Projeto e Análise de Algoritmos

Lista de Exercícios I

August 27, 2018

1. Verdadeiro ou Falso?

   (a) $2^{n+1} = O(2^n)$?
   2*2^n = O(2^n)
   C>4
   Veradadeiro

   (b) $2^{2n} = O(2^n)$?
   2^n * 2^n
   C<2^n
   Falso

2. Para cada um dos pares de funções abaixo, verifique se f(n) = O(g(n)), f(n) = (g(n)) ou f(n) = (g(n)).

   (a) $f(n) = \log(n^2)$; $g(n) = \log(n) + 5$
   Log(n^2) = 2log(n)
   5 para inifinito vai pra 0
   2log(n)==2log(n), (C vale 2)
   Logo é f(n) = Big Theta (g(n))

   (b) $f(n) = n$; $g(n) = \log(n^2)$
   2log(n) cresce menos q n
   Logo g(x) é Big omega
   –

   (c) f(n) = n log(n) + n; g(n) = log(n)
   Nlog(n) cresce muito mais que log(n).
   Logo g(x) é big Omega de f(x)

   (d) f(n) = 10; g(n) = log(10)
   Podemos considerar as duas como constantes, portanto são Bih Theta

   (e) $f(n) = 2^n$; $g(n) = 10n^2$
   Exponecial crece mais rapidamente do que algo polinomial. Logo f(n)=O(g(n))

   (f) $f(n) = 2^n$; $g(n) = 3^n$
   As duas crescem numa mesma proporção pois podem ser multiplicadas por uma constante

3. Prove que n^3 $3n^2$ $n + 1 =$ (n^3)

   Lim n->infinito ((n^3+3n^2+n+1)/n^3), todos vão para zero menos o n^3.
   Logo lim n->infinito (n^3/n^3)= 1

4. Prove que $n^2 = O(2^n)$
   Lim n->infinito(2^n/n^2) = inifinito

5. Dadas as funções abaixo, ordene as funções de acordo com a ordem de crescimento (da menor para a maior).

   (a) n

   (b) n − n^3 + 7n^2

   (c) n^2 + lg(n)

(d) n^3

(e) 2^n

(f) lg(n)

(g) n^2

(h) (lg(n))^2

(i) nlg(n)

(j) sqrt(n)

(k) 2^(n−1)

(l) n!

(Lg(n))<((lg(n))^2)<(sqrt(n))<(n)<(n*lg(n)<(n^2,n^2+lg(n))<(n^3,n-n^3+7n^2) < (2^(n-1),2^n) <(n!)

6. Utilize a notação apropriada (O, ,) para inidicar a eficiência temporal de uma busca sequencial e de uma busca binária.

   (a) No pior caso
   Sequencial-> O(n)
   Binária-> log(n)

   (b) No melhor caso
   Sequencial-> O(1)
   Binária-> O(1)

   (c) No caso médio
   Sequencial-> O(n)
   Binária-> log(n)

7. Escreva um algoritmo para verificar se todos os elementos de um vetor de inteiros são dis-tintos. Mostre a complexidade temporal.

```
bool verifica_repetidos(vector<int> v){
        auto max=max_element(v.begin(),v.end()); (N)
        vector<bool> bin(*max,false); (N)
        for (int i =0;i<v.size();i++){ (N)
                if (bin[v[i]])              (N)
                        return false;  (1)
                bin[v[i]]=true;           (N)
        }
        return true;                     (1)

}
```

Complexidade = Theta(N)

8. Mostre a complexidade temporal mais estrita dos algoritmos abaixo:

(a) for(int i = n; i > 0; i /= 2) { log2(n)
for(int j = 1; j < n; j *= 2)
{   log2(n)^2
for(int k = 0; k < n; k += 2) { (n log(n)^2)
... // constant number of operations
}
}
}

N+ log(n)^2
Theta(nlog(n)^2)

(b) for (i=1; i < n; i *= 2) {
for (j = n; j > 0; j /= 2) {
for (k = j; k < n; k += 2) {
sum += (i + j * k);
}
}
}
nlog(n)^2+log(n)^2+log(n)
Theta(nlog(n)^2)

(c) for(int i = n; i > 0; i--) { n
for(int j = 1; j < n; j *= 2) {n log(n)
for(int k = 0; k < j; k++) {nlogn* E(nj)= nlogn*n=n^2*log(n)
... // constant number C of operations
}
}
}

Theta(n^2*log(n))

(d) for(int bound = 1; bound <= n; bound *= 2) { for(int i = 0; i < bound; i++) { (logn)
for(int j = 0; j < n; j += 2) { nlogn
... // constant number of operations
}
for(int j = 1; j < n; j *= 2) { (logn)^2
... // constant number of operations
}
}
}

Theta(nlogn)

9. Verdadeiro ou Falso?

(a) 3^n = Omega (2^n)?
3^n>K^2
Verdadeiro
(b) log(3^n) = Omega(log(2^n)?
Nlog(3) =Omega(NLog(2))
N=Omega(N)
Verdadeiro

10. Algoritmo. Você tem n > 2 moedas “idênticas” e uma balança com duas bases de medição. Uma das moedas é falsas. Você não sabe se a moeda falsa é mais leve ou mais pesada que as originais (que possuem o mesmo peso). Projete um algoritmo com complexidade (1) para determinar se a moeda falsa é mais leve ou mais pesada que as demais.

Def v_moedas(Moeda_falsa,Vetor_moedas_verdadeiras[0]):
If(Vetor_moedas_verdadeiras[0]>Moeda_falsa):

```
        Return “moedas verdadeiras são maiores”
 elif(Vetor_moedas_verdadeiras[0]<Moeda_falsa):
        Return “moedas verdadeiras são menores”
Return “as duas tem o mesmo tamanho”
```

11. Algoritmo. Você está olhando para um muro que tem dimensões infinitas em ambas as direções (direta / esquerda). Existe uma única porta neste muro, mas você não sabe o quão distante você está e nem a direção. Você somente pode perceber que existe uma porta quando está próximo a ela. Projete um algoritmo que permite você encontrar a porta caminhando no máximo O(n) passos, onde n é o número de passos entre você e a porta.

```
Auto it = pos, it2=pos;
For (;it->.isPorta || it2->isPorta;it++,it2--);
```