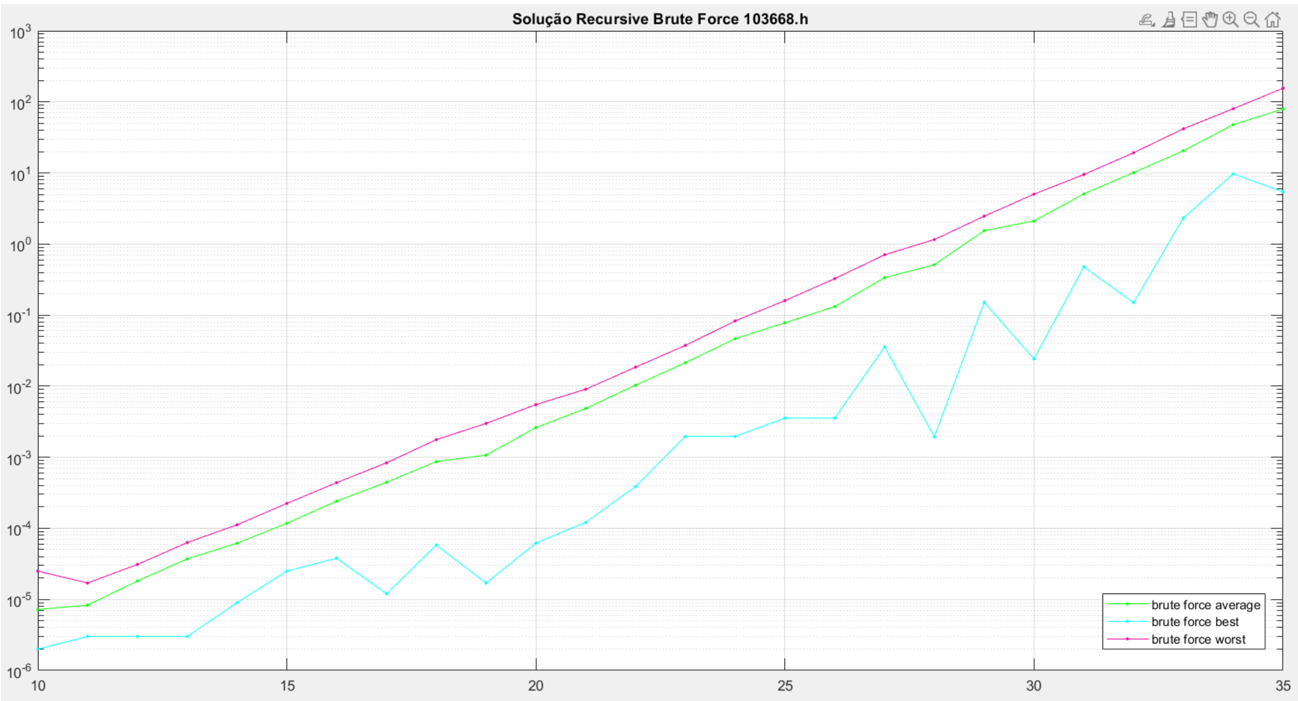

Figura 4 - Recursive Brute Force 103668.h

Clever Recursive Brute Force

Complexidade computacional: O(2^n)

O algoritmo clever_recursive_brute_force, é semelhante ao algoritmo "recursive_brute_force", no entanto, apresenta mais dois parâmetros de entrada que servem para calcular uma soma parcial a partir dos últimos índices para os primeiros. E para além de apresentar as condições de paragem do algoritmo "recursive_brute_force", também apresenta mais duas condições de paragem.

Este algoritmo não difere em complexidade computacional do anterior mencionado, no entanto, difere na capacidade de conseguir parar a sua execução quando, após observar provas concretas, interpreta que não será possível encontrar uma solução ao problema.

Os dois parâmetros novos serão utilizados para interpretar se é de facto possível chegar à solução com os valores dispostos ou a soma será sempre inferior à soma desejada. O parâmetro "current_last_index" será decrementado em cada chamada da função recursiva e o parâmetro "bigger_soma" irá guardar a soma dos valores sucessivos de "p[current_last_index]". Quando o "current_index" é igualado a "current_last_index + 1" significa que já passámos a metade de "p", logo se a soma de "bigger_soma" com "partial_soma" continuar a ser inferior a "desired_soma" então não é possível alcançar a soma desejada com os números presentes no vetor "p".

A condição de paragem restante é "if(partial_soma > desired_soma)". Pelo facto do array "p" se encontrar ordenado e pelo facto de apresentar valores inteiros positivos, sabendo que também a "partial_soma" é sempre incrementada e não decrementada, então a partir do momento em que "partial_soma" é superior a "desired_soma", não há como "voltar atrás" e assim é possível afirmar que não é encontrada a solução.

Gráficos Clever Recursive Brute Force

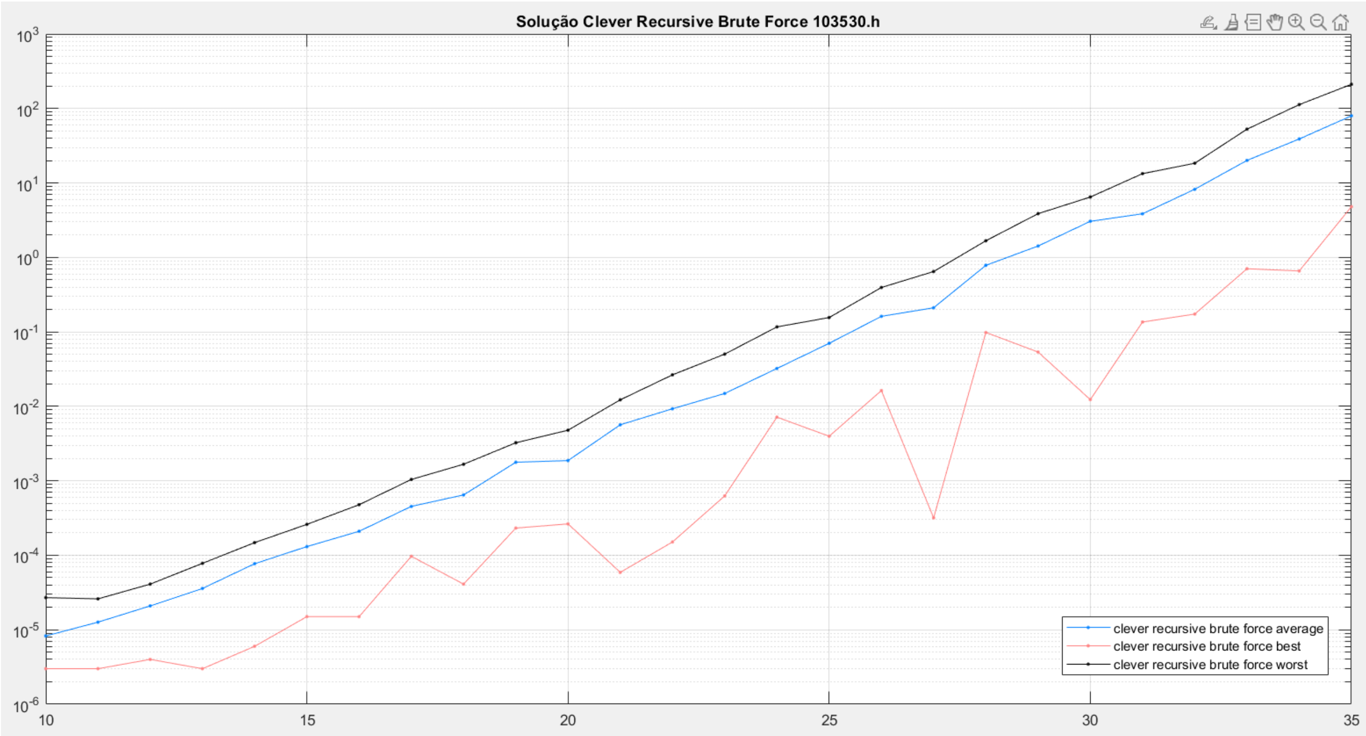

Figura 5 - Clever Recursive Brute Force 103530.h

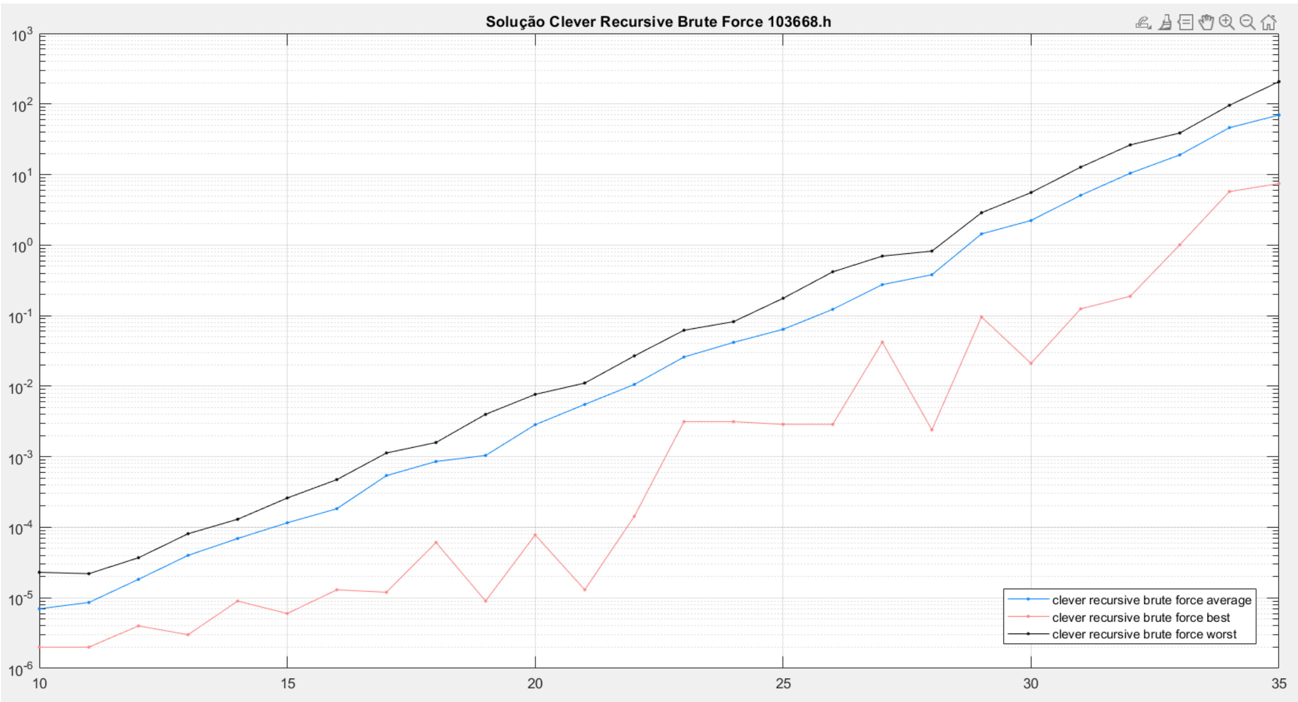

Figura 6 - Clever Recursive Brute Force 103668.h

Horowitz-Sahni

Complexidade computacional: $O(2^{\frac{n}{2}}\frac{n}{2})$

O algoritmo de Horowitz-Sahni levou-nos a criar uma struct ("mascara_struct_t") que guardasse e mapeasse de melhor forma as somas parciais e as máscaras binárias associadas a elas.

Esta abordagem utiliza a técnica meet-in-the-middle.

Inicialmente o array "p" é dividido de forma aproximadamente igual em dois arrays, "p1" e "p2". Pelo facto do array "p" se encontrar ordenado, "p1" terá os valores menores de "p" e "p2" os valores maiores de "p". De seguida são calculadas todas as somas internas possíveis de "p1" e de "p2", estas somas são calculadas através da função "soma_all_subsets", que consiste na adaptação das funções recursivas anteriormente referidas, para que estas estejam em concordância com a struct "mascara_struct_t", não existindo no entanto uma condição de paragem à exceção de "if(current_index == n)" (pois queremos todas as combinações possíveis e não a combinação-solução como anteriormente).

Em cada chamada recursiva é feita uma operação bitwise OR ("|") do valor de "mascara" com o valor de 1 bit deslocado para a esquerda de "current_index" bits, caso o bit do elemento de "p" seja "um".

Por exemplo, vamos supor que temos o valor inicial da "mascara" 0 e que o elemento que estamos a avaliar tem indice de 5 no vetor "p" e vamos colocar este elemento com bit 1, ou seja, participa na soma parcial, então "mascara | (1<<5)" = "mascara | 100000" = " 000000 | 100000" = "100000" (o valor da mascara é extendido com bits "zeros" mais significativos até ser igual à dimensão com a qual estamos a fazer a operação bitwise OR, o mesmo acontece inversamente), esta operação e propriedade lógica permite-nos "armazenar" o valor do bit do elemento (que pode ser 0 ou 1) na máscara.

No entanto, é ainda de realçar, dentro da função "soma_all_subsets" que a forma de armazenamento dos dados nas estruturas no array "somas", array que contém todas as struct "mascara_struct_t" inicializadas, teve que ser feita da seguinte forma:

Para não haver conflito quando quisermos guardar elementos no array "somas", quando há o branching do ramo que iguala o bit do elemento a ser avaliado a 0 (não participa na soma parcial) é incrementado o índice dos índices ímpares e o ramo que iguala o bit do elemento a ser avaliado a 1 (participa na soma parcial) é incrementado o índice dos índices pares. Implementámos esta abordagem pois observámos que se incrementássemos de forma sucessiva o valor de "index", por exemplo com "index++" iria existir "over-write" de dados já presentes no array "somas".

Após termos os arrays "a" e "c" com as "mascara_struct_t" atualizadas em concordância com todas as somas possíveis de cada array "p1" e "p2", sendo que "a" tem as somas de "p1" e "c" as somas de "p2", é necessário ordenar os arrays por ordem crescente. Para este efeito elaboramos duas versões, uma em que a ordenação era feita através do algoritmo de ordenação "QuickSort" e uma em que a ordenação era feita através do algoritmo de ordenação "MergeSort".

O último passo é efetivamente o algoritmo de Horowitz-Sahni, onde percorremos num while loop os arrays "a" e "c", "a" a começar do primeiro elemento e "c" a começar do último até um dos índices de "a" ou de "c" "transbordar" ou até encontrarmos a soma desejada.

Em cada iteração é avaliado se a soma parcial é superior à soma desejada, caso o seja, então o índice de "c" é decrementado, caso contrário, se a soma parcial for inferior à soma desejada então é incrementado o índice de "a".

Quando for atingida a soma desejada, a solução consistirá na "junção" das máscaras das somas parciais, a máscara da soma parcial de "c" terá o peso dos bits mais significativos e a máscara da soma parcial de "a" terá o peso dos bits menos significativos.

Relativamente à complexidade computacional:

- "soma_all_subsets", por si só tem uma complexidade computacional de O(2^{n}), no entanto, no enquadramento do problema, irá contribuir com

$$O(2^{\frac{n}{2}+1})$$

- o algoritmo de ordenação "QuickSort", no caso médio tem complexidade de O($nlog(n)$), mas enquadrando-o ao nosso problema terá O($2^{\frac{n}{2}+1}\log_2(2^{\frac{n}{2}+1})$), que simplificado ficará

$$O(2^{\frac{n}{2}+1}(\frac{n}{2} + 1))$$

- o algoritmo de ordenação "MergeSort", no pior caso tem complexidade de O($nlog(n)$), mas enquadrando-o ao nosso problema terá O($2^{\frac{n}{2}+1}\log_2(2^{\frac{n}{2}+1})$), que simplificado ficará O($2^{\frac{n}{2}+1}(\frac{n}{2} + 1)$)
- o while-loop nos piores dos casos itera "length_a"+"length_c" vezes, logo tem como complexidade computacional O($2^{\frac{n}{2}} + 2^{\frac{n}{2}}$) =

$$O(2^{\frac{n}{2}+1})$$

A complexidade do algoritmo, tanto na versão de QuickSort como na versão MergeSort, assume a componente de complexidade de maior peso, logo o algoritmo terá

$$O(2^{\frac{n}{2}}\frac{n}{2}).$$

Gráficos Horowitz-Sahni (103530.h)

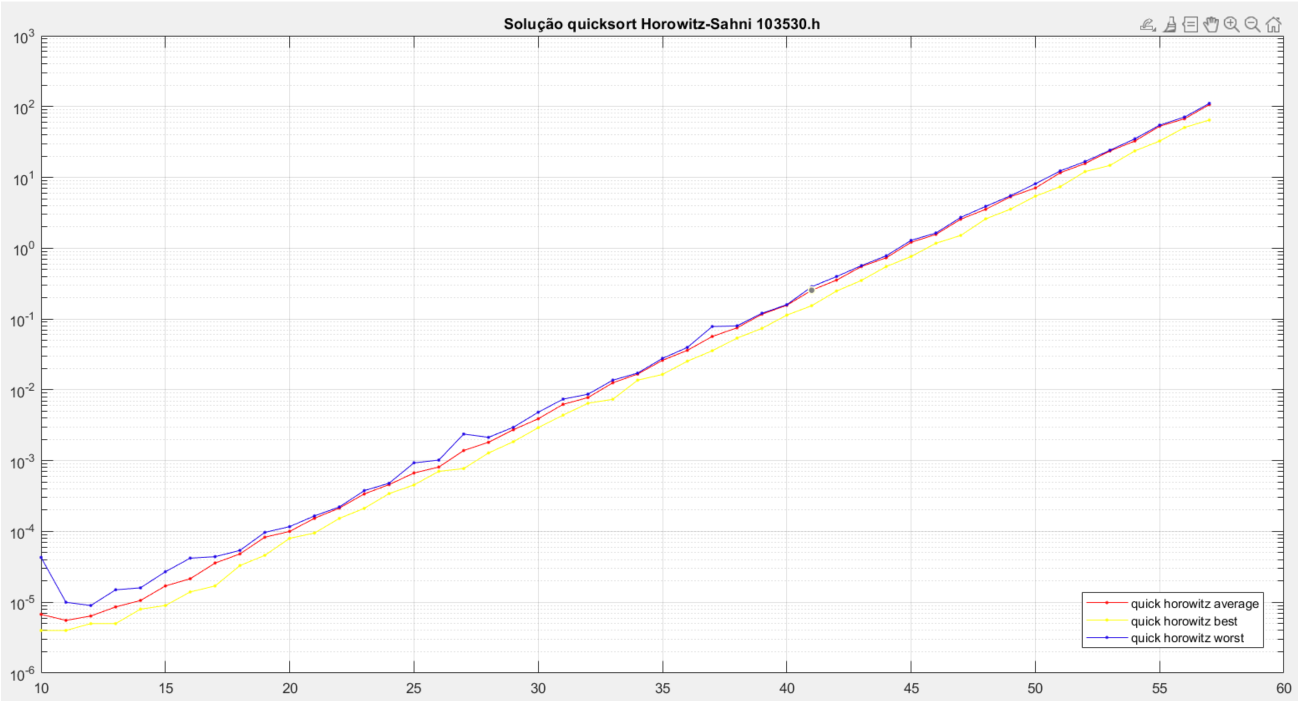

Figura 7 - Horowitz-Sahni QuickSort 103530.h

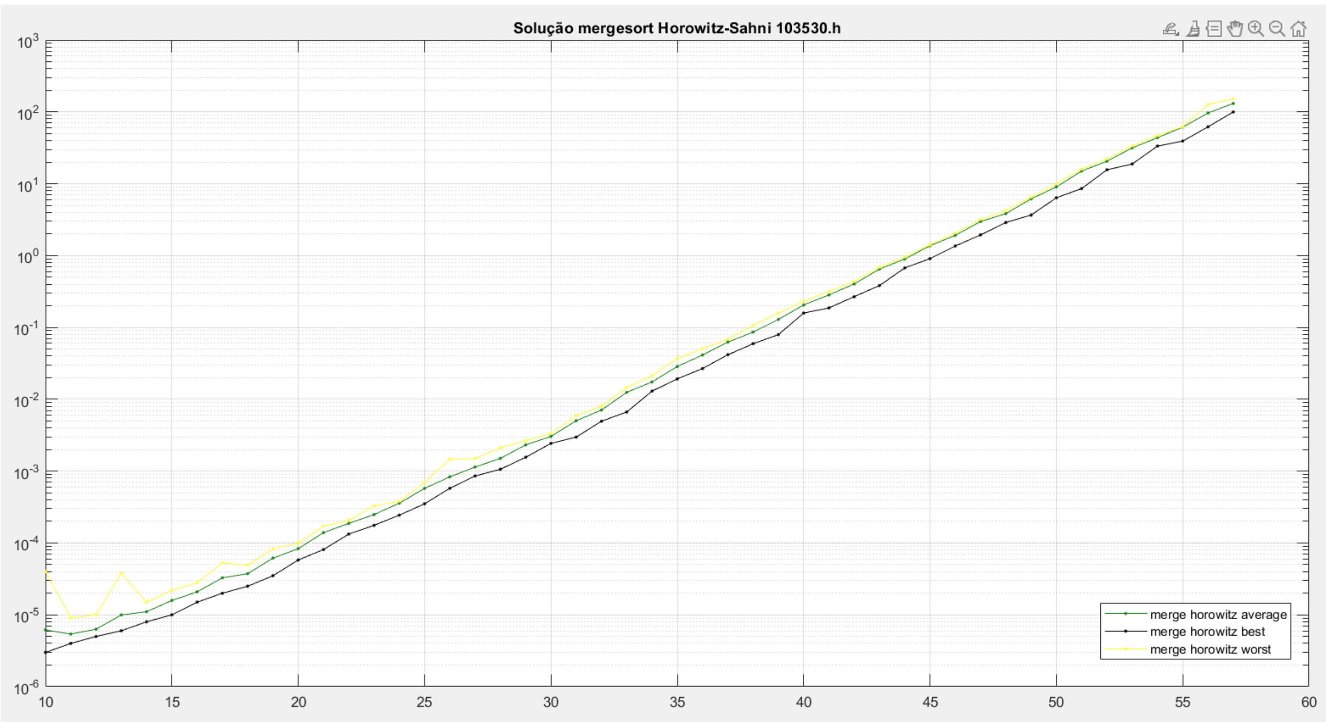

Figura 8 - Horowitz-Sahni MergeSort 103530.h

Gráficos Horowitz-Sahni (103668.h)

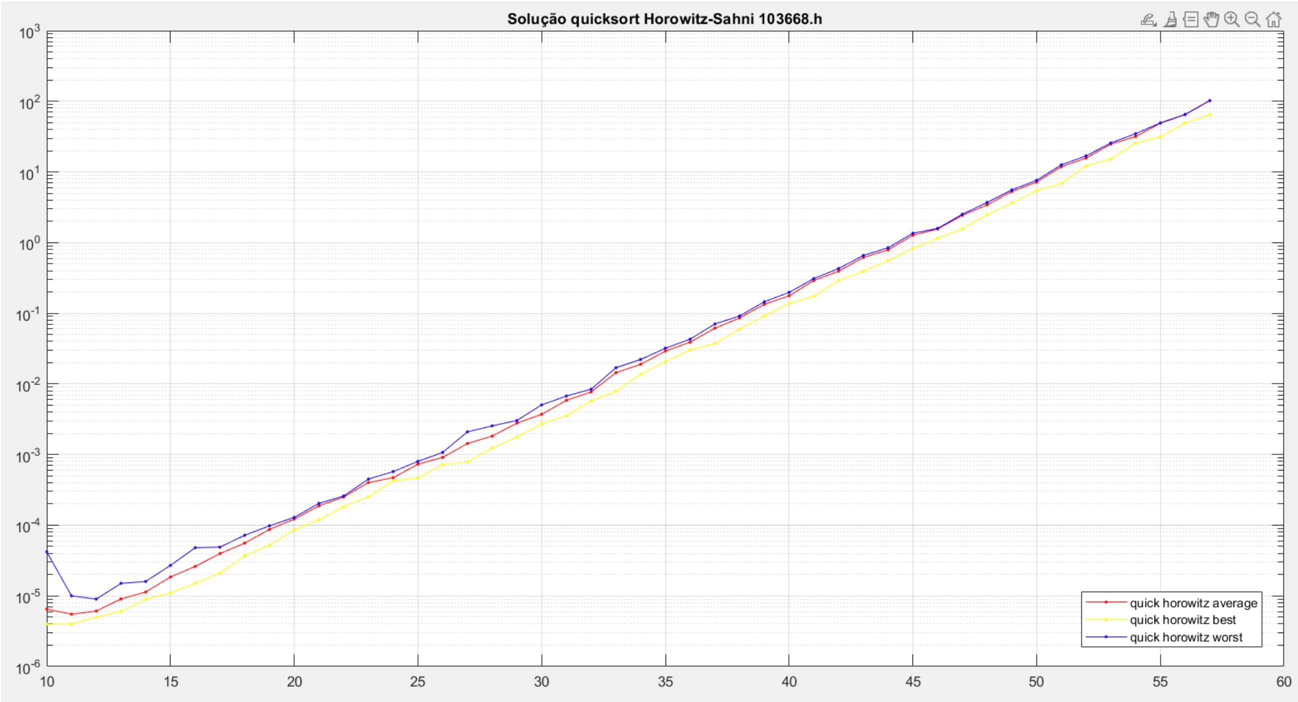

Figura 9 - Horowitz-Sahni QuickSort 103668.h

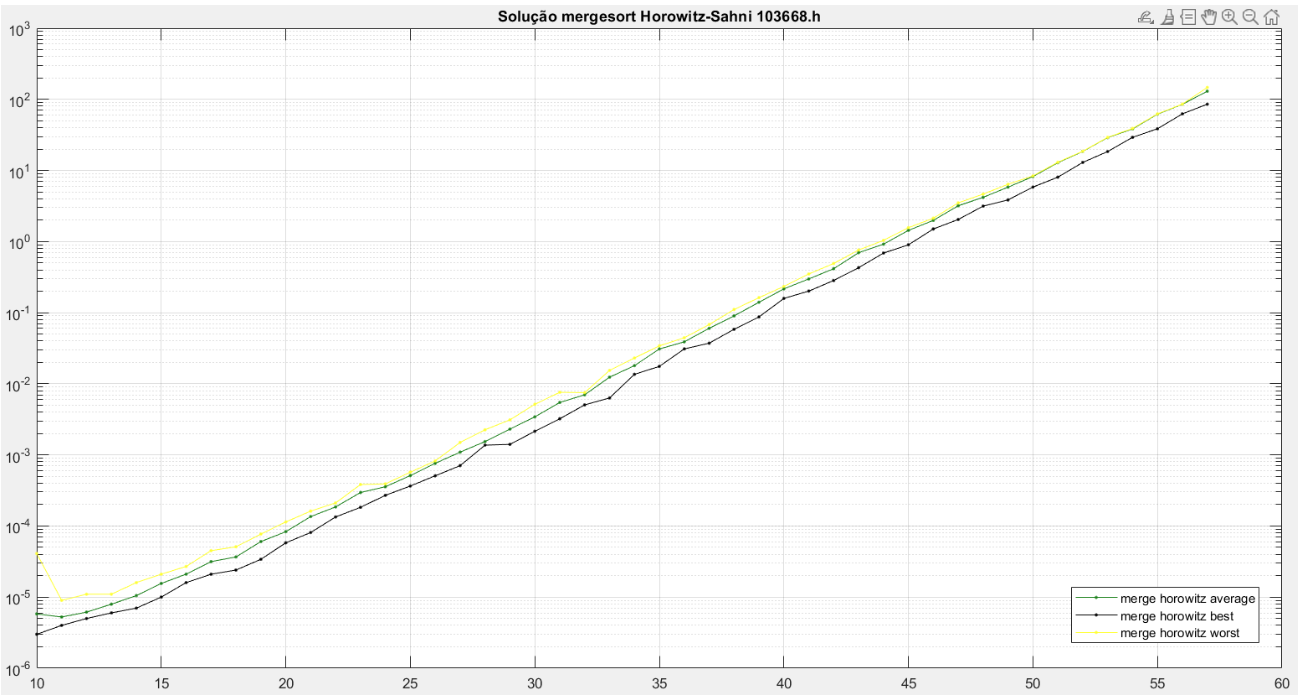

Figura 10 - Horowitz-Sahni MergeSort 103668.h

Schroeppel-Shamir

Complexidade computacional: $O(2^{\frac{n}{2}} \frac{n}{2})$

O algoritmo de Schroeppel-Shamir é um algoritmo bastante semelhante ao algoritmo de Horowitz-Sahni. Aplica também a técnica meet-in-the-middle, no entanto é uma abordagem que requer menos memória, pois a soma parcial é calculada através da utilização de uma min-heap e de uma max-heap.

Nesta abordagem o array "p" é dividido de forma aproximadamente igual em quatro arrays, "p1", "p2", "p3" e "p4". De seguida são calculadas todas as somas possíveis internas de cada cada array e ordenadas através do QuickSort (ou do MergeSort).

Foi necessário construir uma struct de elementos que vamos guardar nas heaps, sendo que este elemento tem uma "mascara" (combinação binária), "soma", "indice1" e "indice2", este últimos atributos irão mapear os elementos dos array "a1" e "a2" (para a min-heap) e "b1" e "b2" (para a max-heap), respetivamente.

A max-heap e min-heap são inicializados com os valores de "b1" e "a1", respectivamente, somados com o último valor de "b2" para a max-heap e com o primeiro valor de "a2" para a min-heap. O mesmo acontece para as máscaras com o procedimento mencionado no algoritmo de "Horowitz-Sahni".

Para manipular as heaps, foram utilizados os métodos "push" e "pop". A função "push" insere um elemento na heap e reordena-a consoante o seu tipo, se o tipo for "1", então corresponde à min-heap, se o seu tipo for "2", então corresponde à max-heap. A função "pop" remove um elemento da heap e retorna-o, re-ordenando de seguida a heap consoante o seu tipo como descrito na função "push".

No algoritmo de "Schroeppel-Shamir", ao contrário do que acontece no algoritmo de "Horowitz-Sahni", em cada iteração é calculada a soma dos elementos que se encontram no topo das heaps. Caso a soma seja inferior à desejada, é retirado da min-heap o elemento que se encontra no seu topo, é incrementado o atributo “indice2”, caso o valor de “indice2” não seja superior ao tamanho de “a2”, então é criada uma nova “heapStruct_t” com os valores atualizados do elemento retirado e é voltada a ser colocada na min-heap. A condição de ser inferior ao tamanho de “a2” deve-se ao facto de, caso esta seja verdadeira, já foram “testadas” todas as combinações do valor “indice1” do vetor “a1” com os valores de “a2”, logo não deve permanecer na min-heap.

O mesmo acontece na max-heap, mas agora com os arrays “b1” e “b2” e o decremento do valor de “indice2”.

Quando é encontrada a soma-solução, então a solução é guardada tal como foi guardada no algoritmo de “Horowitz-Sahni”.

Relativamente à complexidade computacional:

- “soma_all_subsets”, por si só tem uma complexidade computacional de $O(2^{n})$, no entanto, no enquadramento do problema, irá contribuir com

$$O(2^{\frac{n}{4}+2})$$

- o algoritmo de ordenação “QuickSort”, no caso médio tem complexidade de $O(nlog(n))$, mas enquadrando-o ao nosso problema terá $O(2^{\frac{n}{4}+2}\log_2(2^{\frac{n}{4}+2}))$, que simplificado ficará

$$O(2^{\frac{n}{4}+2}(\frac{n}{4}+2))$$

- o algoritmo de ordenação “MergeSort”, no pior caso tem complexidade de $O(nlog(n))$, mas enquadrando-o ao nosso problema terá $O(2^{\frac{n}{4}+2}\log_2(2^{\frac{n}{4}+2}))$, que simplificado ficará

$$O(2^{\frac{n}{4}+2}(\frac{n}{4}+2))$$

- o while-loop nos piores dos casos itera até ser falsa a sua condição de while loop, o que significa que uma das heaps ficou vazia, o que nos dá uma complexidade computacional de

$$O(2^{\frac{n}{2}+2}\frac{n}{2})$$

A complexidade do algoritmo, tanto na versão de QuickSort como na versão MergeSort, assume a componente de complexidade de maior peso, logo o algoritmo terá $O(2^{\frac{n}{2}}\frac{n}{2})$.

Gráficos Schroeppel-Shamir (103530.h)

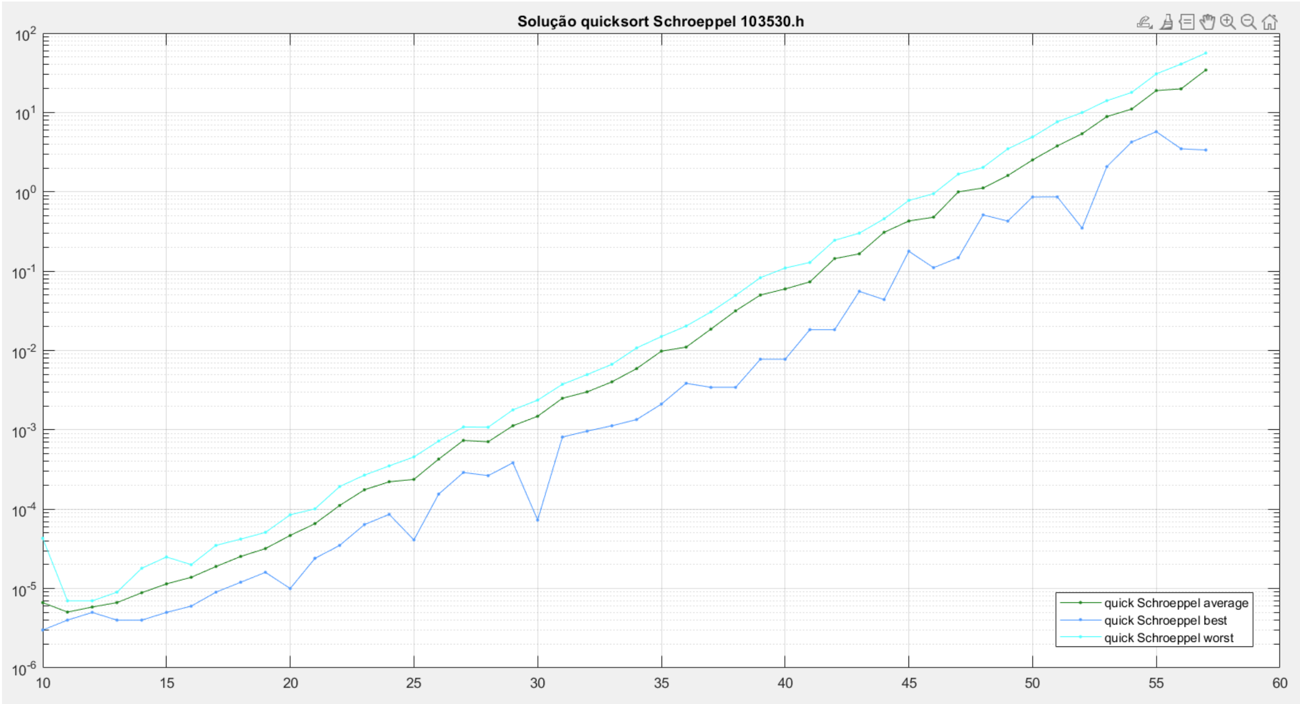

Figura 11 - Schroeppel-Shamir QuickSort 103530.h

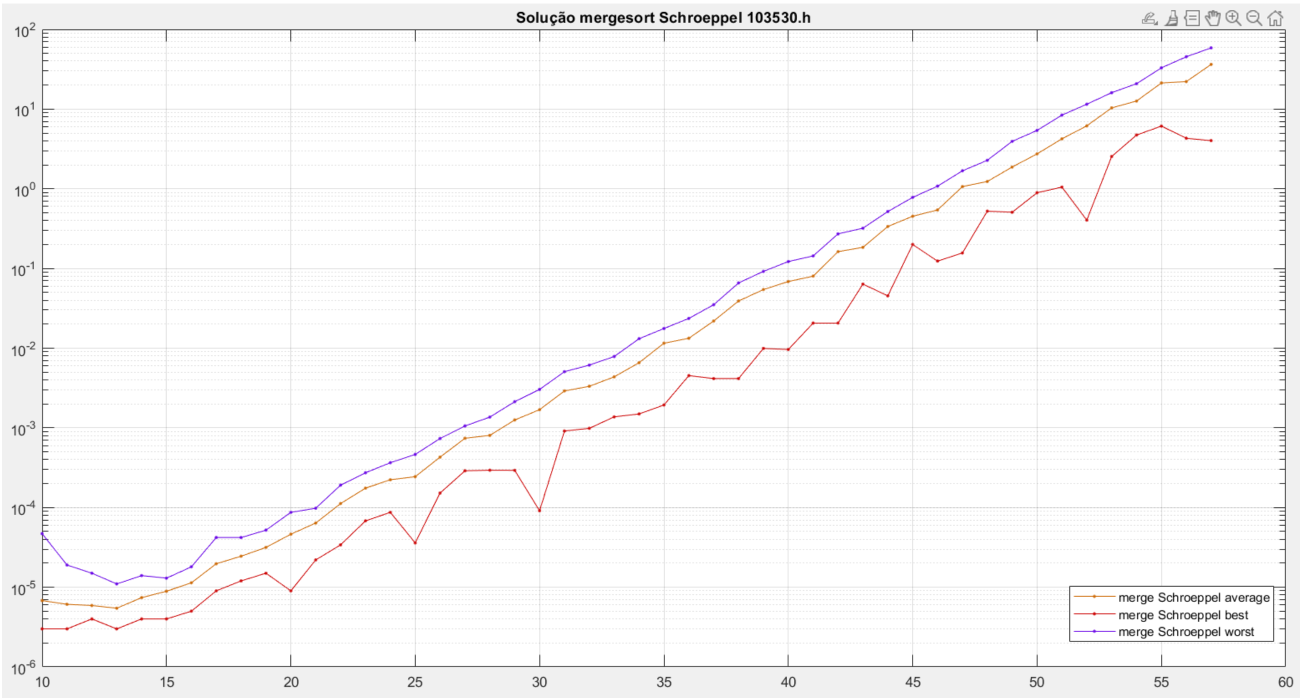

Figura 12 - Schroeppel-Shamir MergeSort 103530.h

Gráficos Schroeppel-Shamir (103668.h)

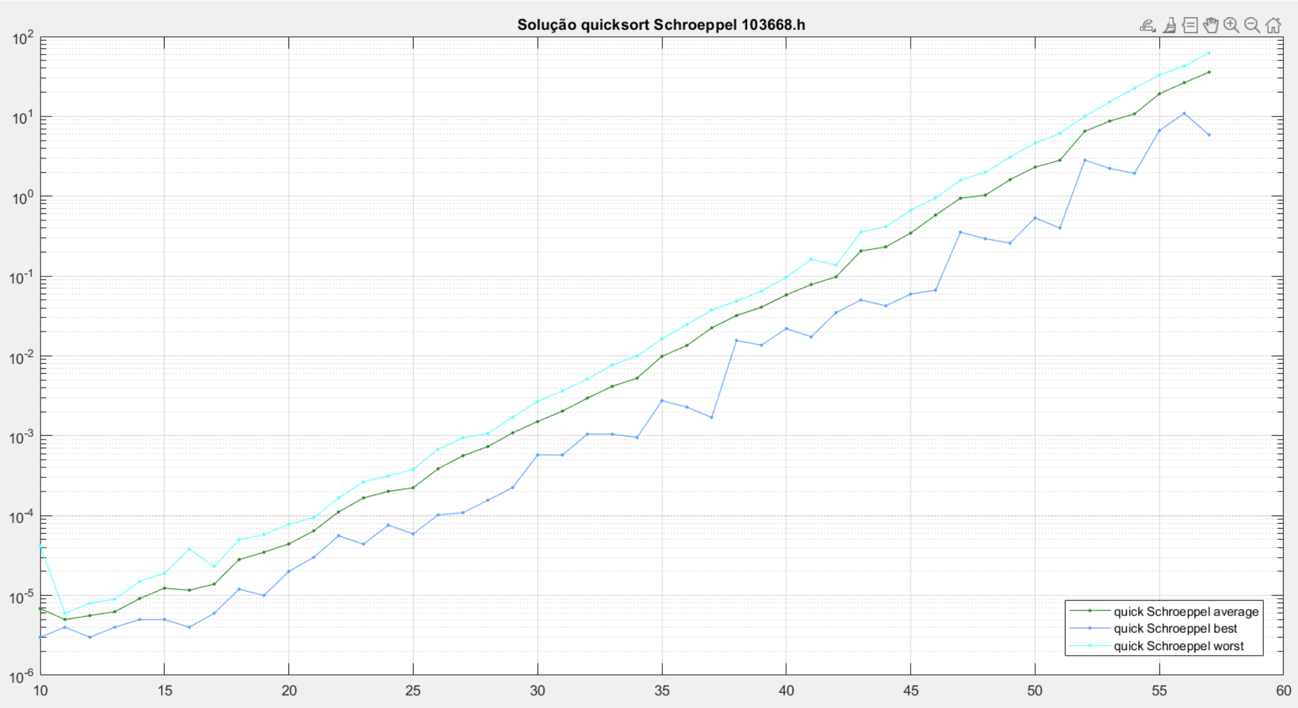

Figura 13 - Schroeppel-Shamir QuickSort 103668.h

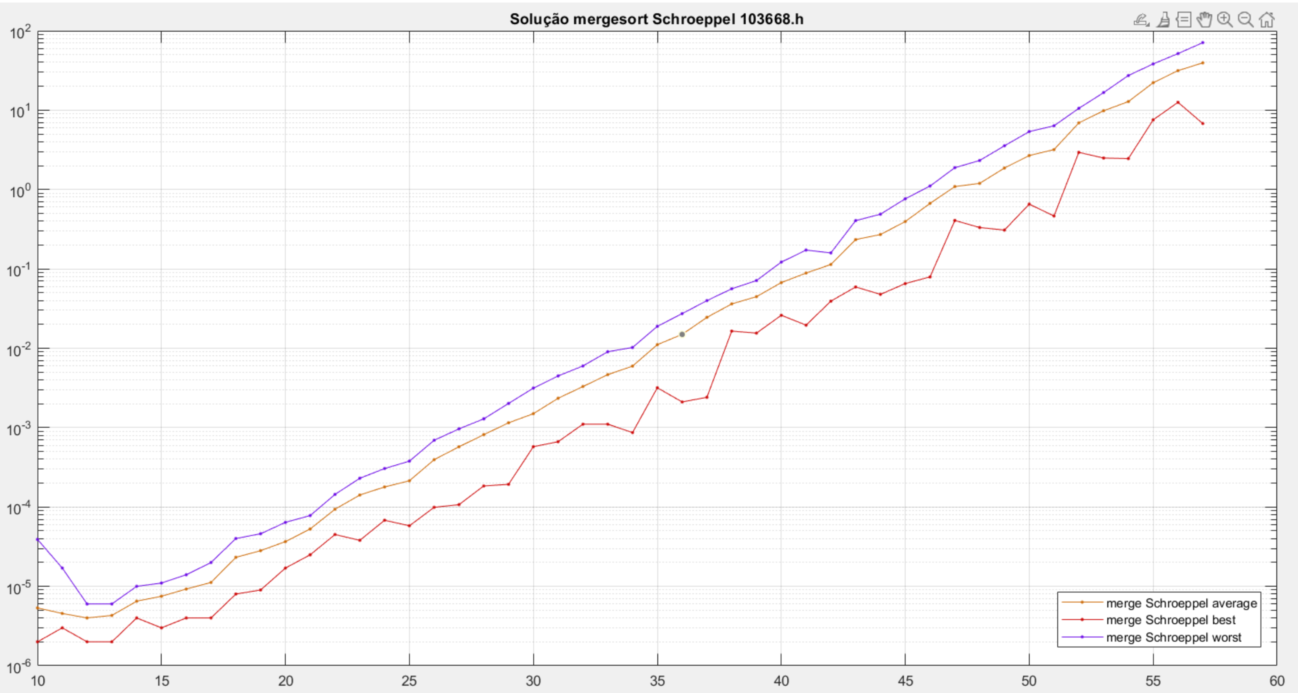

Figura 14 - Schroeppel-Shamir MergeSort 103668.h

Comentários/Interpretações

O primeiro algoritmo desenvolvido foi o "Iterative Brute Force", apesar da sua complexidade computacional ser a pior das soluções propostas, este foi extremamente importante na resolução do problema, pois serviu de algoritmo "piloto" que desencadeou a resolução do problema de outras maneiras. Para além disso, foi interessante identificar que em alguns momentos, esta abordagem, apesar de trivial, conseguia encontrar a solução em tempos extremamente inferiores aos restantes algoritmos.

Relativamente às abordagens recursivas, "Recursive Brute Force" e "Clever Recursive Brute Force", apesar de serem teoricamente diferentes, apresentaram resultados extremamente semelhantes. Isto deve-se ao facto, da otimização que o "Clever Recursive Brute Force" apresenta reduz a quantidade de cálculos quando não existe uma solução ao problema, o que nunca aconteceu neste trabalho, pois todos os problemas presentes nos ficheiros "103530.h" e "103668.h" tinham solução. Observámos, com maior detalhe, que o próprio "Clever Recursive Brute Force" era ligeiramente "pior" (pode ser observado nas figuras 15 e 16), devido aos dois parâmetros extra na chamada recursiva.

As abordagens mais eficientes são claramente "Horowitz-Sahni" e "Schroeppel-Shamir" tendo chegado ao final de ambos os ficheiros-problema (até n=57). Nestas abordagens tivemos também a curiosidade de estudar o melhor algoritmo de ordenação para o efeito, tendo utilizado o algoritmo de "QuickSort" e de "MergeSort", ambos têm complexidades computacionais semelhantes, diferindo somente no pior caso, onde o "QuickSort" apresenta uma complexidade de n^2, enquanto o "MergeSort" apresenta $nlog(n)$, no entanto, o pior caso do "QuickSort", no nosso contexto, nunca acontece. Dito isto, foi observado tanto na comparação numérica como gráfica (figuras 17, 18, 19 e 20) que o algoritmo de ordenação "QuickSort" aparentava ser mais eficiente do que o algoritmo de ordenação "MergeSort", portanto nos gráficos de solução final colocámos só a versão do "QuickSort" das abordagens (figuras 21 e 22).

A abordagem que se destacou foi a de “Schroeppel-Shamir”, tendo alcançado n=57 de forma mais eficaz em comparação à abordagem “Horowitz-Sahni”, no entanto ambas apresentam a mesma complexidade computacional.

É através da abordagem "divide-and-conquer" que este problema diminui de complexidade computacional. Se compararmos as abordagens dos algoritmos recursivos com os últimos dois algoritmos descritos, a complexidade do cálculo de todas as possibilidades das somas parciais (que corresponde a um passo nos algoritmos que requer muitos recursos) vai diminuindo à medida que dividimos cada vez mais o array “p” (para as recursivas, $O(2^{n})$, para “Horowitz-Sahni”, $O(2^{\frac{n}{2}+1})$ e para “Schroeppel-Shamir”, $O(2^{\frac{n}{4}+2}))$, notando-se um padrão de $O(2^{\frac{n}{x}+\frac{x}{2}})$ de complexidade, sendo “x” o número de partes aproximadamente iguais em que o array “p” é dividido.

Era expectável que a complexidade computacional do algoritmo de “Schroeppel-Shamir” fosse menor e não igual à do “Horowitz-Sahni”. Na verdade, o algoritmo de “Schroeppel-Shamir” apresenta mais eficiência (figura 21 e 22), mas isto deve-se ao facto dos recursos de memória não serem tão requeridos pelo algoritmo em si, o que poderá por sua vez influenciar a performance da CPU.

Gráficos Finais

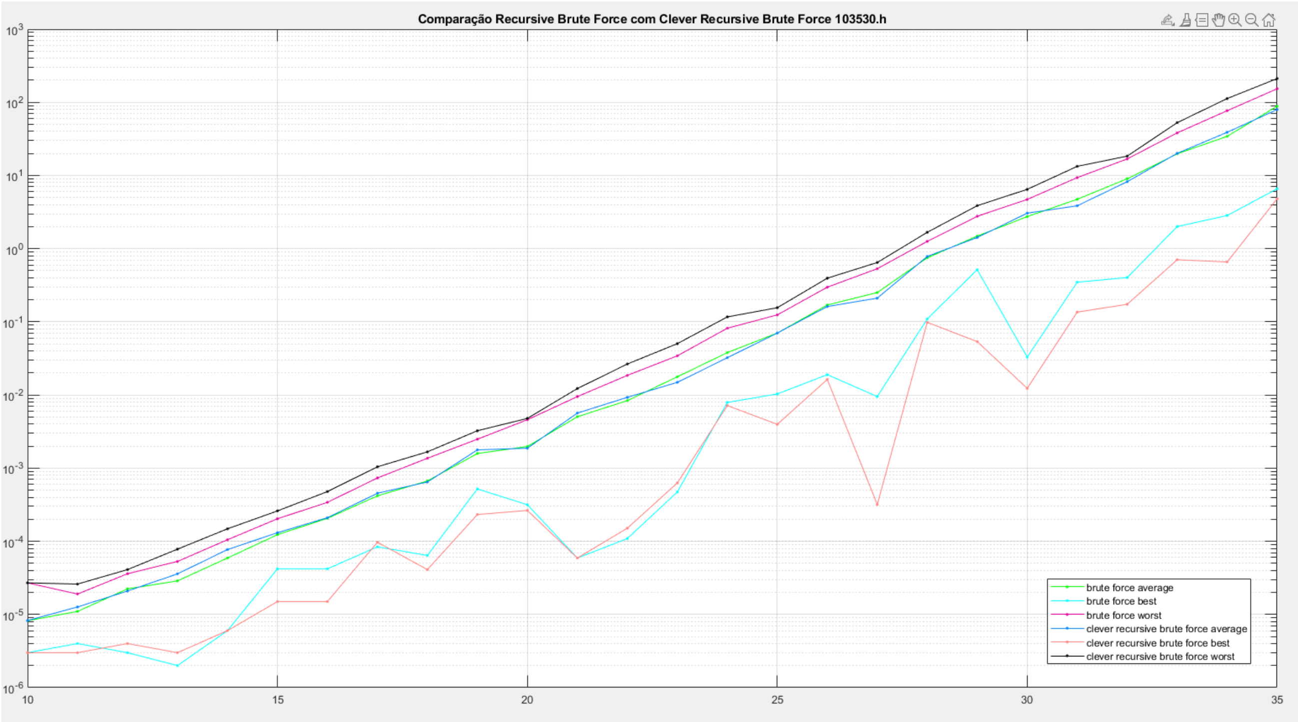

Figura 15 - Estudo Recursive Brute Force e Clever Recursive Brute Force 103530.h

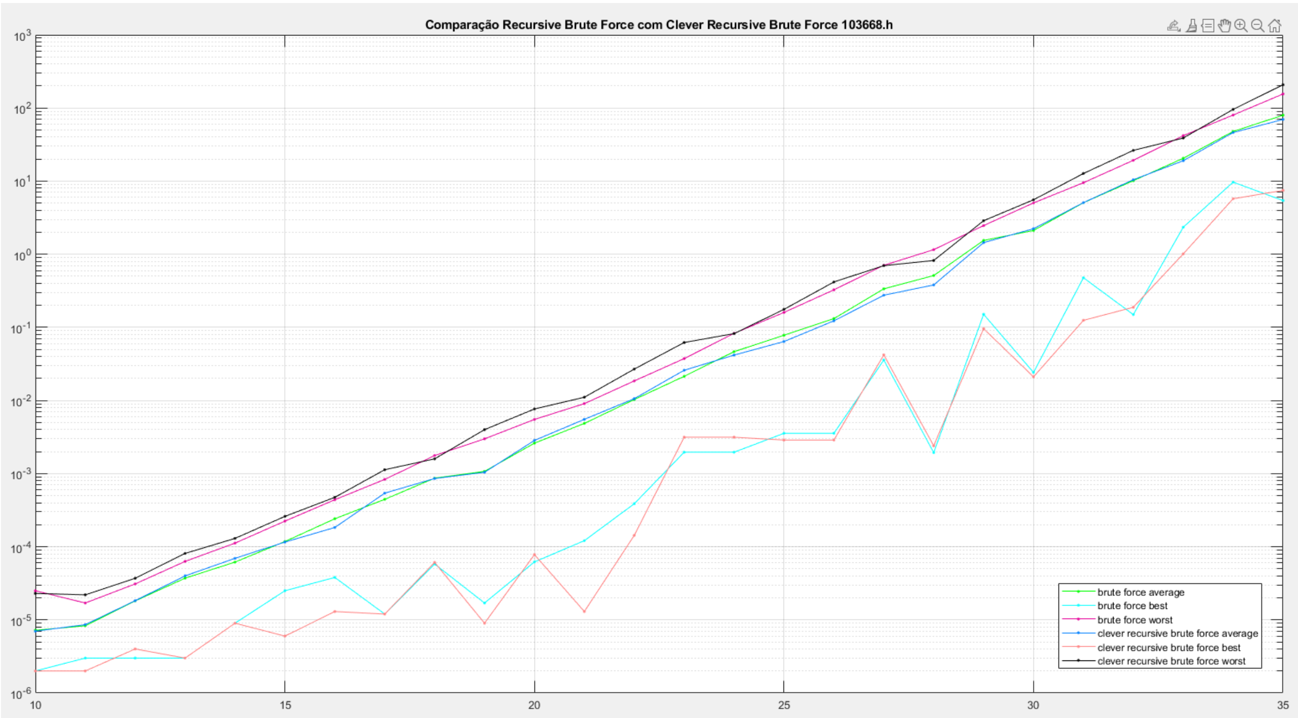

Figura 16 - Estudo Recursive Brute Force e Clever Recursive Brute Force 103668.h

Gráficos Finais

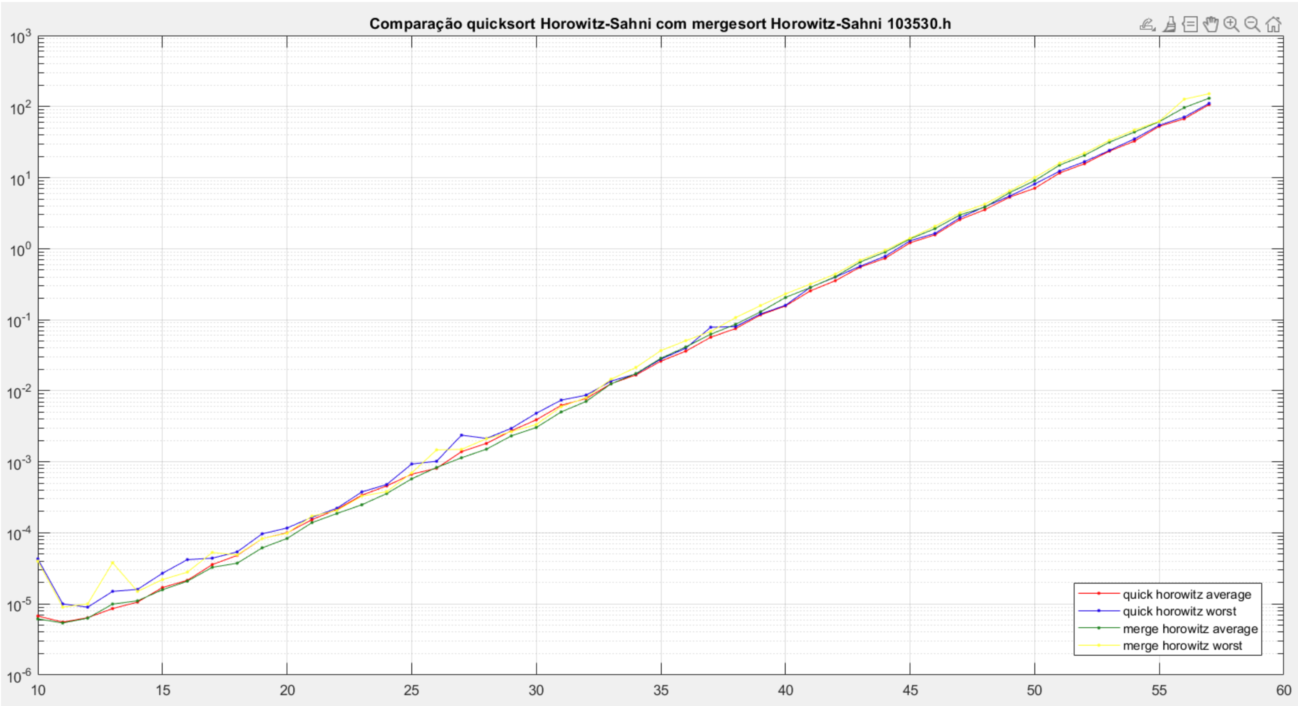

Figura 17 - Estudo MergeSort/QuickSort em Horowitz-Sahni 103530.h

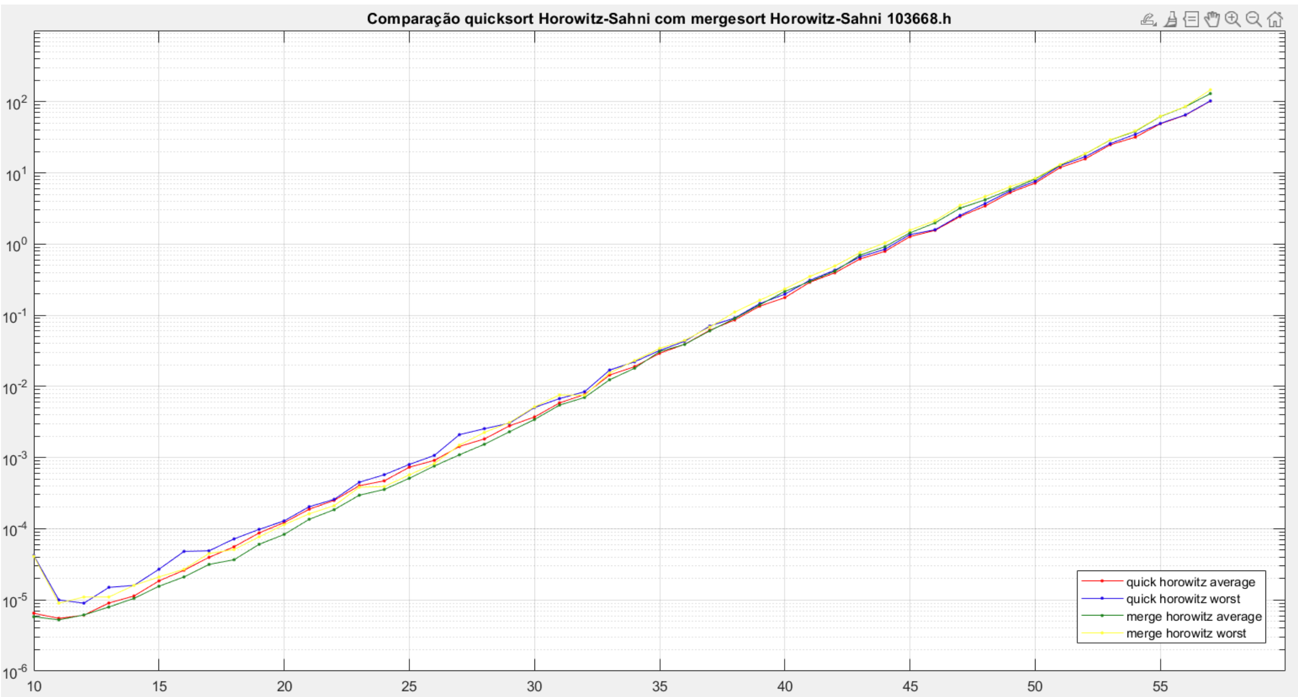

Figura 18 - Estudo MergeSort/QuickSort em Horowitz-Sahni 103668.h

Gráficos Finais

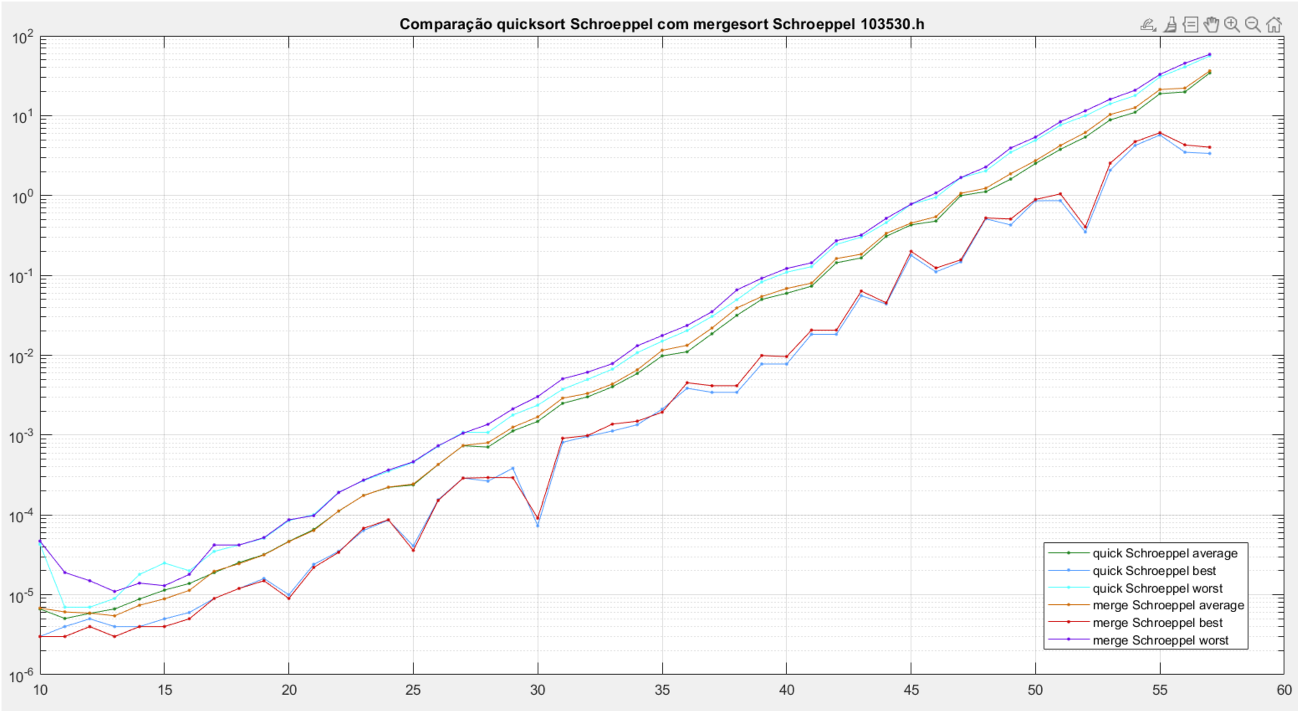

Figura 19 - Estudo MergeSort/QuickSort em Schroeppel-Shamir 103530.h

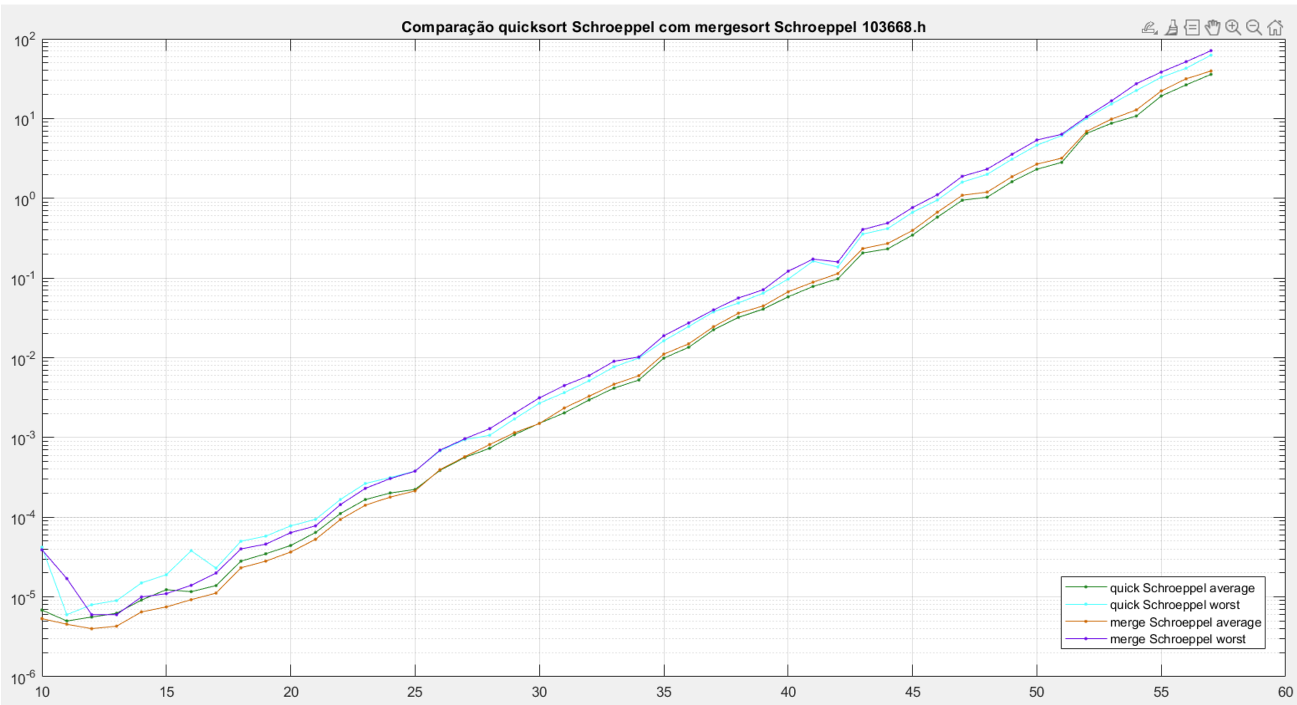

Figura 20 - Estudo MergeSort/QuickSort em Schroeppel-Shamir 103668.h

Gráficos Finais

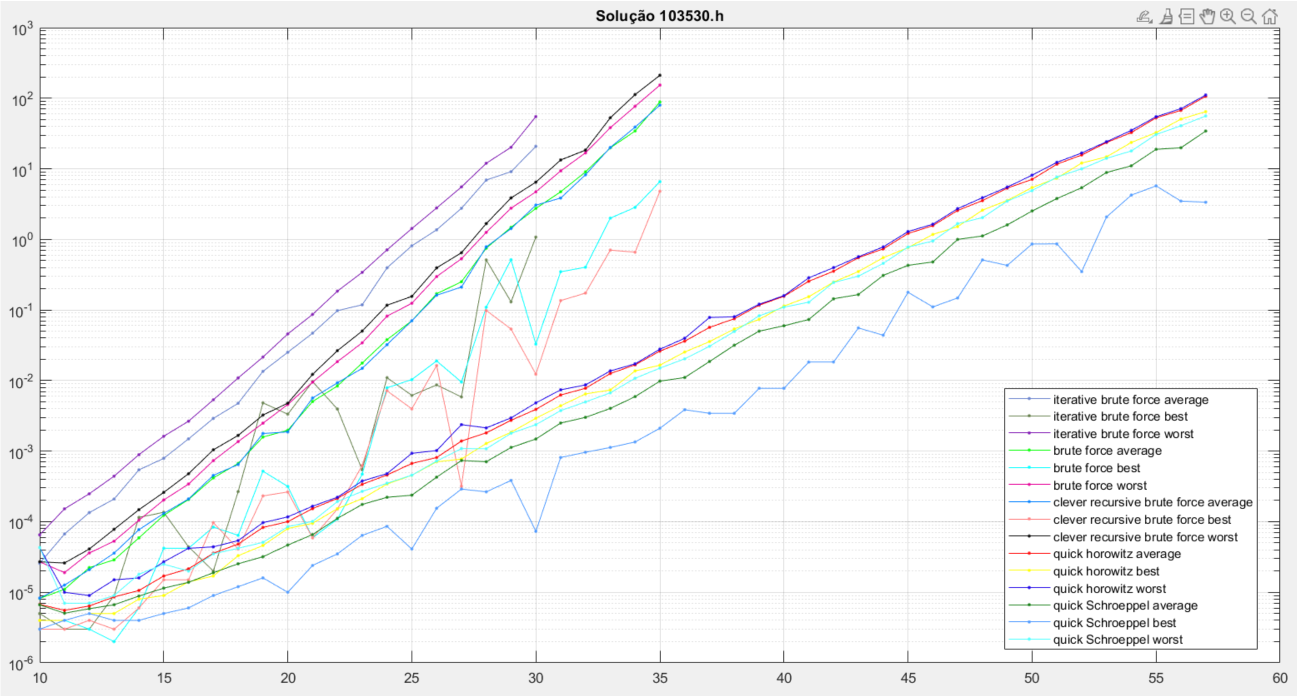

Figura 21 - Solução 103530.h

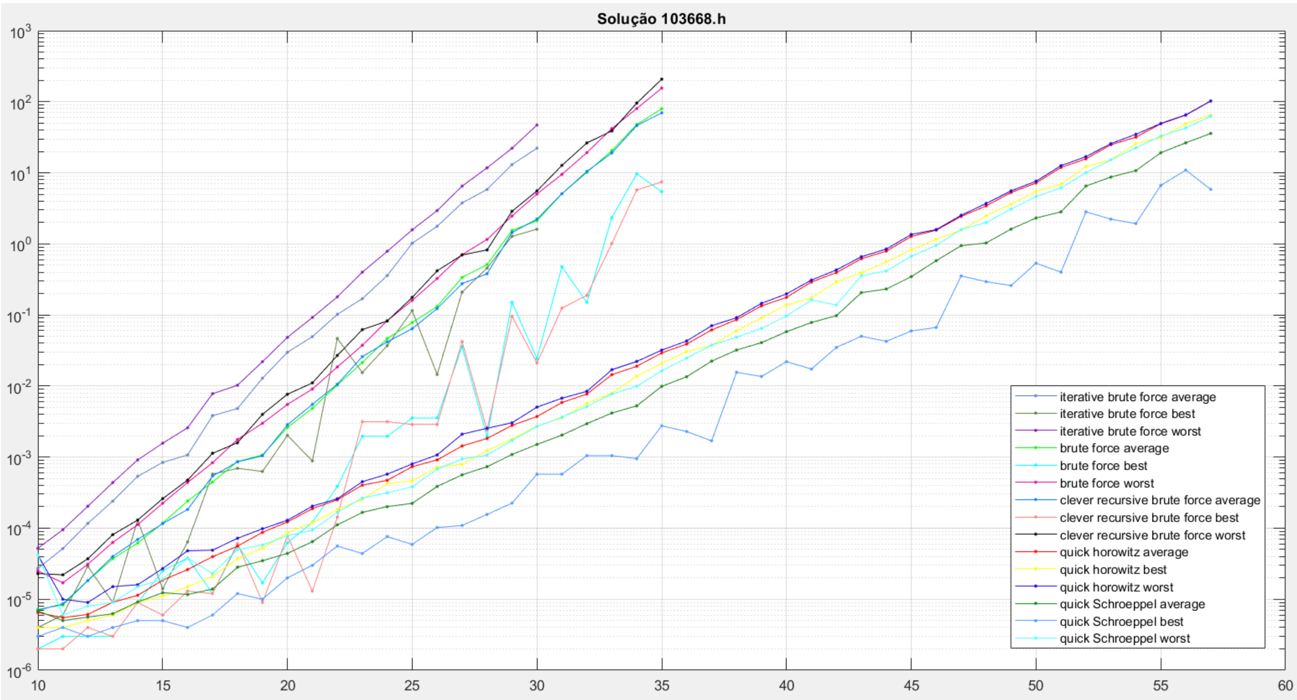

Figura 22 - Solução 103668.h

Soluções obtidas

Solução 103530.h

n=10
0110100001
1100010000
1110110100
0100000110
1110110000
1010000010
0101100001
1101110100
0110001101
1010001010
1011100000
0010010111
0110111110
1010111100
0001100010
1110011011
1000001110
0110101000
1101010101
1000000011

n=11
01000100101
11111111011
00010000011
00101101111
11011000110
00011110100
10000100000
11101100100
10001111011
00101011001
11110111101
11101011100
11001100110
01111101010
11000001110
01000011110
00111111111
00100000111
11000001111
10001110110

n=12
110100010001
101110000110
010110001011
011110011101
001010111101
101100011101
111000100000
110101000001
011101000100

110111110010
011001011100
001011101011
000011000001
011101101111
100111001111
110010000000
011001001001
111100110010
111100000111
110111101110

n=13
1000110110000
1010101001000
1000111000000
0000100001000
1100110011111
1011011100100
0110111011111
1101000001100
0111100000010
0101100101000
0011011110000
1010101100110
1000001100010
0100000010100
1100101010100
0001110000100
0101111010100
0001000001000
1111010001000
1100010111100

n=14
11001110110001
01001010001100
10111111111001
01110000111001
11010000010010
10111111100110
00111011100101
01000110011010
01101010001111
11010100001010
00110011111000
00111111110100
10100001101101
10101100001101
10000001000111
10110001011000
00001010000110
01010001001001
11010000000101
11110100011111

n=15
010010101001111
110011111011110
110110101100111
100000111011000
011100010101111
100111011011010
010011000101010
101100000001111
011010111010100
010011100010110
110000001101000
010110010101010
111100001010010
110011010010101
001011111001100
110100101011010
111000011110101
100110010101010
100001101011101
001100000000101

n=16
1101011011010011
0110000100001101
0100110101011101
0110110010100001
0100001011001000
1110000100011001
0010001110100101
1001001010100101
1001001001111100
0011100110111101
0101111111100110
1011101111110101
1011100000011000
1010001011111110
1010010110011010
1100011101010001
1101000011101011
1001110101100000
0011110100100000
1100110100111010

n=17
11100110110100010
11111110010000101
01010110011100110
10001001100100010
11010100001100010
00100111001000110
11100011100000000
00101111101011010
10000100000001010
11111110001010010
11111111100000000
01101010000000101
11010001100110100
10011111100000010
00011100101101100
01110010011100000
00110101111101010
00111101010000100
10110011011110010
01101100010001010

n=18
111011010101010000
100111001110100000
110001110111001000
011000110110010100
010100001001101100
000010001011101000
111010010100100001
010000110001000001
100100001100101000
011010100000110010
000110011100001000
100100100011001000
011110100010000000
101110100100101100
000011100011011010
011110110000110110
010000100011110010
111011101111000000
100110101010011111
011011100101010011

n=19
0111100111011100001
1100010101100110110
1110010001111000001
1010011110100101001
1111011110001011110
0100000011101010010
1100011010110010001
1001110000110000111
1001111000011100010

1100010111001000001
1010011001101011011
0101000000100000011
1110001101011101001
1100010010111010101
1001111000011110111
1100101010010001010
0100100001010101100
1000111100001011111
0101011100101110111
1101010111101011011

n=20
10000000000111100111
10010000100110000011
00100111010100011011
10010110100101110110
01111111001101010010
00010001000111100010
10100010000001111111
01001001101000011010
01011000010010101000
11110011111011001000
00011001101100011110
01010001001010011001
01001111111100001101
10010000001100100101
01000010111000111110
01010111000010011000
10001110110100011010
00010111111001101011
00110010111100101111
01001001110101100111

n=21
000110100001000101100
001101110000000101110
011110011101110011010
111110110110100011000
100010111111011110010
110001110111111010100
011110110000010001110
010101100101101110000
011100011001000000110
001110111000101001100
111110000011100010110
010010111101111011110
111101110001110011100
001001010010001000110
110000011011010110000
101101110011001010010
101000100110111101010
010110101111111111100
000000011000111011000
111101010000111010100

n=22
0001001000011110000010
1011011110011100011111
1101000000010110100000
0110010001000011101110
1000000011100110111111
1110111001101101111010
0000101101111011001001
0001000011000001101010
1001101110101101000000
0001100000000101011000
1010101000000000100101
1101101101101011100001
0111010110100011001001
0110110001111101000100
0110111111011110110010
0010011011100100001000
0000000101110101111100
0100001011100001011000
0111011011100100110010
0101111001011001110001

n=23
00010001110000011011000
11011001010010101110100
00110000111101110100101
10001110010101011101010
01100011011100000100100
00101011111101011100110
10000001010000101000010
10010111000010010001010
01111101111010110000010
10000010111001000000000
10100001111111101001101
00101100001000100101000
00000010101111111111001
10001010011101011110111
10001101100101000100000
11010010101011010001100
11101100110010111111001
01001110110010100111100
01011001100100000110110
10001010011010100100001

n=24
001010001110000000110101
001101000100110001011011
000110011111001101001011
110000010111001001000010
011101001000100100001101
000110010010001111001100
010110001110010000010110
100110110011010001000001
111101111110101011110111
110110100111000010010010
011001010010000010010110
001101001100001110101001
100000100011010011010001
110011000111101001011011
001101011110110000100101
011110010100111010011000
010111110010011000001101
011100010000110010000101
010001000110101111000000
111111001110011011010111

n=25
1010100100011100100010000
0010000001000101011011010
0111110001011011100010000
1010000001001001100110111
0000111111011001111110000
1000110101111101110001100
0101010000100110001111100
0011111100110111001010110
0100001010101010110010100
0110011000100100011011000
1011100011101000101100010
0111010010000001111000111
0111001010000101100100110
1010011100010100010000000
0001011011100000101000100
0111101001000000011110010
1011011011010111111011010
0010111111001110011110110
0101111111111010111010000
1001000001110000110101000

n=26
11101001111111000010000100
11001010011011110100000000
10100010001110110000000010
11110001000000001010101001
00111111010111110011100000
01111000010101010001100101
00110000000101100100110011
10101000011001000000110000
01101101011110100101010000

10101010111000110110001010
01101101101011011100100100
00010000001110001010001101
01110011101000101010010100
10010100100110100101100101
10110110101010010100010011
01011111100001101001000010
10010100000100010010010010
11111101010111010100000101
10111010000110110100001100
01011100100010110011010110

n=27
000111010110100011000010000
010001101011111110000000000
001110110111000111010011011
101110110011000001000101100
110100110010101101110110010
000100101011100011100100101
010110010111111111100010000
011011100101000110011011100
011100001100111111011100110
101001110101000001110000100
000100101000100000100000000
011111101010101000111110100
110110100000001010000001010
100101111010101111011010110
000000111111000110001110011
001010110101010010100010001
100100100000111010110000001
011101101111111001010011101
101111000101100010100010101
001011101000001100001111100

n=28
1001011101101000010011111010
1010000001110001000010101101
1011110001011000011010000111
1011010010100000111110000111
1010011100011101100101010000
1111101101110011010000001011
1011010101101010111000001111
0101111011000110111000011110
0001111000101100111010101000
1001101111001100000001011001
1111110010111010001111011100
1101000110110100010100011111
0100011010101001100100001001
0010110101110001010111110100
1101000000011101110110010110
1011111110011010001110001111
0111000011100000001001111001 0
1100110110001100111111001001
0101110010101011000010111010
0110011011100000110100100100

n=29
11000110000001001000100010000
01010010100111000011011111110
00111011000010110001011010000
00111110001100111011100111110
01111111000010010111110011110
11111110011111110101000001100
10101101011110000100000100000
10101010110001010110110000010
00110001011010110110100111000
00111111011111010000100011110
01000101010101110110110100000
01001111111001010101001111100
11111011010111010110110001110
01111100110010000010000010100
10000011001101110011000000000
10101101100010000001111001100
10000000101111110101010000000
11100101110010010010101001110
11111001100001000101100000010
11000000010100101111101000100

n=30
111011011001001111011100101000
000101001000100011001000100000
110010000100000011011101010010
110101000111110100111100000011
010000100111110101011010101001
101101001100011110011110100100
101110100001110111001100011000
011110011111011100111100100011
100110001110110110111000010000
000000011010000011100000000110
000100001101110001011010001100
010110100000110101100010010001
111001001110011010010000001001
100111001111101010001010110100
110011000100110000011100001101
100100011111000010100110010100
101001111100001011111000110000
110010001110010111101100010101
110011101100011000111101001100
001010111010100000100100001010

n=31
0000110011001110011110011001110
0101010111101101101111110011010
0110011001100011111111001110010 1
0000111011101111100111011001010
1010111010000100011000000100101
0001010110011100010100000111101
0110110011101111111101010010100
0010001010101110001010000001000
1101001100000101101000001100000
1110100011110010100011000001010
1001010010000101110000110001010
1111101000010111111000001110000
0111000000000001000011101110000 0
1111100000111010010111010100111
0101000101000110101101000001111
1010110001111100001110010110010
1000011000100100110000000110100
0000100100111011000101111000000
1001101111110101110001011001111
1100110101100110100110000101100

n=32
10000011010010111111010110101011
10111110011100011101111101000110
01000101001100111110000011000100
10111001101100110010011011100000
00101001000101111001000110101101
01011011001001110101101100111100
11010001101101110001100100101000
01110100111000000001111100010000
10110100011110111101110000001010
01110100001101100011100101111100
11001001111110000100101101010110
10000111111010101010110111101111
11011111010100010000011100010000
00010110100011001101010011001111
01001000000111100010010110110110
01000001000101111101011100110110
00000101011000001011101001000100
01011111010110011010110100101110
10010010101101111110010100001010 0
01100100110110000000110101011000

n=33
111111000100001010010011011010010
000011101001110000111111001010010
110110010100101111001101011111010
001011011101110000101000000000000
110011000011110100010110000000000
100100101010010000101010010010010
100101110001000100011000010101010
101010001101000000111010110101110
101010000110101000110101001111000

0011010011001110110101001000000001
1010111001101001000100000011111010
1101001010100001000110001011101000
0000110101110001000010100100100010
0011100000110010001100001111101110
1001011010011111000010111011111000
1001010111000110110000110111111000
0011011110111000000101001011111100
1101110011011100100000110110101000
0011111001000110000101101101011110
0110000100101000000100011111101100

n=34
00010001100011111111100111001011110
11111101111101010111010100000001101
11100100111110000001111111011111111
00111000100111110000010100000110110
11111110110000011111111000000111000
10111010111110110101010000010010101
00001011011011111111010101011100001
10001010000010110110110110110000001
00001001011100010000000001000100110
10010000101111111001010100111000010
10001111000010101111111000001001001
10011101110010001100101110110000001
00111001010011100100111011010000101
00101100000110111000101110100000110
01001100100011110100111101110100000
11110001101100100110000000100000010
10001010101000100001111100010101010
00111000110110010000000001110000010
00100010000111001100011000101100000
00010110000000000101101110101000010

n=35
001100000101011111111001010110010111
101011000001101001100001111101000010
110000101100100010000000000100011010
000101100010101010010110000010101000
110000110010110001010100000010010101
101110100000010101001100101000110100
011110010011001101000010101111100010
000101101101001000000011110000010101
110000000101000101101010000110110111
111110010110100111000011001001001111
001110101111001101011100101010000100
001001011010111000101111101010110000
110101100011110010011111101111000111
111111110111101011101101110101101100
011010111101100001111110100111110111
100001110001001010100100010001001010
100101100000110110001010000101010010
011100001111110011100011011110101100
110010110100111111001110111110000100
111111111000010011111101001100000010

n=36
0011011100001010011010110011001011011
1011110001110001100001000101011010001
1101010010010110001100110110011100010
0000111010110110000101001111110110011
0111001100011101111101011101000010011
0010011010111001101001111011100011011
0001000101001111001010010110000111100
1011011001010001011001101001000101111
0010000101110010100111101000000100011
0110000001110111001101101111111011111
0000100101100011011110111011100000111
1001111011010000010100111000100010100
1000101110001100001000001111100011111
0000110010110101100101011011010011011
1110101111010100100001001110001100011
1100000101111101000010010011101011111
1001011011010001010111010111001110111
1100010010010101000110001101011100011
1101010101011110100011001111000110000
1010011101100110111000101010111111000

n=37
0001011010000000111101010010011110000
1110111110100110001000100110010010010
0000111000111111110110010101011110110
1001101111001001001111011011110111100
0001110001100101101101110011000110010
0111110110110111101010010001000011110
0000101000101011001101000010001111100
0011001000000001101101101000111001 1000
1001111111001001000010101001001100000
1011110000101111010010000100101010010
0010000101110011110111110101111111100
1000010101011110101011101000010001110
0101001101111011011011010010011100110
0001110110111000100000001111100101010
1110011011111000001010011011101110010
1111001001111000011110011010000100010
0101011011001011101100111010011100100
1111011001010110001100011010010001110
1000010000100101100101011011001100100
1001001011100000101111111001001011000

n=38
01000000100100111110110100110100001101
01010001011011001111100001010010110001
00111011111011110010011010001000110100
11000010100010011111100011110101000100
10111011111110100101101010111000011000
00101110001010111110000101001000001100
01101011000000101100100101010001101000
01110100101101000010001101000010100011
01111001010011110100100011101001010100
11100100011001110110101000011010000001
11010001111110000111011100111111000100
00001110110111000001101110010001001011
10011111000110110010101111000100110000
10111101110101010011110100000000110100
01100110100111100001110101011000010010
11110010111011110000010110001010110001
11011101010110101001011011101000100011
00101001110111010010111100001010100110
11001111101010101101111011100010011000
01101010000110011001000000101000000110

n=39
100000000111011011001011111001000100100
001111100110000000100110001100101111 0000
110010110000101111101110110001011000011
111110010110011010101001000011111001000
000011001111111101111100110100010010 1001
110000001101110110001100111001101110101
100000010000000100101101111011000011 0000
001000000000101110000000101110011101 00001
101110000010011111101111011101000100010
100000111010111101111111001010010101 1001
001010010011101111111111110010000000100
000001001010001000101001001000101101111
001100100100110010010100100010101101110
001111101111001000101010100110110001011
011101010100101111010110011100001000101
010111010101110001101010100011110010111
111110111110100001000000000111001101001
101101111010000000000001111000100001 0001
000101001110010000010001100010000010111
010001001110010001011011101111010011101

n=40
0101001100000000101110111111001111111 0110
0110000000000001011011111111001001011111 11
0000010101000100111100001000110010100010
1110010000111010101110100011001001110001
1110110000101100110110000001000110110101 0
0010001100001000101101110111011101000011
0100101011111011010101101011100001101100
1110011101110101101001000100100100100001
1100001101010111001110110010000111101011

0010100101010111101100000111110001010110 10
0110001111000000001101111011101001111010
1101010111111001111110100110001110000100
0001100011010101111000010101101011100110
0101110001010010011011100110010110111100
1101011110001100101011011000010111101101
0110010111100001101010111011011001010010
0100110100111110100011110001110000010110
0000101111101010101000010001111000010001
1001100110001100000001101001010001000001
1010001000010111000111001101000011000111

n=41
10001101101111110100101011010110101011000
01101111000001101100010000100100101100101
11111001011110001100111010110110100100010
10011011100000110001111111111001110000100
11011001011000100011010001010000000000000
11110101011001010001110110011000000100000
00000011001101101000110101010011101100100
01000101010000001101111111111011110110000
10000001100010011111000110001000000001000
10111000010100000000100101000110100000100
11010001001100100000100111110100000010001
00010001010110110111010110110100001011101
10001000001100001110111100010011011000110
01011010111011110100000011100111000110001
11010100010010100011000100001100101010001
10001001110000001111001000010110101010100
01010111101001100101110110101111000000110
01100111101101000001011010001111101110110
01010101000110110000010101100010000001110
01111011000101000000100101000101101110110

n=42
011111111010110011100111001110110110110000
100101000010111110101100100110101000111010
010010001100010111111101111100111110000000
011100100011011111100101011010110000110100
010100111011111001101010001000000100001000
000101000111011101100001000000111000011001
101010011110111110000110110101111111000000
001010101001110111101110100100010001100010
001111000000111010101001110110011000101100
110110100011101001111010010100000001101000
001101110111100000101110101100110101011010
011100100111000110001101101001100100000111
110110110110011000001000011011101000110110
000111010010111010001000010011000010000010
011101101000010000000001100000000001101000
011000010101111000011110010000101011000110
101000110011000001010010011011110000100100
011010101110101100110000110110010010000111
011101100000111101011000101001010110001000
001101010001111101100100000001000011101001 0

n=43
0111000010101011000100110110011101001010011
0110000000000011100000100111101100110000011
0010011010100011100111000110110110110100001
1101101000000001101011010010010100010011000
0101000110100110101000011110000111011011101
1010011111001001011111010001110011011110000
0000000100011110110111000100011110000101011
1001001111000111111110110100111001110111010
1001001011100001011001101110010110101010001
1101011001100100101100101110111101101001110
1110111110000011011110101111001000011001011
0101101101001100001100010011011111011001111
0100110110001110010011010001010001100111001
0101100101010010110100101110111111101101011
1001101111011101000101111011010011101010011
1010001110000011111010001000101101011011000
1001010110001110100111001111111101101010110
1000100100001100011101101111010100011010111
0100111010100100010000100000000101100010001 1
1111001101110001001011100100101010010010101

n=44
00110010000110000101111010111100011101100010
01010010101101001001010100001001001111111011
01111010001101011011100010101001101110010111
01100101011010010000111001010011001110110101
10110001011010010101111101110101111100111110
10100000000100101011011011111000010011101010 1
10000010000110001110110010111000100100100110
11000001000110101010111101101010001000010001
10011001000010010011000101001111001111011111
00010011001010100010010000111000010001100100
01101111111110111011110101001001010011100001
01111111111101101010101011010010111000110011
11010101001111111001010001110001110100000110
01110110100011101110000101111011001001101010
10110101000110011010101010010000110001000101
10010011111111110010110010001111001010100010
11001110111110111010100111010000000101101001 1
01100100111101010111111101010011100001001010
01010110110110001000100001000111001010000110
01110110111001000111100001000000110101000001

n=45
110101101111011100111000010000100111110010000
101000010011001110100111110000000100111100100
110110000111100100100110111010101110000010110
100001001000011100101011010100100010001001110
011111100100000111001000001110101101011001100
000111011010111000011110001110001000001101110
100010010111010111000100000011001111110010000
100110100110101000110111101011000111101111000
000010011000011001011001101011010100001011100
110000101010101001100000110100011100110101010
010110001011001001010111100000000100010011100 0
100000011100011001011111100110100100101110010
001011100100000100000000111001100101100000000
100110001110001111101111000000000101101111100
101010000101100011001101001010010100101110110
101010101100000010000011001111110000011010000 0
011000011000100101101010111011100011110000000
000101001011100010100101010111000000111000001 0
101100001000101110011100101101101101011110100
000010110001101000100100101110010111100101000

n=46
0110010010100101001100011000010010111100100001
1010101000111000011010111010011010001100001100
0101010011111101100010011111011101100110001010
0110100001100101011100001000000011110001100000
1001000100111000100001011001001000000000101011 1
0100110010100110000011011010001010000110010101
1111111010010100000100011011000110100011111100
1001100110011100101010000011001111101001010011
1100111110111000111100010100001111001010101010
1010010110000101010010110011110001101100000110
0001011001000110011001011000111011000100011100
0000100100111010111110010110100010010001011000
0010110000011111100101000111001000010000100100
1101111011001100010101000011010011111001101100
1110001001101001011101111001111100110010011000
0101000010001011001001010110110111101000110010
1001011001010011100000010110110000110011100010
1000010010110011101111110111001000010111010100 0
1101000100001000110001111000110011010010101100
1101001010000100010100000111110110000011010010

n=47
10010101001100001100111001000000111111111100000
00001001100011000000110001111000100100100110010
10011000000100010000000000101110000111001110010 1
01101100000101101001001001000101101101001111110
11011011100111110011110011010010110000101001011
00000000110000000110001010100001101100100001110 0
11111100101010001001101100000000100000011001110
10000101110000110101111100110010111001101110110
10100100111010010100101000010110001111001001101

11111101001111110001000111000010001010010100101
10100110001001010010010101010100010100111000110
10010001011110010001110111011100011011011010000
010000101110101101100111000100010111110000000000
10011110110111101010000110111001100111101000000
00000101111000111010001111011011110100001010001
10111100011010110011000100011011100010011010100
10001110000000101100100000000000001011110000010110
00110000101010111001001110000111101101101000110
01111010010011101100010010111100010100010011001
00100011010001111101001100010100001011101100100

n=48
0100100101010101110001110011000100000000100000001
000111010101111001101010000101110111111101010001
0101001110000000100111101010110101100110001011110
000001110001000010100000000100101000011010101101
11010010010110000000011001000000011100001011111101
11000101000000101011010000001100110100111001001000
0111110100100111101101100001001100110110000000000
010000110111101010001111001110111011110111000001
010100011001011000011111001010011110110011110010
110110101011110101100000110100011111010011001111
1001110110001011100000001010100110000000000100101
011111011101010000011001111111011000011101000110
1110010101001100011011111111000000101000011101100
101100111101100011010111101111011110110100001001
100101011010111100110100110111001010110010111011
000010011000111000101111100111111011100110010110
0001000000000000001110111010011100010101110100011
000101001101001001100100001010010100101001000111
1111111111110011110110100111101101101110010111101
111101011001110101101111010000101100001101011111

n=49
0100000110110011010000011110010101111010100100110
1011011010011111101110110000000010010000101001100
0000100111011000100101010111010111011100100000000
1001010111101110100011010110101011100110101110100
0000101001101001010000011110011001011001001110000
0001011010101010001111011100100001110011101010010
0110011111111010110011010010100011011111111011010
11100010111110111111111111001100001111101001111110
0000000101100010100011100011111101100001001111100
0001000111100001111010101000001000111000011111110
10011000001110000110010010100000001001110010111100
10000100011000010000000101000011001110001110000010
10111101011101111011101000001100001010001101111100
00001010011110001000100111010000000000100010100010
01110000001000101001010101111110001110111101000100
01100110100110100010011100101001110111111111101110
00111111000000001111100000000000111101101101001000
1100111000101111010100000101001100110111010110100
00001000000110000001011110000111010111010011111010
01110001101000111111101100001011000000111000100100

n=50
00101101101010011100100111010011101000010001101011
00000000100111001000101110110011101111000111010110
00110110011100101001100010011110011110010000110000
101111011100101110101010101111000000001000100101001
00111000000011110001000010111010000011010011101000
11111110010001111111001100101001110100100110100101
010011001010000001010101100101000100000000000001010
11100100100111011110000100011010111010000010001101
00000110011010101011111001001110100110011000010110
1010100111111111110110010000000100111111110000000100
10101011010110110000011110100001110110011100000101
10001000011101010110000011110000101110110000110110
11011001010011100101000101000010000011000101010011
01000110010000101110001100110011101001010000100000
11100100110010101010100000101001110011001110101000
10011110001110011111001111000110010000000101000001
01111101101100101000100000101001001101000110010110
11111111001101001001001010011101110110011100100010
00101111010101010001010001100000100110101100000101
01001111101100101001101111111011101101110011000110

n=51
01100010111001110000001110110010100011010110011101 0
10101001001110001100010101100111001110011010100101 0
10000000101001100100000110111101110010111111001001 1
00101110100011110011010010100000010010001000110011 0
10001100101010100001100010101001011000111000101001 0
01001001110010011100110011001001101101100111101000 0
00101011111111111000111010100001100100100000101011 00
00010011000001011010110111111000001110111000111011 1
11100100000000000010011100000100001111111011011000 110
01010011101001111010100010011111101001100101101101 1
10110001110111001100001001011111011001101111100010 1
00111100001010001011011110010101010110000010110110 0
00011011011111101001000010000111111000010010000100 1
01011011000111111110100011010001000110011010101111 0
11100111000100111001110001000000111011110011100000 00
01001010101010111011001000000110101011011110110111 1
10011110000100110000011110111011111011011000010101 1
01111010101001000000010001001100010011000011100010 00
10001011001111110101000011011011110010010101110101 1
01111110010110000101010100110001000110100110000001 0

n=52
11000101111100001010100010111011101011010100100101 11
10111110011101011011011011101001011100010000011000 10
00000010011110000101110011110010011110000100101000 10
01011010001101011110001001100011110001101100011101 11
10111000110110010110011010110000101101100101011100 10
00100001101000100010000110011110111001000101101000 10
11111100101111010101100100100010101000010000011101 10
01000101011010011100101011110000011000000101111001 11
10000100100100001010001010100110111101101111111011 001
01101011010000000001110101010101000011101101101100 101
01011000100010001011110100101000000010010101011110 10
10101100010100111001100100111101010000000101000010 100
01100100011010010011100110011110100001100110111011 10
10111011010101101010011010000110000010000010000110 00
11100011110100001101100101110100101001010001000001 10
10010010100010100111010010110000111100010000100000 11
00101101111001010100111101011100100100101010110010 11
00011011001111100110010101000111001111000011000110 01
11011000100111010101111000110101010111101011010011 00
00000011110100110001110001101000111100111111101100 11

n=53
01000011010111101111010010001000010100000111110001 010
11011001111001101100000101001101001110000010001000 110
01001101110011001100001010110100000000111011001010 110
01010011001110010100100001100000101010100000111111 000
10111011000000101101100101101000000011101001101010 000
00111110010101001111110011111110000000000000000001 01100
01000110110101000101110001100101011101000111101110 100
01100111101101011111110111100110110101111011111111 1000
10011000111100110001101110111000010101010110001001 001
01001011000010100010011111110110010110100000011000 100
01000000100111011010101111101101110101011100111010 000
00100000100111011101110101001000010000100100000000 1000
11011101110100000110111001110101100011100110111001 100
10111100110100011010010011000010111110011000110100 110
01000010101001010010011010110110000011000110100101 010
10011011100000100101111101011110000000000001110101 1100
10001101000100111001101000101001001110101110111100 100
10100001010110010100011010111010101010011100100011 100
10001010001000111100101011100010100110010010110011 000
10010111001110010100011110011110010101100110010100 000

n=54
01000011110100010000011101001111000110001111010100 0001
01100011110001110100010110000011100001101000100000 1011
00100000101101101000010101000101101010110111001011 10001 0
01011110110101101011001011011111010101011011000000 1101
00011111110011011101101010000010010110111010101000 1000
00110001011100010111100110000010110011110001001110 0100
11011001111101011001001101000111111111001011000000 1010
01110110011010000100111001011011000100010001100101 1000
00011111011110100000011111111110101110000010000000 1011

1011011000000001100001101101010011111100000011000110 11
10110011000111100110101100111101110001001100000110100 1
100111101101110100110000101110101100110111100110010011
110001110111000100010100100100101100010010000010101010
000011110100100111010110010000111110100010100100110001
011001010110000000011000101000011001110111100000010110
000011101011111100101011011000111110110010111100010100
101010001000000111010001010001000101001000110101000101 0
110001100001100111001101101011001010101011001000110100
111111110101111111011100011110111010100010101011000 01
001100001001100001001100101001110011010001011101001010

n=55

0010001010101011000110011010001010101000100010110101001
0111000011011000110110010101000011101000010011000010111
0010010000000000000100010011010001010111001000100100001 0
1000000101101101101110101111110111111111101010100100010 0
1110011011001111110110001101010101010111110100010000010
1101000000001001001110101100110101101110100000000011000 0
1110010000100110111110001001011001000111100101101110101
0000000011100010111000101110011111001011110100000110110
0000001011111010000000000100000100101001111010010101101 0
1000010011111111001001000110001010011111001001111101000
0010101100001001000010010001000110111001000101110100110
0111110000000111110010100010000111110010000100011111111 0
0011010011010100101101110110100100010001110011011110001
0001111110101001111001101011000110010100010011100001111
1111100111110100101001110011001010100100000000010100000 0
1010010101000101011011000000101110001100100010101100011
1011001011111110011000111001011000000100111000000010110 10
0010110100000000111111101010101011111001100110001111010
0000010001101110100010101000111010111010110110110001011
1100000101001011011101001101001101100100110010011110100

n=56

10101001000110110110101110110110000001110110000101110000
00100110100110110101001000010000111110100011011101011111
01010010010101010000000100011010001011011110111001010111 1
11001110001110010001010100010001111001100100111010110100
01100110010011111000110011011101100100010001010101110100
01001011101001111000101010110111101010110000010110100011
00000010001101000010000101000100111110110011001110101010
01000000100111101111100100010010100000000100111101100001 11
00010010101010111101010110010101101010000110011110011110
10110001000011110001011010110001000110000101110111011011
01111101001100011111010110111001010000010101101010011011
11010101101010001001000110100110100101000011100010100011
01000100000000100100001101010010100011000110100100110001 0
00101000011000000011001001011000111111111111000111001001
00001111111100101110110000110001000110101111000101101010
11001001000101111011001100011110101100110001011011110001
10010011011011000100010001100111010010010100111001011001
11100100000011011011011101011001010111101110111101001110
00101001101110010100010011111101111100011110010110111101
11101011110100010000010000000001001110010011111010001110 1

n=57

001111111011001011000111111011101011101011110010000111111 0
110001011110010010011001101100101001011111000111100100100
000000001000000100111100111100101101100011001001111010001 0
010110001011010010000000010011111111001001011101111001001 0
101000100100001111110000111001101000101101101110001010010
100001110001001110101111011010100101111101011000010101110
011110110001011100000001010010100000011000000000101010011 00
000001001111100010111111001011001101101001101110000001110
011101001101010001101111111010000101001100000000000000000 00
111010010001101110000011000000101100110111111110111100011 10
010001110000010010101011010011101100010000000001101111100 0
101111101100100000001000010101110110100111110101100110010 0
111100110111010101001000011000001000111110000001000100010 0
001010011001110011101100001010100101010110011111111110100
010011111100100010110110110101111101110001001010000011110
011101101111110000000100000000000000100101110000000100011 0
000010101101010100111001001011111100010101110100010011100
100100000110110000111101011100111111011011111110001101010 0
111010100000110001110011101000110101100100100011110001100
010010000111010110000011000010011111000110000001100110111 0

Solução 103668.h

n=10
0100101001
0000000110
0010101010
1110010001
1110011010
0010100000
0000011101
1001111110
0000010100
0100110010
0101110000
0000001011
0101011011
0000000001
1101100000
0000001011
0000011001
0000010011
1100100001
1010110101

n=11
01101011011
00011110100
10001010010
00110011001
00000110011
01110001011
00000000000
10110110101
00100011101
00000011100
01101001001
11010101100
00100001000
00001010010
01101101011
10110011011
01101010010
00011110000
00110110110
00011011011

n=12
111011000100
010111100110
001001011001
000010110011
011001101001
001011001010
001000100111
010001100010
100001000110
011110100100
101110001001
101001010101
001011010011
110000110101
111100110010
010101100101
111000011011
110000000100
110101000010
111000100111

n=13
1111101101110
1000010100010
1110111111001
1100011000011
1011000011011
0101100010011
1011101110101
0101001011101
1101010000000
1001100010000
1100010001110
0010000111110
1001000100000
1110101011110
0011011111001
0000001100111

1111010110110
0100001101010
1011010110100
0011111001011

n=14
01110000010100
11101010001110
00001110001001
01010110110111
10100011001101
10001101110010
01001111011110
10001010110111
10111011000100
11001001111100
01110101000011
00010010001110
01010100011001
01110110110011
10111000100111
10111011001010
10110001100011
00101110111001
00111000111100
11011000000000

n=15
100010010101010
011001010100001
111000010000100
001000001110001
111101110010001
010100010101011
001011101001010
000110001100111
110110110110011
110010011100001
100101100010010
110011011111001
000000011001010
110000011010101
001001001000000
000000001000100
001001101101101
011110011111101
001101001101110
111011100110001

n=16
1100111100110110
1000100001000100
0111100010000010
1001000001001000
0110111010100000
1011010000010110
1001110101110100
0100000011110101
0110110000001010
0001110010000110
1110000011000100
1111001100001010
1001111100100101
1001100101010010
0100010001000010
1100001101110110
0011110010110110
0101100001001011
1011110101110000
0001010010000010

n=17
00011000111111000
10111111000000000
00010100010010001
01010001011110001
11011000101101000
11100100111110111
00001111100100100
10011101011110011
00011001000000000
01011011000100111
10100100111001111
00001011000000000
11010110011101010
10001101011000111
10111110101101100
00000010111011000
11000101000010111
10110110101110100
10010000111101011
11101001011100111

n=18
011110111110000111
000000000101001000
010100010011011010
111110100100101000
000001111011110010
000011101000111010
111101010010010101
101111000001011010
101000110100101011
010100011011100100
000100111011111000
110010111100000110
100100111110110001
101001101100110101
001010010001011000
101001111111000110
101010001011011101
011111110011001110
101110110011111001
001001001011110000

n=19
0000011001000111101
0010110001100001111
0100011010001100010
0000011001010010111
0000000001001001101
0000000101010001000
1111100000001100110
0001011110101001101
1111001010001000100
0000101001010110001
1001011101011011000
0001001000110000100
1100001111110111101
0001011011101100000
1000111011110000011
0100011000101101011
0001011110010101101
1001010110010001101
0001010001011011101
1001101010101000111

n=20
01011011010010100001
00110110101110001111
11111111000011111111
10001100101001010100
11110010001000110100
10101100101010101000
10010100111011010000
00100101000000100110
00100001011101011001
10110111101111010101
11011110111010111111
01100001101001110111
01011001110010010001
10110011100110001011
01011100010000101011
11000101001001100111
00000011001001001101
01110110000101000111
00011000111001110010
11010001110101111011

n=21
110101000101001011110
000001000011101000000
000101000110101001110
001011000100101001111
011010001001011100010
011001011010010111010
000000110001001000000
110001011101001010011
101000101001011110101
111000011011110010100
100000011101101111001
111001110101010010101
100110111100101010011
010011000001101110010
111011100110001001101
011101110000110110010
111000100010011101010
000110001001010010111
010110111001100101001
011111011010000110111

n=22
1000101111110010011001
0100110100110110001010
0000110001001001100010
0001100100001100111100

1100011101000011000101
1111101010101101111100
1010011110010100011001
1011111010001010010001
1011111101001100100101
0111010111101110001010
0101110000110111001011
0000010101000100100001
1001001110000011110101
1101111001111000110111
0001011010000101110101
1000111010100110010110
1010010010101001001001
1111000110000111110110
1000100101101000001110
1100000101100011000001

n=23
00111110011010011010010
10011101001110111101000
00010001011011001100111
10001111001001001000110
10001101110111110101000
00110011001111010011010
00000000000111010111001
10100011111000101011111
00000000000101011000001
10100100010001100111010
10000000100010100011011
11110110101110100101101
11000111110100011111110
01011011000101101110000
10001011101001100101100
10000001100011011110000
10101001100101110110111
11000110100011001010000
00000000000110010110100
00001011111100011110110

n=24
111110001100010101001001
010001001101010110101110
001010011101001100010100
011110011010110110110111
100001001000011110001100
101111011010111101010110
111100000000110010010110
110111101000110000011100
101011000011000111011011
111101000011000100110010
011000010101001000110000
001010011111101000111111
011001000101011010100101
111000100101110011000001
010111000001010110100001
111000000010000001110011
001000111011100000100110
101111101110100010111100
100111011010010011001100
010101101111001010100100

n=25
1011111000010111111000110
0100111100110101001101111
1000000101100100000100111
1011001101000001100011110
1101101000000011000010100
1010011010001000010010110
1000110011101100001010111
1111101010000100110001111
0110000110001010101100110
1110111110010110001001010
0001110110001001110111111
0111101100010100011100010
1011010101101111011110011
0010011000101110101101001
0101011011011010000000111
0000100011011100100101101
0000101100101100101010001
0111001010000000000100111
0111000010000100100011010
0000010011100100101000111

n=26
10111000110100110100100001
00110100100110011101111101
00010001101011110101110101
10100101111001010100110101
10001011101000010010001001
00101101100110000111101100
00101111010000100001011011
10000101010011000101100111
00111101011011001000000001

00110100111100011000101110
01101001000001000111001000
11110011101101101011110001
00101111110100101001110010
00010011000011011111110101
00001100010100111110100101
11110100110111001100011001
01000110000011011111100101
00110001000111010010000000
10010111100110111011001011
10100010010111011010000110

n=27
000011100111000110110001011
011001011010111110100011001
010001100111111100000101001
100100010011000110000011111
100111001010010101111100000
110000110010011101001110111
000011011100001110110011001
010111101101100100101101101
000101001110010111101111111
010100011000111011100000110
001011101101010100111001011
101011100011111000001010110
111101011110101100011001010
110011100011100001010100100
101111101010101010000100011
110100100110011000000000101
011100101011100110010110101
000101011001111010101010110
100000000100111101101101100
101100001011001110000110001

n=28
1101100101000110000010010110
0011000111100111111110011001
0111110111101001010111101010
0101001111001101000110101011
0000101000110111000010010000
0111001001011101100111101101
0001101001010010111110000111
0000011101000100000001100001
0000000001011111111001010010
0111110110001010000100010010
0101010011010101000111111010
0010000001011000111111110001
1001010101011010000111100001
1000010001011101110011001100
1010101100011101101100010000
0100011101110010111011100100
0110100110110110100110010000
0110011010110001111010011011
1011000000100110010101001001
0110011100000101010110010001

n=29
11110011000011101111100101001
01011001101101011100010101011
11100100010111111011000101100
10101101000011010110110011001
10100100011111110001101100110
00001110011011111010111001011
11001110001110001011000101010
10000011110001101110110001001
00111001001011111110110011110
01100100000011001010000110110
11011100101111101110111011001
11100100101111111001001100001
11101110110110010100110100010
01101101111000000001100101001
10100010111001100011011010011
11101110011111001100011100100
10100000010010001110111100011
00110011000010100110000110000
10011110101101100010110000101
01001110000110001110110110100

n=30
111001111001011011010001110101
000011101000111100000110100100
001000011101001001010110111011
010000011001001010111011101011
101001111101101001110111101110
011111001101001011111001000110
100100101110100110111110000011
111001000000110111100111101000
010111100001001110111111010011
110010100110111111010011101000
001001011100110011000110111101
001000111100011101001001000011
010110011010010111111110010000
100101110111001000100111110010

001110010101100111000011001010
110001000011010101010111001100
010101101100001011100111100000
000000010100110000100010111010
001001011001111100101101101000
011010100000000011100101110100

n=31
0001010011001000101100000111001
1001001001010110011101001011111
0100110110101100111001110101111
0000110011110010001001000010001
0110001010111110100000001010101
1111110110000010010010101100010
1001011111010001101111001010000
1000010010101100101010101111101
0011100000100011101101011000101
1000110101011011100101001110001
0000110001101101100011100101100
1111101000110110100101010110100
1010010010011110111111010100000
1001100101011000010010111010111
1010101000110010011111100101100
0100000100101000000101111011100
1010000011111111000001000110001
1111101000010101110100101110111
1101101010110011110101101101110
1010000010001100110010101001000

n=32
11111111111111100111001110100111 1
01010111100101110111100010101100
10110110000111010010110100111110
01000000101100000000101101100101 10
01101100001110011111010000010000
10100011111001011010101110000011
00000001111111100000011010100111
00000100000010100010100000111110 0
11001111010101111010010100110001
10000101110111000111001101000100
00100011101101011011110100011000
00001001011000001001110000001101
10011100110111010000001011010100
01001111010000011011000111101110
11101000100001100100101110110001
10011110010010111011101011110100
11000110110010001000001011000110
11101111001101010100011101110110
10011111001011101010110001111100
10110101100110001100110011111101

n=33
100010111011101010100001110101001
100010011011110101010111111101001
100110110000011000010010110110110
001101011101011110101010100011111
110011010001111010111000000111101
101001101011001110011100010111001
111111111101111000101010000000111
000100001100110101111000111001011
111010000011001010100100010100111
000111110101100111101010111000110
110111011110001000010011111110010
001101111110110100101000111011010 1
000111000100011000000000010100111
001001010101001000011000010000110
010011010010011110011111010110111
110011100011000010101101110000100
000011011101010000111111011110110
100100011101111111010100101101111 1
011010111000010101100011000000001
111101100001001001111000100101101

n=34
0111000101001001001111111011011111 0
1101101101110101001011111111000111 0
1001000101010101100010011010111101
1110111100010010100001111111010100
0001110101110110000010100101011011
0010101110110100010011101000101100
0100011001011001110111100011011010
1101110111100000011110101111111110
1100001101100011111011000100110111
0101100110011110011100110100111101
0111110101000011100010000001011100
0101111000101000000100101001000010 1
0101101001100111010010011010111100
0111010101000111111100011011101111
1111010011000101001111100110100000
1111011010010010001101000111001001
1111010001000010010101100000101100
1010011001011111001011111110100100
1010011010011000000111100111101111

01010100001110100110010010000110011

n=35
00101010100011010110000110111111101
00011000101111110010010101100011101
11000010010110011011110000111100101
00011011010001000110101010001101100
00101101101100110101000010011000010
11101100111000011001010101110011011
01001111010100000111101010000111000
01000001010101011101110100110100100
10110111011100001101111111001111100
10111010001010101001000001001100111
11101010001101011011111111111100000
11001101110111001110110001010000010
00011011010101101110001110110000100
01011000101100011010100110010000110
10100100110000101000000001101110000
01100110010000000101111101011100100 1
01111100011000011010000101011000101
00001000111001011010100111000110111
11001011110011000101100111001000010
11100100100101010010100101011011010

n=36
010000000010010000111001110101001010
110111000101111011001010001011110110
110000010100000010000101110011101110 1
111010101011111101000111001110011101
111000000100101100101100110001011110
001001001011101011101010101110010000
000001000001011001110010111001010000
000010110011100110000010000000010010
101110111101111110001111011000000010
110010000100110101010110010100111101
000101011000011101000010011100001111
101100010111011100001101110101100000
111100011000110001000101001101001001
101101101011010000001010110100110011
001101101100000111000110110100010101
001101100010001000110010110001111100
111111000100101111001011010000011000
001000101101000000101001011110110101
001110111001011000000001110101100010
000100011011101110010101011010011000

n=37
1101000000110011001010100011000100100
1100110111001111000110001001110000101
1011010000111100001100000000111000000 1
1011010001000000101111010011100100000 0
0111010011110111010111100001000010100
0100111011111110001011100011010011010 0
0011000100110111101011000001010110100
1010001001000011000000111100111111111 0
1010011110100110110010110111011011011
0111111110010011011101111000100101001
1111001100110010001001010010111011010
0011001110100111100001111011101100100
0100011010100000101101110010101100010
1101001111101101011111011000000100010
1100001110000000101100110110001101100 1
0011100110011000101000110010100101011
1010110010111001001010100011101000010
1110100000111010100000001101111011010 1
1011000010010111001101000000001010010 0
1010000000110111100110100001010100001

n=38
11100000011000100001011000010100001011
11010010010011010101000100010101111011
11110010000010011001010010101011001110
11010111111000011111110001101010101111
11000101001001110101111100011011100000
00100100000010000000101000010111111001 001
01100100101111010010111110110011010000
01000100101111111101100101001010001110
01111010101000110010010110000111100001
00001100110100101101111111000010000101
01000110000111011111100111111111000110
10111011111001100101011110000110101101
10010010001010101001100001010011101001
00101101110110111101110001011001011000
11011101111010011101111011010010100100
11111111011111000100000000101010001111 0
11000100110110011100100010111010001010
10110001001100111110111111000000110010
11111000000101110001010011101111011001 0
10001011010101101100111000111001001010

n=39
110101100000110001010110100101011010010
011100101001111000000100100011010010101

101101011011111001000000110111111110001001
1010010010011101101001011101011111101100
110100111111001001111000110001001100111
1011010111110001010000000110111001000000
1011011110010001100011000111111001111110
011011001101110001100100110101111000011
1001001010010000000001110110010000110001
000000100101110111000101001101101110101
111101101101001000111111010000100111001 1
0011010010011111001110101001111110000010
1110101101010000101000000000100110101110
0000000111001000000011101100011010010100
0110101010000100001101001101101000010110
0100011111110100001011011011100100011 00
1100100111101010110001111100011111101001
110111010011010001001100101101010110111
1000101000010000110001010011001101111010
0010111100011110100111001110001010010 10

n=40
10100111000111111111010110100010010001 11
110010000011101100011000001011101110111 0
011101110010100111001110011011010101101 1
001011011100010011001000000101011000001 00
111010111010101110101000000010111010100 1
111011001000000100010111000110110010011 01
111010010010010110110001010101011110010 0
000100110001101100101101000101001100111 0
001010111001101011101011001101000100001 0
000111100010010111010111000010001000010 1
010111000101101001101011111000110111011 1
110111100001011110010001000011111000110 0
111100001010000010010000101100111000011 0
011101111001111000100111100110111000001 0
110100111111000110101111110100111111010 1
110001000000000111100011100001011010100 10
000010000000000111011001011010111110101 00
100011000000101100111000000000111111111 000
000110111010100011000001000000001100010 00
101110111100001101101110110000101101001 1

n=41
01000010100000110010110101111010110111001
01100110100001110111111010000100101100000 0
00101011000110011000011110010011100011110
11001011010010000011111111111000000010100
10100111110000011001001101111111010100000 1
11110011110010010101111100001110111101111
10111001000111111110110001110101110110011
01000101110100110101100111010101011111101
11011101111000101110001100000000110111011
01011111010000101001110110100001001101111
00000001000111100111111111101011000110000
00110110001110001100111111110010010111001
10001110011010110101000110100010000100000
01110110011111110011110101011100111100100
10010101011001111000110000111110101011111
10100010011110110100011100100000101101101
01011111101100111000011110000001001100100 0
01101000010100100000001010110110001010111 1
01110111011100101001101100001010001100010
01100011010111101110111110010000001111000 0

n=42
001101000000000010100001010111111010101100 01
011000111000000101101011111001010110000011 1
000011101110101011100100111110001110011100
001101001100001110010000100100111111110011
000001000111111110100110100101101010111111 1
101111001010100110111011101110110011001101
101111000111001101001010101111111001100011 0
100010010101101011001110001101011010111001
000100010010101000101100010001101011011100
111110101101011100011111001101011110110100
110110110011111011001111111101001010001110
100110000111010101100101001100110011010110
100011110011110000000101110111011001101000 1
100011100111100100101011011010110101001010
000011010010011010110111110011111001001111
001100010100110111000000000100111110011011 1
110110001001010001001001011101101101100110
110011100000011010101000010011011001010110
101000000101110100110011111111001011011010 0
010000011011100010011010101101001111000001

n=43
1000101110011001101111111111111111011010000110
100001110010110110100010001010100100111000 0
101001000111101110010001110110000000101001 01
010110110001010010111000011000111100011100 1
000111001010110110010110001000111110011010 0
100001111100010111000000010100101110001100 0
000011111001111110111110100000011101011011 1

00000010000111110011101011001000100101000001
10100100110100000100111000101110000011100000
10101110001100011100101010001000010111101111
11110011011011011111000001010010100111000000
01001010100010100010011000001111111010100111
11001011011000000111000110001011011110000110
01101010001110111010110100000110110000010110
10110101010011001101101001100010000001011101
00000001100011011000010110001100101011111110
00100111010100110011110000100010001100101111
01110000001000011101101111111000011110100111
00100111101100110100011111111110111000100010
10000110101001111100111101100100100100101111

n=44
001000000111110001100010100110100101110111110
111011100111010001010101101111000100001111010
010101000011111010011111010011000101110111010
001110110001011000001111001101010110100111100
010010001001110010111011111100110000101111110
10010111110110111111000100101011110110101111
011001010001011110111101101101001100110001000
000001110010011111111001111111010101111011100
10100001011100101011100111100010111110111011
011110010011010111011000110010101100100001100
00001110100101100110010000101001001011111101
00000100011100010111000110110110111101110001
011010111011000110101101001111111111110001100
10001111111001100000100110110000100011001111
010110011000011011101100001011010000110001100
000100011011011111011000110011101001101001100
111011000001110100010000001010000010110110110
111000111110110000110010110100001110101111110
100001001000000010001011111011100100110110110
100110001010110110011000011000011001000010010

n=45
111101110011010011001110011010000100011001101
011110011111001110001111100110101011001100101
110100101111010101010011100000001111011001010
001111010011100111011000000000011100000100101
100011111011100010110000100001100101111110110
100001000000010010011000110101100100101100110
111100100011001111000111001100110010110011001
000010101110110001011000001001010000111000000
000110001111000001100010110001111111110001110
100110001010111110000011100100110100001010010
001011100000101000101000010010000111111110111
100000101111101111101001100101110011101111100
000000001000000110010000110010111011110110110
001010110010000000010001010001101101110011100
110010111111110111001010011011110110101110001
011101111100101101011110100010100110001010010
110000100110001100111110011111001010100111101
110011001011100110000000010000110111100001001
111111111100001011110010111011011000111000100
011101011001010011000011110111101111100011001

n=46
1010111000001010100111110100110101110101110111
1100001100101110101011101000000000000101011100
0101000000111110101011101100001001100101011011
1001011111010111000011110001100001101110000111
0110101110110010101010111010001010101100110100
1011011110111011000010001001101110011111101010
1010011010110010011001011011001110000100010010
0100101110100000001101101110000101000010110100
1110111100001101111100110100001101000001000100
1100001001110101111110111111000101001001100111
0101111101111100001101010000110101010001110001
1010010010010100101110011100010001000101110101
1111100000011010000001100111111011010110110110
1110001011100001000101100001111100100011010011
0110100101100011100001000100110000110101000110
0101111010111001110101010100010100111000111000
1000001001000101110110000100101100111100111101
0011110010110011111010010001001101100010111111
1110001010100000101100100000000011101101100011
0111011101011001011101001010111001010001100111

n=47
001101001000100101110100100010111111111101000110
111111101000001101010010100100111001000111011111
111101000100110110100101101000110110100011000010
101000101100111101000110111001100111001101101110
001011011010101111011011000100111101011101101111
001001110110011000011001010100111001011111010010
001101000110000110110100101101101100100101110000
100100000010110110011111010000110001100100011001
000110111011001110011110101001010010011010001000
101011001111101111010101001101111110101000100010
000101110011000101111101001001111001011010010000
110000000000100100010100011101101000101001100101

11010101010111100101101001011101001001010011000
10100100111000011011111101000010000110011000000
10101111101100011000111010010101100110111110010
11000000101010110010100010101001011101010000100
10111101011101111011011111001110001011110000010
111100011011100101001000010000001101010010111000
110000000001001100110001001011001011101100101110
11001001011100100010111001110111110011100100011

n=48
001000100100011010011011100000011011100011111110
1100100000001100000001110101100111011000110110110
001010010000000100011110010001010100100011001010
0000111110011110110011110101000100000010001011110
111000111011010110000100100110101011000111010100
010100010010011011101000110110111101101101110001
111110000101111000111001010111101110100111010110
0110101110011111101100001111011111101000011100011
100110100100110110111001111010001111001011100100
001101010111100000110100101011000110010010010010
0001000000001111001011000010110010111101110000111
111000110110000101100110000110111101100111011100
0011110111000100010000001110001010000111000111100
1010010101100100000011100110011110001110001010111
1011111111100111011011001100111110011001000101010
110010010110101101110011101001100010101111001010
010010110001001001010100100110100100101010011000
1101111101111110111000001010010101010001110010100
011100011011011101010111111110000101111000110010
100100001011100110101101001001010100000011100001

n=49
0110110011010010010001001110110001000011111110100
11000000010100001100001101101000010000110001100110
0001001110010111011010111101010011100011001110000
01001000000010100011011000111011011100111111110001
1011011010110011101110001011001110100110011101001
00001100001101110000000101100000101011101011011001
0100101000010101011011111011110001100001000111101
1011101001001000111111100100110001010110110011101
11000100011001110101100111001101000000001000110011
0101110111110101100110001100111011111010101100111
0010010110001010010111101100101000101110011111101
01001101000011110100010101111000000010110010110110
0110111111000101111011000010010101011110010110111
0101010110111100011101111101010000111101101000010
01000011110000001110011110010010100000001110001110
0110010001101000110111010101010011110001011101100
1101000001101100110001100111011101011001001110101
1001101111110011001111111000100011011100010000001
0111111000100111010000110011101110100110010101010
0101010001100000111110111011001101101001001100011

n=50
00011100011000000101100100111010011011010001010111
01111100110010000111100011100000111000101100011000
010101011001000110001000000010111110001000011110110
000110101111010011111000000010110101011000111110011
01101000001100001110000111011010110100101010100000
10101011011101010010001110010011000011101101100001
10100100001010011000001011110101100011011110000010
011111000110101110011010000001001110110010001101100
11000110101111111100110011110101010110100000001000
01000101111011100110010010111100001101101010000111
010001000000111000110101111100101001010000011111110
11101100100110110111010001011101101100000001110100
11101111010010011011000011101101011101000101100111
10010111110101000100011010110100100000100111100111
001000000110000001000011001010110001110101101001101
101010000001100010010101010100110111110100000001111
01110111111000001101111010010110010100100001011000
0000001000110011101010000000001000010010000000010010
00111110010111010100100101001011111001100111110100
01110001011010110000101000110010110111010001111111

n=51
0010000001001100001000110000100110001100011000011011
0010001001011111100101101111001110110100111011100110
1010000001010011011001100100011110110010010101101011
111111100111101110101000111011011010101100111111001
00110100010100101110100000011110000100000001111101111
00000010110000001001100000010011101001010111101111001
1001110000111010011010100100001110001011101111101000
0000011000101101001010000101110110111111111101010001
100001100100000100000101110110010000001010001010011000
010010100010001001110000011010100011011110101000011
1110011100000011010110001001100001010101010000101100
0100111011000111000011001011001101101111111111010000
111101110010110000111010101010111011101001110010101
110000110100100011001100011110010100100011110100100
0001010010000001110111100011100001010111001111010101
000001011101010101110011100101110000000011101100111
0110110111100000001001010011100111111001011010110001

11101000111000011000011001000111100111001100111100
10101101010110001110101000010100111110000100010001
01001100001011010001110101001001010001111101001011

n=52
11011100011011001100011001001001001101011010010001
10000101110111111101011000100110110001101011010110
01111011010111010000110011100101011001010010011110
10110001101100010101000001110011001110100000011011
01011010000011011011010111000001111110000001000000
11000010101111011111110100010001100100011000101111
11011111101010110011001110111001011100011101100001
11101000011111011100001001001001011011000100111010
11100100001110000101100111010110100010010111011010
01010010000111001010010101001100101111110111011010
11010000001001000010111101111101011000011000000010
11001011100110000011001100001001001001011101011001
11110101111100101011110011100001101001001001100011
11010110100100111010101001110110100010000101111111
11011000001111010110111001111111111110011011110010
01010110111100010111010101010001110001000101101001
11110111111110111001000110011100110010100010001101
00110110011010001110110100111110010001101010101011
10000110000101110100000000101100000000111000101011
01101101100011111000001111000000110111001110111000

n=53
10110110100011011001011000111101011110010100110000
01010011110111000011001111011111011001001111110010
00011100111011100101000111010011111100011001111101
00110110100101111010011111000111101000110111111001
01101010101011111111101101010100111110011001000011
00110010011000111001011010000001000110111110110111
00010101110101100111001111001010010101100010110110
10010111101110010101100111100011111100101011100001
11011100011010111100011101000011001101110001101101
00000101110101000010000000000001011111101001110010
00001101000011101011010111110101111100101110001001
11010100110111000011111101111110011011100100000000
01000101110011111001101111110110000000100001101010
00001111001101000010010001100000000001010111010001
00001110001001110011000000010010100011001101100000
00000111100111101011101011000101011001001011010010
11001011010010101010010011001111111001000010110111
01110010111100011100010001111011100001001111000101
00100111011100011110000001100100000110010111001100
11011101101001100000110111000000110010100101010001

n=54
01110110001001001000011011011000110000000100010101
01000111101111111111111001100101100100011010101100
11010010010101101010011110110100101011101110011010
10010111111001111010111000000000101111100101110011
11011010000010101101010001001111010110111111011011
01011011010001111000000101011011100110100011111110
00111101000011111101001101110011010011111101000110
10010110010000011110011000010101000000101011001000
00001011001011001010000001110110111100010101100000
11111001001011000000101011111010111111011100000011
00000001011110111111011010001110000011100011011110
11000111111000011010011100011101001010010110111101
01010000011010011001111110110100000010000000000100
10011100110101111000100000011011001000110001010110
00110101011011111111011100101010111000101011101101
11111011001100000001011001001110001110110010111011
11011001001110100010010001010101000101000110000011
00100100111001111110010010010010101000101011100000
11001110000110000101011111000010100111101110111111
01001011110111101111111001000101001110110100011001

n=55
01110101001111100011101101001101010011101010011101
01011001010010101101010111001111000000101001000110
11010110100101110111000011100000100010001100110010
01011110000111001110110000110101011001001100101101
10110001010101011011011101000100011101100001010101
01101101011000101101110010110101011010111011101010
10110001001011111001000101101010000111010011101000
01100010111111101111100100010000111010111000111101
11010011110110111100011110001101011111000101100100
01010111110010000000101110001101110010100000101101
10100100100000000100001000101100010010100100111000
01101111100011001101001110011110000100000101001110
00011100110100011010000001010111001110001110101110
01111100011011010100111011000110011100001110011110
01101000111110110111110010010101111011111001011100
11100011011010011101101010111100000000010001101000
01001010100001010001101110011011011001101110010011
11010011011100111100010100011100011011100010001010
01100000110111100111001011001001101010101111001101
01001011111110010101110100111101011010100100100101

n=56

10100111110001000010011110101010000110101111111110110000
11001001010001111001101000010111010010000110011101100111
00111001100111101100111001000101001000101000001010001001
00100100010100101010000101010100010100011101111100011101
00010011100011000010101010001011001111010000001100110011
01111100011000010111011100110011100111100101010100000001
00111001011010101110111110110110001001100111001011110111
00010010011101011010001001111101101010110000010111010011
01001110000110011011111001001000010011111111111110001100
01110001110101110101001001000111100101011001000011111010
11100001100010010011100011111001000001101111110010111011
00011000111000110011111111000000000101100110110101101101
10001010010111001010010000101001000000001001100110011001
00000001110110010100010101110100010000100010110110001011
11000111110100111010110011001110000010110101101011010000
01001001111001100111100101110010111100011010100010010101
00011011010011101111101010000010000000111100011010100100
11000100000111011110000100001000101011111000100101010001
00101011111011110100001010111110001010001000000111111001
10001111111110010001011000011100101101111000110010000001

n=57

110011110101101110110000000100000111011100101100111100100
101010011101001010011001111111100001001100011101000101101
011101011111110000000000011001100001110011010111111101010
100101011011101101110011110100100101000001100101000111110
000111100110101001100100100010000010100000110001011001010
100110111010111011000111001110111010111010000000010100110
001011100100011100010101000000010000110001100111111110111
100001100000110101101000010110111100110101111111011111001
110011111110010100000011100101001110010001000111101000110
001011111010101010110011001011110100101001100101111100011
111101010101111000011000100101101101000100001010011001011
111010000000000100001000110011101110001101010010111001100
100001011101110111000001010110101101001111110111100101100
111000110101011011100011010110010101101100010100100101010
101010101110010000111011100111110010010000100100011011011
101100110110000101110101110011000100000000101010000110001
001000000111001110010000111100101100011101011001010111100
001011000010011111111101010100101110010100001110111101011
101000010101101010001101100010010010110101101100111100101
000010010010010000001001011001101111111000000100101000001

Código Desenvolvido

Código Desenvolvido em C

```c
//
// AED, November 2021
//
// Solution of the first practical assignement (subset soma problem)
//
// Place your student numbers and names here
// Student 1: André Butuc (nMec: 103530)
// Student 2: Gonçalo Silva (nMec: 103668)

#if __STDC_VERSION__ < 199901L
# error "This code must must be compiled in c99 mode or later (-std=c99)" // to handle the unsigned long long data type
#endif
#ifndef STUDENT_H_FILE
# define STUDENT_H_FILE "103530.h"
#endif

// Para manipular os algoritmos que são corridos.
//#define IterativeBruteForce         // -> Ativa iterative_brute_force se descomentado
//#define RecursiveBruteForce         // -> Ativa recursive_brute_force se descomentado
//#define CleverRecursiveBruteForce     // -> Ativa clever_recursive_brute_force se descomentado
//#define HorowitzSahni               // -> Ativa horowitz_sahni se descomentado
#define SchroeppelShamir            // -> Ativa schroeppel_shamir se descomentado

// Para manipular as versões HorowitzSahni e SchroeppelShamir
#define QuickSort                   // -> Ativa versão QuickSort em Horowitz e Schroeppel se descomentado
//#define MergeSort                   // -> Ativa versão MergeSort em Horowitz e Schroeppel se descomentado

// Para manipular se há ou não escrita em ficheiro.
#define WriteFile                   // -> Ativa escrita em ficheiro dos tempos de execução, se descomentado.
//#define PrintTime                     // -> Ativa a impressão em terminal dos tempos de execução, se descomentado.
#define PrintOnlySolution           // -> Imprime só as combinações no Iterativo,
                                    //não ativar quando não estiver a correr o IterativeBruteForce ou não ativar
                                    //quando não está ativada o OutputSolutionsRC ou OutputSolutionHS

// Para manipular se há impressão ou não das soluções no terminal.
//#define OutputSolutionsRC       // -> Ativar para ver as soluções dos algoritmos recursive_brute_force ou clever_recursive_brute_force, nunca ativar com outros algoritmos.
#define OutputSolutionsHS         // -> Ativar para ver as soluções dos algoritmos horowitz_sahni ou schroeppel_shamir, nunca ativar com outros algoritmos.


// include files
#include <stdio.h>
#include <stdlib.h>
#include "../P02/elapsed_time.h"
#include STUDENT_H_FILE

typedef struct {
    integer_t mascara;
    integer_t soma;
} mascara_struct_t;
```

```c
typedef struct {
    integer_t mascara;
    integer_t soma;
    int indice1;
    int indice2;
} heapStruct_t;

/*

Seria uma boa ideia colocar este algoritmo no relatório de forma a dar entender ao professor
o nosso algoritmo piloto que nos permitiu estudar o problema e começar a produzir soluções
É possível calcular as somas até n=27.
É também interessante mostrar o gráfico desta solução porque o best-case e o worst-case irão ser bastantes diferentes.
Por exemplo: Para n=28; Best-Case: 1.4 segundos; Worst-Case: > 15 segundos;

*/
// ----------iterative_brute_force---------------- //

int iterative_brute_force(int n, integer_t p[n], integer_t desired_soma)
{ integer_t test_soma;
  for (int comb=0; comb<(1<<n); comb++)
  {
    test_soma=0;
    for (int bit=0; bit<n; bit++)
    {
      if ((comb & (1 << bit)) != 0)
      {
        test_soma += p[bit];
      }
    }
    if (test_soma==desired_soma)
    {
      for (int bit=0; bit<n; bit++)
      {
        if ((comb & (1 << bit)) != 0)
        {
          printf("%d ", bit);
        }
      }
      printf("\n");
      return 1;
    }
  }
  return 0;
}
// ---------------------------------------- //
```

```
O algoritmo recursive_brute_force, como o nome indica encontra a soma recursivamente. Em cada valor de p há um "branching": o bit do valor de p fica a 1
ou fica a 0. Em cada ramo, a soma parcial (partial_soma) é incrementada (no ramo do bit igual a 1) ou mantém-se igual (no ramo do bit igual a 0).
De seguida é incrementado o indice do array p e é chamada de novo a função recursive_brute_force.
As condições de paragem da função recursiva são:
  - if(partial_soma == desired_soma), que significa que foi encontrada a solução. Retornando 1 irá desencadear o desenvolvimento das
  sucessivas chamadas prévias da função. Sendo que o bit da folha que desencadeou a terminação da função com sucesso irá ser
  armazenado no array b e assim sucessivamente até à raiz.
  - if(current_index == n), que significa, caso verdadeiro, que a recursividade chegou ao fim do array p e como já não
  há mais elementos ou combinações a somar e como ainda não foi encontrada solução, retorna o valor 0 que por sua vez
  indica que não foi encontrada a solução-soma.
*/

// ------------recursive_brute_force----------------- //

int recursive_brute_force(int n, integer_t p[n], int b[n], integer_t desired_soma, int current_index, integer_t partial_soma)
{
  if(partial_soma == desired_soma)
    return 1;
  if(current_index == n)
    return 0;
  if (recursive_brute_force(n, p, b, desired_soma, current_index + 1, partial_soma + 0 * p[current_index]) != 0){
    b[current_index]=0;
    return 1;
    }
  if (recursive_brute_force(n, p, b,desired_soma, current_index + 1, partial_soma + 1 * p[current_index]) != 0){
    b[current_index]=1;
    return 1;
    }
  return 0;
}
// ---------------------------------------------- //

/*

O algoritmo clever_recursive_brute_force, é semelhante ao algoritmo recursive_brute_force, no entanto, apresenta mais
dois argumentos de entrada que servem calcular uma soma a contar dos últimos indices para os primeiros.
É feita esta soma simultanea de baixo para cima e de cima para baixo do array p para que quando os current_index e current_last_index
se "encontrarem", ou seja, quando forem iguais, podemos avaliar se é possível realmente chegar à soma-solução ou se o nosso array de valores
tem valores demasidado pequenos para alguma vez chegar à soma solução.

As condições de paragem da função recursiva são:
  - if(partial_soma == desired_soma), que significa que foi encontrada a solução. Retornando 1 irá desencadear o desenvolvimento das
  sucessivas chamadas prévias da função. Sendo que o bit da folha que desencadeou a terminação da função com sucesso irá ser
  armazenado no array b e assim sucessivamente até à raiz.
  - if(current_index == n), que significa, caso verdadeiro, que a recursividade chegou ao fim do array p e como já não
  há mais elementos ou combinações a somar e como ainda não foi encontrada solução, retorna o valor 0 que por sua vez
  indica que não foi encontrada a solução-soma.
(Até agora iguais ao algoritmo de recursive_brute_force.)
  - if(partial_soma > desired_soma), a partial_soma já é superior à soma-solução logo nunca irá ser alcançada porque os valores de p
```

```
int partition(mascara_struct_t arr[], int low, int high) {
    mascara_struct_t pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j].soma < pivot.soma) {
            i++;
            swap( &arr[i], &arr[j]);
        }
    }
    swap( &arr[i + 1], &arr[high]);
    return (i + 1);
}


void quicksort(mascara_struct_t arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quicksort(arr, low, pi - 1);
        quicksort(arr, pi + 1, high);
    }
}

void insertion_sort(mascara_struct_t *data, int first,int one_after_last)
{
  int i,j;
  for(i = first + 1;i < one_after_last;i++)
  {
    mascara_struct_t tmp = data[i];
    for(j = i; j > first && tmp.soma < data[j - 1].soma;j--){
      data[j] = data[j - 1];
    }
    data[j] = tmp;
  }
}

void merge_sort(mascara_struct_t *data,int first,int one_after_last)
{
  int i,j,k,middle;
  mascara_struct_t *buffer;
  if(one_after_last - first < 40)
    insertion_sort(data,first,one_after_last);
  else
  {
    middle = (first + one_after_last) / 2;
    merge_sort(data,first,middle);
    merge_sort(data,middle,one_after_last);
    buffer = (mascara_struct_t *)malloc((size_t)(one_after_last - first) * sizeof(mascara_struct_t)) - first; // no error check!
    i = first;
    j = middle;
    k = first;
```

```c
      while(k < one_after_last)
        if(j == one_after_last || (i < middle && data[i].soma <= data[j].soma))
          buffer[k++] = data[i++];
        else
          buffer[k++] = data[j++];
      for(i = first;i < one_after_last;i++)
        data[i]= buffer[i];
      free(buffer + first);
    }
}
// ---------------------------------------------- //

/*
void soma_all_subsets(int n, integer_t p[n], mascara_struct_t result[1 << n], int level, integer_t mascara, integer_t subsoma, int idx)

O algoritmo void soma_all_subsets é um algoritmo recursivo que procura encontrar todas as somas possíveis e as correspondentes máscaras
das somas. Parte do mesmo princípio dos algoritmos recursive_brute_force e clever_recursive_brute_force, no entanto como sabemos que a função terá que percorrer todo
o vetor p, a única condição de paragem é if(current_index == n) que significa que o current_index foi incrementado até ter chegado ao último indice do array p.
No entanto, como sabemos que teremos que guardar todas as "máscaras" das somas, quando a função retorna todas devem ser guardadas.
Devido a este fator desenvolvemos uma estrutura que guarda tanto a soma como a máscara:
typedef struct {
   integer_t mascara;
   integer_t soma;
} mascara_struct_t;

E agora nas sucessivas chamadas recursivas já não é avaliado o valor de retorno, sendo este sempre void, portanto o armazenamento não é feito por indexação
num array que guarda os bits como nos algoritmos anteriores, mas é feito com o auxílio da variável mascara que é representada por bits, que é inicialmente 0.
Em cada chamada recursiva é feita uma operação bit-wise OR do valor da mascara com o valor de 1 bit deslocado para a esquerda de current_index bits, caso
o bit do elemento de p seja "um".
Vamos supor que temos o valor inicial da mascara 0 e que o elemento que estamos a avaliar tem indice de 5 no vetor p e vamos colocar este elemento com bit 1,
ou seja, participa na soma parcial. Então mascara | (1 << 5) => mascara | 100000 => 000000 | 100000 = 100000  (o valor da mascara é extendido com 0 mais significativos
até se igual à dimensão com a qual estamos a fazer a operação bit-wise OR, o mesmo acontece inversamente).

Para não haver conflito quando quiseremos guardar elementos no vetor somas (que é o vetor de resultados), quando há o branching do ramo que iguala o bit do elemento a ser
avaliado a 0 (não participa na soma parcial) é incrementado o indice dos indices impares e o ramo que iguala o bit do elemento a ser avaliado a 1 (participa na soma parcial)
é incrementado o indice dos indices pares.

Quando o indice chega ao final do array p a recursão pára e todos os dados são guardados nos campos da sua estrutura na posição index no vetor soma.

*/

void soma_all_subsets(int n, integer_t p[n], mascara_struct_t somas[1 << n], int current_index, integer_t mascara, integer_t partial_soma, int index){
  if(current_index == n)
  {
    somas[index].soma = partial_soma;
    somas[index].mascara = mascara;
    return;
  }
  soma_all_subsets(n, p, somas, current_index + 1, mascara, partial_soma, 2*index);
  soma_all_subsets(n, p, somas, current_index + 1, mascara | (1 << current_index), partial_soma + p[current_index], 2*index+1);
```

```c
// type == 1 -> min_heap      type == 2 -> max_heap

void push(heapStruct_t h[], heapStruct_t elem, int* h_size, int type)
{
  int i = 0;
  for (i = *h_size; i > 0 && ((h[(i - 1) / 2].soma > elem.soma && type == 1) || (h[(i - 1) / 2].soma < elem.soma && type == 2)); i = (i - 1) / 2)
  {
      h[i] = h[(i - 1) / 2];
  }
  h[i] = elem;
  (*h_size)++;
}

// type == 1 -> min_heap      type == 2 -> max_heap

heapStruct_t pop(heapStruct_t h[], int* h_size, int type)
{
  int i, c;
  heapStruct_t elem = h[0];

  (*h_size)--;
  for (i = 0; i * 2 + 1 <= *h_size; i = c) {
    c = 2 * i + 1;
    if (c < *h_size && ((h[c].soma > h[c + 1].soma && type == 1) || (h[c].soma < h[c + 1].soma && type == 2)))
      c++;
    if ((h[c].soma < h[*h_size].soma && type == 1) || (h[c].soma > h[*h_size].soma && type == 2))
      h[i] = h[c];
    else
      break;
   }
   h[i] = h[*h_size];
   return elem;
}
// ---------------------------------------------- //


// ------------horowitz_sahni--------------- //

int horowitz_sahni(int n, integer_t p[n], integer_t *solucao_mascara, integer_t desired_sum, int current_index, integer_t partial_soma){
  int n1 = n/2;
  int n2 = n - n1;
  integer_t p1[n1];
  integer_t p2[n2];
  int length_a = 1 << n1;
  int length_c = 1 << n2;
  mascara_struct_t *a = malloc(length_a * sizeof(mascara_struct_t));
  mascara_struct_t *c = malloc(length_c * sizeof(mascara_struct_t));
```

```c
    for (int i=0; i<n; i++){
      if (i < n/2){
        p1[i] = p[i];
      }
      else {
        p2[i - (n/2)] = p[i];
      }
    }

    soma_all_subsets(n1, p1, a, 0, 0, 0, 0);
    soma_all_subsets(n2, p2, c, 0, 0, 0, 0);

    #ifdef MergeSort
    merge_sort(a, 0, length_a);
    merge_sort(c, 0, length_c);
    #endif

    #ifdef QuickSort
    quicksort(a, 0, length_a-1);
    quicksort(c, 0, length_c-1);
    #endif

    int indice_a = 0;
    int indice_c = length_c - 1;
    integer_t soma = 0;
    integer_t solucao = 0;
    int ret = 0;

    while (indice_a < length_a && indice_c >= 0) {
      soma = a[indice_a].soma + c[indice_c].soma;
      if (soma == desired_sum) {
          solucao = a[indice_a].mascara | (c[indice_c].mascara << n1);
          ret = 1;
          break;
      }
      if (soma < desired_sum) {
          indice_a++;
      }
      if (soma > desired_sum){
          indice_c--;
      }
    }

    *solucao_mascara = solucao;
    free(a);
    free(c);
    return ret;
}

```

```c
// ------------------------------------------- //

// ------------schroeppel_shamir------------------- //

int schroeppel_shamir (int n, integer_t p[], integer_t desired_soma, integer_t *solucao_mascara) {

  int n1 = n/4 + (n%4)/2;
  int n2 = n/4;
  int n3 = n/4 + (n%4)/2;
  int n4 = n-n1-n2-n3;

  integer_t p1[n];
  integer_t p2[n2];
  integer_t p3[n3];
  integer_t p4[n4];

  for (int i=0; i<n; i++){
    if (i < n1){
      p1[i] = p[i];
    }
    else if (i < n1+n2){
      p2[i - n1] = p[i];
    }
    else if (i < n1+n2+n3){
      p3[i - n1 - n2] = p[i];
    }
    else {
      p4[i - n1 - n2 - n3] = p[i];
    }
  }

  int length_a1 = 1 << n1;
  int length_a2 = 1 << n2;
  int length_b1 = 1 << n3;
  int length_b2 = 1 << n4;

  mascara_struct_t * a1 = malloc((length_a1) * sizeof(mascara_struct_t));
  mascara_struct_t * a2 = malloc((length_a2) * sizeof(mascara_struct_t));
  mascara_struct_t * b1 = malloc((length_b1) * sizeof(mascara_struct_t));
  mascara_struct_t * b2 = malloc((length_b2) * sizeof(mascara_struct_t));

  soma_all_subsets(n1, p1, a1, 0, 0, 0, 0);
  soma_all_subsets(n2, p2, a2, 0, 0, 0, 0);
  soma_all_subsets(n3, p3, b1, 0, 0, 0, 0);
  soma_all_subsets(n4, p4, b2, 0, 0, 0, 0);


  #ifdef MergeSort
  merge_sort(a1, 0, length_a1);
```

```c
    merge_sort(a2, 0, length_a2);
    merge_sort(b1, 0, length_b1);
    merge_sort(b2, 0, length_b2);
    #endif

    #ifdef QuickSort
    quicksort(a1, 0, length_a1 - 1);
    quicksort(a2, 0, length_a2 - 1);
    quicksort(b1, 0, length_b1 - 1);
    quicksort(b2, 0, length_b2 - 1);
    #endif


    heapStruct_t min_heap[length_a1];
    int n_min_heap = 0;

    heapStruct_t max_heap[length_b1];
    int n_max_heap = 0;

    for (int i = 0; i < length_a1; i++) {
      heapStruct_t soma = {
          .mascara = a1[i].mascara | (a2[0].mascara << n1),
          .soma = a1[i].soma + a2[0].soma,
          .indice1 = i,
          .indice2 = 0
      };
      push(min_heap, soma, &n_min_heap, 1);
     }

    for (int i = 0; i < length_b1; i++) {
      heapStruct_t soma = {
          .mascara = b1[i].mascara | (b2[length_b2 - 1].mascara << n3),
          .soma = b1[i].soma + b2[length_b2- 1].soma,
          .indice1 = i,
          .indice2 = length_b2 - 1
      };
      push(max_heap,soma,&n_max_heap, 2);
     }

    int ret = 0;
    integer_t solucao = 0;

    while (n_max_heap > 0 && n_min_heap > 0){
      integer_t partial_soma = min_heap[0].soma;
      partial_soma += max_heap[0].soma;

      if (partial_soma ==  desired_soma) {
          solucao = min_heap[0].mascara | (max_heap[0].mascara << (n1+n2));
          ret = 1;
          break;
```

```c
      }

      if (partial_soma < desired_soma){
          heapStruct_t vmin = pop(min_heap, &n_min_heap, 1);
          vmin.indice2++;
          if (vmin.indice2 < length_a2) {
            heapStruct_t w = {
                .mascara = a1[vmin.indice1].mascara | (a2[vmin.indice2].mascara << n1),
                .soma = a1[vmin.indice1].soma + a2[vmin.indice2].soma,
                .indice1 = vmin.indice1,
                .indice2 = vmin.indice2
            };
            push(min_heap, w, &n_min_heap, 1);
           }
        }

      if (partial_soma > desired_soma){
          heapStruct_t vmax = pop(max_heap,&n_max_heap, 2);
          vmax.indice2--;

          if (vmax.indice2 >= 0) {
            heapStruct_t w = {
                .mascara = b1[vmax.indice1].mascara | (b1[vmax.indice2].mascara << n3),
                .soma = b1[vmax.indice1].soma + b2[vmax.indice2].soma,
                .indice1 = vmax.indice1,
                .indice2 = vmax.indice2
            };
            push(max_heap,w,&n_max_heap, 2);
          }
      }
    }

  *solucao_mascara = solucao;
  free(a1);
  free(a2);
  free(b1);
  free(b2);
  return ret;
}
// ---------------------------------------------- //


// ---------------main program------------------- //
int main(void)
{
  fprintf(stderr,"Program configuration:\n");
  fprintf(stderr,"  min_n ....... %d\n",min_n);
  fprintf(stderr,"  max_n ....... %d\n",max_n);
  fprintf(stderr,"  n_somas ...... %d\n",n_sums);
  fprintf(stderr,"  n_problems .. %d\n",n_problems);
```

```c
  fprintf(stderr,"  integer_t ... %d bits\n",8 * (int)sizeof(integer_t));

  #ifdef WriteFile
  FILE *fpb = NULL;
  fpb = fopen("test.txt", "w");
  #endif
  for(int i = 0;i < n_problems;i++)
  {
    int n = all_subset_sum_problems[i].n;
    if(n > max_n)
      continue;
    #if defined(OutputSolutions) || defined(PrintOnlySolution)
    printf("n=%d\n", n);
    #endif
    integer_t *p = all_subset_sum_problems[i].p;
    for(int j = 0;j < n_sums;j++)
    {
      integer_t desired_soma = all_subset_sum_problems[i].sums[j];
      #if defined(HorowitzSahni) || defined(SchroeppelShamir)
      integer_t b = 0;
      #endif

      #if defined(RecursiveBruteForce) || defined(CleverRecursiveBruteForce)
      int b[n];
      for (int l=0; l < n; l++){
        b[l]=0;
      }
      #endif
      int success;
      if (n>=min_n){
        #ifdef WriteFile
        char stringb[20];
        sprintf(stringb, "%d", n);
        fprintf(fpb, "%s ", stringb);
        #endif

        double time = cpu_time();
        #ifdef IterativeBruteForce
          success=iterative_brute_force(n, p, desired_soma);
        #endif

        #ifdef RecursiveBruteForce
          success=recursive_brute_force(n, p, b, desired_soma, 0, 0);
        #endif

        #ifdef CleverRecursiveBruteForce
          success=clever_recursive_brute_force(n, p, b, desired_soma, 0, 0, n-1, 0);
        #endif

```

```c
        #ifdef HorowitzSahni
          success=horowitz_sahni(n, p, &b, desired_soma, 0, 0);
        #endif

        #ifdef SchroeppelShamir
          success=schroeppel_shamir(n, p, desired_soma, &b);
        #endif

        if (success==1){
          ;
          #if !defined(PrintOnlySolution)
          printf("Solução encontrada, n=%d.\n", n);
          #endif

          #ifdef OutputSolutionsHS
          for (int i = 0; i < n; i++) {
            printf("%s", b & 1 ? "1" : "0");
            b = b >> 1;
          }
          printf("\n");
          #endif

          #ifdef OutputSolutionsRC
          for (int i=0; i<n; i++){
            printf("%d", b[i]);
          }
          printf("\n");
          #endif
        }
        else {
          printf("Solution not found.\n");
          return 0;

        }
        time = cpu_time()-time;
        #if defined(PrintTime) && !defined(PrintOnlySolution)
        printf("time needed= %f\n", time);
        #endif
        #ifdef WriteFile
        char tempob[50];
        sprintf(tempob, "%f\n", time);
        fprintf(fpb, "%s", tempob);
        #endif
      }
    }
    printf("\n");

  }
```

```c
  }
  #ifdef WriteFile
  fclose(fpb);
  #endif
  return 1;
}

// ------------------------------------------ //
```

Código Desenvolvido em MatLab

```
%% iterative_brute_force
fileID=fopen('iterative_brute_force_103668.txt','r');

tline=fgetl(fileID);
n_array_ib=zeros(1,21);
time_array=zeros(1,20*21);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    disp(time);
    if ~ismember(n,n_array_ib)
        n_array_ib(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end

time_array_media_ib=zeros(1,21);
time_array_best_ib=zeros(1,21);
time_array_worst_ib=zeros(1,21);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_ib(count_tm)=media;
        time_array_best_ib(count_tm)=best;
        time_array_worst_ib(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% recursive brute_force
fileID=fopen('recursive_brute_force_103530.txt','r');
tline=fgetl(fileID);
n_array_rb=zeros(1,26);
time_array=zeros(1,20*26);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    disp(time);
    if ~ismember(n,n_array_rb)
        n_array_rb(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end

time_array_media_rb=zeros(1,26);
time_array_best_rb=zeros(1,26);
time_array_worst_rb=zeros(1,26);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_rb(count_tm)=media;
        time_array_best_rb(count_tm)=best;
        time_array_worst_rb(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% clever_recursive_brute_force
fileID=fopen('clever_recursive_brute_force_103530.txt','r');
tline=fgetl(fileID);
n_array_cb=zeros(1,26);
time_array=zeros(1,20*26);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    disp(time);
    if ~ismember(n,n_array_cb)
        n_array_cb(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;
    tline=fgetl(fileID);
end
time_array_media_cb=zeros(1,25);
time_array_best_cb=zeros(1,25);
time_array_worst_cb=zeros(1,25);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_cb(count_tm)=media;
        time_array_best_cb(count_tm)=best;
        time_array_worst_cb(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% quicksort horowitz
fileID=fopen('quicksort_horowitz_103668.txt','r');
tline=fgetl(fileID);
n_array_qh=zeros(1,48);
time_array=zeros(1,20*48);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    if ~ismember(n,n_array_qh)
        n_array_qh(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end
time_array_media_qh=zeros(1,48);
time_array_best_qh=zeros(1,48);
time_array_worst_qh=zeros(1,48);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_qh(count_tm)=media;
        time_array_best_qh(count_tm)=best;
        time_array_worst_qh(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% merge sort horowitz
fileID=fopen('mergesort_horowitz_103668.txt','r');
tline=fgetl(fileID);
n_array_mh=zeros(1,48);
time_array=zeros(1,20*48);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    if ~ismember(n,n_array_mh)
        n_array_mh(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end
time_array_media_mh=zeros(1,48);
time_array_best_mh=zeros(1,48);
time_array_worst_mh=zeros(1,48);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_mh(count_tm)=media;
        time_array_best_mh(count_tm)=best;
        time_array_worst_mh(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% quicksort Schroeppel
fileID=fopen('quicksort_schroeppel_103668.txt','r');
tline=fgetl(fileID);
n_array_qs=zeros(1,48);
time_array=zeros(1,20*48);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    if ~ismember(n,n_array_qs)
        n_array_qs(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end
time_array_media_qs=zeros(1,48);
time_array_best_qs=zeros(1,48);
time_array_worst_qs=zeros(1,48);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_qs(count_tm)=media;
        time_array_best_qs(count_tm)=best;
        time_array_worst_qs(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% merge sort Schroeppel
fileID=fopen('mergesort_schroeppel_103668.txt','r');
tline=fgetl(fileID);
n_array_ms=zeros(1,48);
time_array=zeros(1,20*48);
count=1;
count_t=1;
while ischar(tline)
    n=str2double(tline(1:2));
    time=str2double(tline(4:end));
    if ~ismember(n,n_array_ms)
        n_array_ms(count)=n;
        count=count+1;
    end
    time_array(count_t)=time;
    count_t=count_t+1;

    tline=fgetl(fileID);
end
time_array_media_ms=zeros(1,48);
time_array_best_ms=zeros(1,48);
time_array_worst_ms=zeros(1,48);
count_20=1;
count_tm=1;
soma=0;
media=0;
worst=time_array(1);
best=time_array(1);
for i=1:length(time_array)
    if time_array(i)>worst
```

```
for i=1:length(time_array)
    if time_array(i)>worst
        worst=time_array(i);
    end
    if time_array(i)<best
        best=time_array(i);
    end
    if count_20==20
        count_20=1;
        soma=soma+time_array(i);
        media=soma/20;
        time_array_media_ms(count_tm)=media;
        time_array_best_ms(count_tm)=best;
        time_array_worst_ms(count_tm)=worst;
        worst=time_array(i);
        best=time_array(i);
        soma=0;
        media=0;
        count_tm=count_tm+1;
        continue
    end
    soma=soma+time_array(i);
    count_20=count_20+1;
end
```

```
%% all algorithms graph
figure(1);
ib(1) = semilogy(n_array_ib,time_array_media_ib,".-","DisplayName","iterative brute force average","Color",[0.4 0.5 0.8]);
hold on;
ib(2) = semilogy(n_array_ib,time_array_best_ib,".-","DisplayName","iterative brute force best","Color",[0.4 0.5 0.3]);
ib(3) = semilogy(n_array_ib,time_array_worst_ib,".-","DisplayName","iterative brute force worst","Color",[0.5 0.1 0.7]);

rb(1) = semilogy(n_array_rb,time_array_media_rb,".-","DisplayName","brute force average","Color",[0 1 0]);
rb(2) = semilogy(n_array_rb,time_array_best_rb,".-","DisplayName","brute force best","Color",[0 1 1]);
rb(3) = semilogy(n_array_rb,time_array_worst_rb,".-","DisplayName","brute force worst","Color",[0.9 0 0.6]);

cb(1) = semilogy(n_array_cb,time_array_media_cb,".-","DisplayName","clever recursive brute force average","Color",[0 0.5 1]);
cb(2) = semilogy(n_array_cb,time_array_best_cb,".-","DisplayName","clever recursive brute force best","Color",[1 0.5 0.5]);
cb(3) = semilogy(n_array_cb,time_array_worst_cb,".-","DisplayName","clever recursive brute force worst","Color",[0 0 0]);

qh(1) = semilogy(n_array_qh,time_array_media_qh,".-","DisplayName","quick horowitz average","Color",[1 0 0]);
qh(2) = semilogy(n_array_qh,time_array_best_qh,".-","DisplayName","quick horowitz best","Color",[1 1 0]);
qh(3) = semilogy(n_array_qh,time_array_worst_qh,".-","DisplayName","quick horowitz worst","Color",[0.1 0 0.9]);

mh(1) = semilogy(n_array_mh,time_array_media_mh,".-","DisplayName","merge horowitz average","Color",[0.1 0.5 0.1]);
mh(2) = semilogy(n_array_mh,time_array_best_mh,".-","DisplayName","merge horowitz best","Color","black");
mh(3) = semilogy(n_array_mh,time_array_worst_mh,".-","DisplayName","merge horowitz worst","Color",[1 1 0.2 ]);

qs(1) = semilogy(n_array_qs,time_array_media_qs,".-","DisplayName","quick Schroeppel average","Color",[0.1 0.5 0.1]);
qs(2) = semilogy(n_array_qs,time_array_best_qs,".-","DisplayName","quick Schroeppel best","Color",[0.3 0.6 1]);
qs(3) = semilogy(n_array_qs,time_array_worst_qs,".-","DisplayName","quick Schroeppel worst","Color",[0.3 1 1]);

ms(1) = semilogy(n_array_ms,time_array_media_ms,".-","DisplayName","merge Schroeppel average","Color",[0.8 0.4 0]);
ms(2) = semilogy(n_array_ms,time_array_best_ms,".-","DisplayName","merge Schroeppel best","Color",[0.8 0 0]);
ms(3) = semilogy(n_array_ms,time_array_worst_ms,".-","DisplayName","merge Schroeppel worst","Color",[0.4 0 0.9]);
```

```
grid on;
title('Titulo para o gráfico');
hold off;
lgd=legend;
```