**SIN110 Algoritmos e Grafos - Exercício E10**

**Aluna:** Caroline Lopes Resek

**Matrícula:** 2017010113

1. Considere como entrada um conjunto S com n números reais e, um número real x:
   1. Projete um algoritmo para determinar se há dois elementos de S cuja soma seja exatamente x. O algoritmo deverá rodar em tempo proporcional a O(nlgn).

int somaX(int \*S, int x, int n){

int k = 0;

for (int i = 0; i < n; i++){

for (int j = i + 1; j < n; j++){

if(S[i] + S[j] == x){

k++;

}

}

}

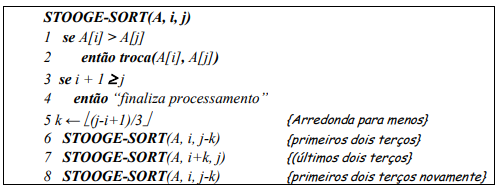
if (k>0) return 0;

else return 1;

}

* 1. Suponha agora que o conjunto S está ordenado. Projete um novo algoritmo que resolva o problema anterior em tempo proporcional a O(n).

1. Três “pesquisadores” propuseram o seguinte algoritmo de ordenação “elegante”:



* 1. Mostre que para n elementos, STOOGE-SORT(A,1, n) ordena corretamente o arranjo de entrada A[1..n].

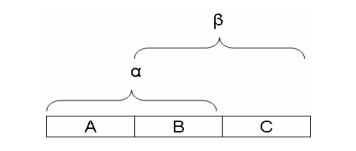
• Para arranjos de 1 ou 2 elementos fornecidos como entrada, retorna o arranjo ordenado. Propriedade observada nas linhas de 1 a 4.

• Na primeira chamada recursiva (linha 6), ele coloca os menores elementos de α em A e os maiores elementos em B.

• Na segunda chamada recursiva (linha 7), ele coloca os menores elementos de β em B e os maiores em C, alterando os valores contidos em B

• Após a terceira chamada recursiva (linha 8) o STOOGE-SORT terá reordenado os elementos de α, com isso todo o arranjo de entrada estará ordenado

Ou seja, o algoritmo ordena corretamente o arranjo de entrada.



* 1. Forneça uma recorrência para o tempo de execução no pior caso de STOOGE-SORT e um limite assintótico restrito (notação ) sobre o tempo de execução no pior caso.

A recorrência para o STOOGE-SORT é T(n) = 3 T() + O(1).

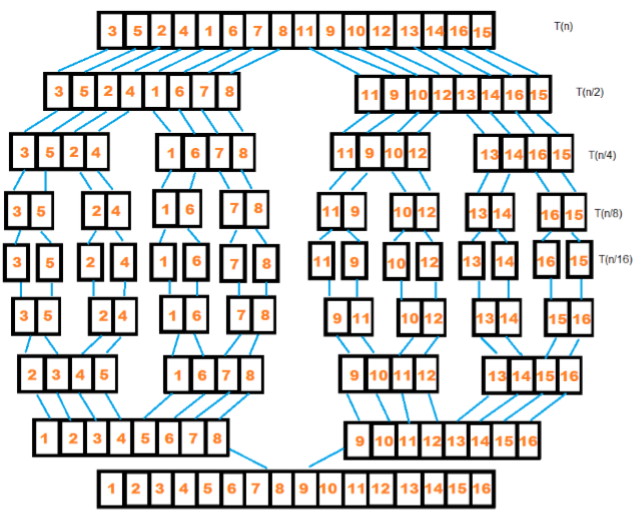
Aplicando o método mestre para: a = 3, b = e f(n) = 1, constata-se que f(n) = , portanto se encaixa no primeiro caso do método.

Para que seja igual a f(n) então , já que .

Obs. 

Então o STOOGE-SORT tem limite assintótico restrito para o pior caso.

1. Desenhe a árvore de recursão para o algoritmo Mergesort aplicado a um vetor de 16 elementos. Por que a técnica de programação dinâmica não é capaz de acelerar o algoritmo?

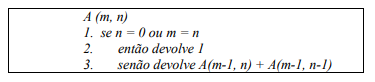


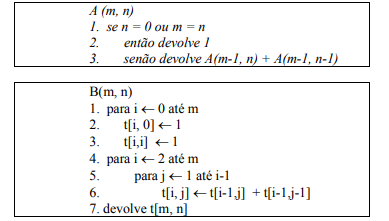
A técnica de programação dinâmica não é capaz de acelerar o algoritmo porque, apesar de ser um algoritmo recursivo e que utiliza o método dividir para conquistar, ele utiliza a função de intercalação que possui tempo linear a qual é chamada repetidas vezes para ir ordenando e fazendo o ‘merge’ dos vetores. Além do fato de que alguns números que anteriormente já foram comparados entre si podem ser comparados entre si novamente.

1. Sabe-se que, para m > 1 e n > 1:



Considere as duas maneiras de implementar uma função que calcula, dados m e n, o valor de  :





* 1. Qual é a complexidade de cada uma das funções? Justifique sua resposta.

Primeira função: ao executá-la nota-se que quanto maiores forem ‘m’ e ‘n’ a função será executadas muitas e muitas vezes pois a função divide-se em muitos subproblemas os quais gerarão somas sucessivas. Por exemplo, o resultado de (15 9) resulta em 5005, ou seja, serão realizadas 5005 somas sucessivas mais as verificações realizadas pela função na linha 1, logo, sua complexidade é exponencial: O(2n).

Segunda função : possui complexidade igual a O(n²), pois, na linha 5 tem um ciclo ‘para’ que está dentro do ciclo ‘para’ da linha 4.

* 1. Qual é a mais eficiente? Por quê?

A segunda função é a mais eficiente pois, apesar de possuir complexidade quadrática, para números muito grandes ela apresenta melhor desempenho se comparada à segunda função que possui complexidade exponencial.