**Lista de Exercícios**

1. **O(n log n)**

Fórmula de recorrência:

T(n) = aT(n/b) + f(n);

Analisando o algoritmo temos 3 chamadas recursivas:

a = 3;

O problema se divide em 3 partes:

b = 3;

A combinação é constituída por n conquistas:

f(n) = n;

Solução da equação:

T(n) = 3T(n/3)+n;

Então,

nlogab = nlog33 = n1 = n;

f(n) = n;

f(n) = θ(nlogab) = θ(n);

logo,

T(n) = θ(nlogab lg n) = θ(f(n)lg n);

ou seja,

**T(n) =** **θ(nlg n);**

2.a) i. **Computador A**

1/100 seg < 6,64/5 seg

ii.**Computador B**

2x106 seg < 2x1012 seg

iii.**Computador B**

46,48x109 seg < 1013 seg

b) **n = 58**

algA: 197062.4806// algB: 195112

3. É correto, pois

f(n) = O(g(n)) = O(h(n));

**θ(n2) = O(n2) = O(n2)**

e se um função f(n) é θ(g(n)) ela também é O(g(n)), já que se ela é limitada inferiormente e superiormente por g(n)e isso inclui o limite superior por g(n).

4.  **É correto**, pois

g(n) = Ω(h(n));

significa que o algoritmo do Bubble Sort é limite inferior do algoritmo de Quick Sort, ou seja Bubble Sort é mais rápido que o Quick Sort, como estamos comparando o melhor caso, em sua maioria

(n log n) > (n)

logo, no melhor caso, o Bubble Sort é mais eficiente que o Quick Sort.

5. **c)**

Pois o algoritmo A tem desempenho θ(n) e o algoritmo B tem θ(n log n) e ao compará-los nota-se que o desempenho de A é melhor que o de B, no qual podemos representar assim:

A é limite inferior de B

g(n) = Ω(f(n));

6. **Computador A,** pois o algoritmo B com desempenho θ(n log n) demora aproximadamente 2 segundos, enquanto o algoritmo A de desempenho θ(n) no computador B dura 100 segundos.

7. n1 = 1, n2 = 800;

for(i=1; i<=10;i++){

palpite = (n1 + n2) // 2;

if(palpite\_Forca(palpite) == “Força Superior”){

n2 = palpite – 1;

}

else if(palpite\_Forca(palpite) == “Força Inferior”){

n1 = palpite + 1;

}else{

printf(“Acertou em %d tentativas, o palpite era %d”,i,palpite);

}

}

Como se utiliza um busca binária a complexidade se torna **O(log n).**

8. O melhor caso seria se o menor número estivesse na primeira posição, com desempenho **θ(1)** e no pior caso seria se estivesse na última posição, com desempenho **θ(n).** O menor elemento, se o vetor estiver ordenado, obrigatoriamente deve estar na primeira posição, o que leva ao desempenho de **θ(1)**.

9. Sim, se **usarmos recursão** e fizermos uma verificação **se o expoente é par ou ímpar** para adaptar o retorno, este algoritmo pode alcançar um desempenho O(log n).

10. **d)**

11. **b)** e **e)**

Pois o desempenho de A é θ(n2) e o de B é θ(n), assim sendo A limite superior de B,

g(n) = O(f(n));

e B sendo limite inferior de A,

f(n) = Ω(g(n));