

1. Processadores: O(n) Tempo: O(1) Custo: O(n)

É ótimo por ser um algoritmo sequencial com O(n) e o custo O(n).

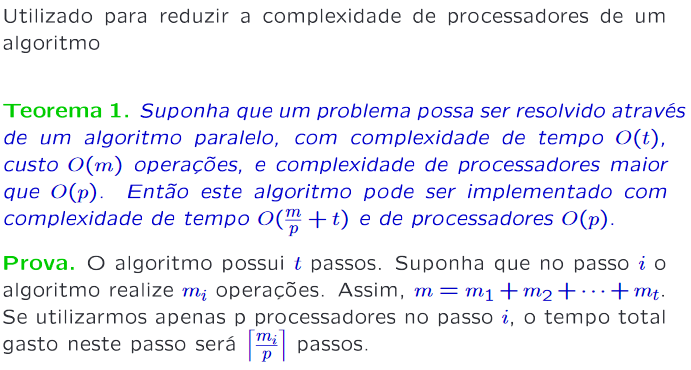
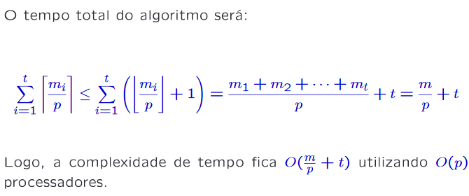
1. para 1 <= i <= n faça em paralelo

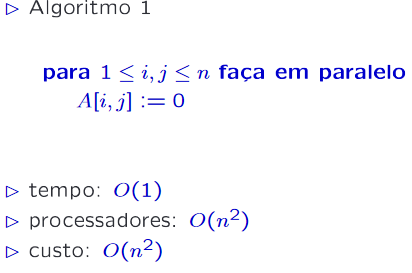
A[i] := i

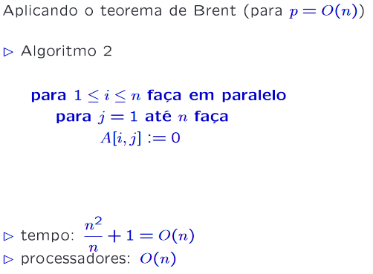
Processadores: O(n) Tempo: O(n) Custo: O(n²)

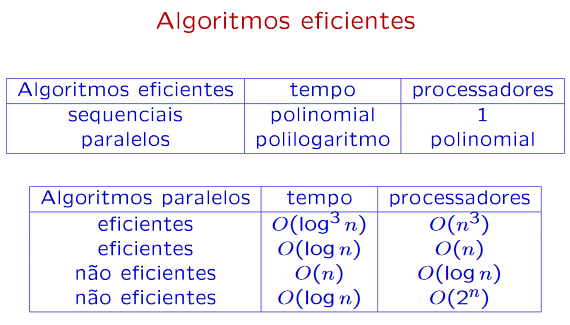
Não é ótimo já que o algoritmo paralelo tem custo maior que o algoritmo sequencial.

1. Aplicar o teorema de Brant:

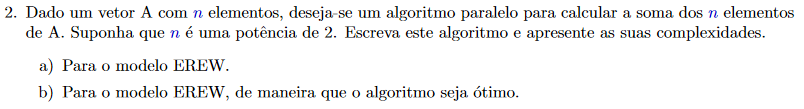








Slides da aula 02 – Página 39 (Brant) e 33 (Algoritmos eficientes)

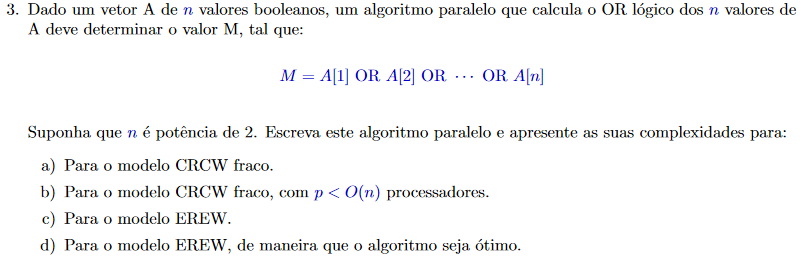


a) para 0 <= i <= n/2 faça em paralelo

para 0 <= j <= (log n) - 1 faça

se (i % 2^j = 0) e (2i + 2^j < n)

A[2i] = A[2i] + A[2i + 2^j]



1. Para o modelo CRCW fraco

B := 0

para 1 ≤ i ≤ n faça em paralelo

se A[i] == 1 então

B := 1

1. Para o modelo CRCW fraco, com p < O(n) processadores.

B := 0

para 1 ≤ i ≤ n/2 faça em paralelo

se A[i] OR A[n/2 + i] então

B := 1



Processadores: O(n log n) Tempo: O(n log n) Custo: O(n log n)

Não é ótimo já que o algoritmo paralelo tem custo maior que o algoritmo sequencial.

#include <omp.h>

#include <stdio.h>

#include <stdlib.h>

int main(int argc, char\* argv[]){

int soma = 0, n = 10;

int a[10] = {0,0,0,0,0,0,0,0,0,10};

#pragma omp parallel for reduction (+:soma)

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

soma = soma + a[i];

fprintf(stdout, "Soma da array: %d\n", soma);

exit(0);

}