

PUC-Rio
Departamento de Informática
Prof. Marcus Vinicius S. Poggi de Aragão
Horário: 3as-feiras de 13 às 16 horas - Sala 154L
23 de março de 2009
Período: 2009.1

PROJETO E ANÁLISE DE ALGORITMOS (INF 2926)

1ª Lista de Exercícios

1. Considere as seguintes funções:

- (a) $10.n^\pi$
- (b) $\log n$
- (c) $\log(n^2)$
- (d) $0.005.n^{0.00001}$
- (e) $1000.(\log n)^2$
- (f) $30.n^3.\log n$
- (g) $50.n.\log^2 n$
- (h) $(\log n)^{\log n}$
- (i) $\frac{n}{\log n}$
- (j) $70.n$
- (k) $\log \log n$
- (l) $(1.5)^{(\log n)^2}$
- (m) $90.n^2.\log n + n^3.\log n$

- Coloque as funções acima em ordem de crescimento assintótico, i.e. valor quando $n \rightarrow \infty$.
- Utilize pelo menos três vezes cada um dos símbolos O , Ω , Θ , o , e ω para indicar a relação existente entre pares das funções acima (não vale recíprocos).

2. Seja $f_i(n)$ o número de operações realizadas pelo programa i abaixo (\log logaritmo na base 2).

- Obtenha $f_i(n)$ em função de n e de grupos de operações que demandam tempo constante (para os quais você pode utilizar símbolos c_1, c_2, \dots para indicar esse número constante de operações). Caso existam valores indeterminados, considere-os como variáveis e dê o intervalo de valores que esta pode assumir.
- Encontre funções $l_i(n)$ e $u_i(n)$ tais que $f_i(n) = \Omega(l_i(n))$ (a maior possível) e $f_i(n) = O(u_i(n))$ (a menor possível).
- Indique quando existe uma função $h_i(n)$ tal que $f_i(n) = \Theta(h_i(n))$.

- Caso o algoritmo seja recursivo, escreva sua relação de recorrência. Caso seja iterativo e apareçam somatórios, escreva-os.

***** A1 *****

Chamada: A1 (n,k)

```
void A1 (int n, int k)
{
    int i,f,m;
    i = 0;
    f = (int)(log(n)/2);
    while ( 1 )
        {
            m = (int)((i+f)/2);
            p = 2**m;
            if (k < p){
                if (k >= (p/2) )
                    {
                        return(p/2);
                    }
                f = m;
            }
            else {
                if (k < 2*p)
                    {
                        return(p);
                    }
                i = m;
            }
        }
}
```

***** A2 *****

Chamada: A2 (1,n,k)

```
int A2 (int i, int f, int k)
{
    int a,b,p;
    if (i < f)
        {
            a = (int)log(i);
            b = (int)log(f);
            h = (int)((a+b)/2);
            p = 2**h;
            if (k < p)
                {
                    if (k >= (p/2) )
                        {
                            return(p/2);
                        }
                    else
                        {
                            return(A2(i, p/2, k));
                        }
                }
        }
}
```

```

        }
    }
else
{
    if (k < 2*p)
    {
        return(p);
    }
    else
    {
        return(A2(2*p, f, k));
    }
}
}
}

```

***** A3 *****

Chamada: A3 (1, n)

void A3 (int i, int f)

```

{
    int j,continue;
    if (i <= f)
    {
        continue = 1;
        j = i;
        while ( (j <= f) && continue)
        {
            if (V[i] > V[j])
            {
                V[i] = V[j];
                continue = 0;
            }
            j = j + 1;
        }
        A3(i,(int)(f/2));
    }
}

```

***** A4 *****

Chamada: A4 (1, n)

void A4 (int i, int f)

```

{
    int j,continue;
    if (i <= f)
    {
        continue = 1;
        j = i;
        while ( (j <= f) && continue)
        {
            if (V[i] > V[j])
            {

```

```

        V[i] = V[j];
        continue = 0;
    }
    j = j + 1;
}
m = (int)((f-i)/5);
A4(i,i+m);
A4(i+m+1,i+2*m);
A4(i+2*m+1,i+3*m);
}
}

```

3. Considere uma matriz quadrada M contendo n números inteiros (você pode assumir que $n = (2^k)^2$ para algum k inteiro). Esta matriz é dada ordenada tanto nas linhas como nas colunas (veja o exemplo abaixo, para $n = 64$, k seria 3). Considere agora o seguinte problema: Dado

$$\begin{bmatrix} 4 & 10 & 21 & 34 & 39 & 40 & 57 & 62 \\ 9 & 17 & 28 & 35 & 41 & 42 & 58 & 72 \\ 19 & 20 & 30 & 47 & 49 & 49 & 65 & 73 \\ 27 & 39 & 40 & 48 & 52 & 60 & 66 & 79 \\ 33 & 43 & 44 & 58 & 59 & 61 & 77 & 81 \\ 46 & 46 & 60 & 63 & 69 & 71 & 79 & 88 \\ 56 & 61 & 62 & 68 & 72 & 76 & 87 & 88 \\ 64 & 67 & 69 & 73 & 83 & 89 & 90 & 91 \end{bmatrix}$$

um número inteiro i , deseja-se saber se este se encontra na matriz M . Analise a complexidade (de pior caso) dos algoritmos abaixo para resolver este problema. A complexidade deve ser em função de n .

(a) **Algoritmo I**

Passo 0: Seja l uma linha de M . Faça $l = 0$.

Passo 1: Execute uma Busca Binária para encontrar i na linha l de M . Se i for encontrado, responda SIM e PARE. Senão, incremente l .

Passo 2 Se l é uma linha de M vá para o passo 1. Senão, responda NÃO e PARE.

(b) **Algoritmo II**

Passo 0: Seja (p, q) a posição central de M , que divide M em M_1, M_2, M_3 e M_4 , onde M_1 é a submatriz superior esquerda, M_2 a superior direita, M_3 a inferior esquerda e M_4 a inferior direita, de tamanhos iguais. Se M_1 for vazia, Responda NÃO e PARE. Senão vá para o passo 1.

Passo 1 Se i for igual a $M(p, q)$, responda SIM e PARE. Senão vá para o passo 2.

Passo 2: Se i for menor que $M(p, q)$ execute Alg. II para as matrizes M_1, M_2 , e M_3 . Senão execute Alg. II para M_2, M_3 e M_4 .

(c) **Algoritmo III**

Passo 0: Seja (p, p) uma posição da diagonal de M .

Passo 1: Busque sequencialmente na diagonal a posição p tal que $M(p, p) < i < M(p+1, p+1)$ utilize essa posição para obter quatro matrizes a partir de M : M_1, M_2, M_3 e M_4 (observe que o pior caso é o que tem estas 4 matrizes de tamanhos iguais). Se M_1, M_2, M_3 e M_4 forem vazias, Responda NÃO e PARE. Senão vá para o passo 2.

Passo 2 Se i for igual a $M(p, p)$ ou $M(p+1, p+1)$, responda SIM e PARE. Senão vá para o passo 3.

Passo 3: Execute Alg. III para as matrizes M_2 e M_3 .

(d) **Algoritmo IV**

Passo 0: Seja (p, p) uma posição da diagonal de M .

Passo 1: Utilize *Busca Binária* na diagonal para encontrar a posição p tal que $M(p, p) < i < M(p+1, p+1)$ utilize essa posição para obter quatro matrizes a partir de M : M_1, M_2, M_3 e M_4 (observe que o pior caso é o que tem estas 4 matrizes de tamanhos iguais). Se M_1, M_2, M_3 e M_4 forem vazias, Responda NÃO e PARE. Senão vá para o passo 2.

Passo 2 Se i for igual a $M(p, p)$ ou $M(p+1, p+1)$, responda SIM e PARE. Senão vá para o passo 3.

Passo 3: Execute Alg. IV para as matrizes M_2 e M_3 .

4. Analise a eficiência do algoritmo abaixo, calculando o número de passos executados pelo mesmo em função de um número n fornecido como entrada. Considere que todas as operações básicas envolvidas (atribuição, operações lógicas, operações aritméticas, entrada/saída) possuem custo constante ($c = O(1)$).

```
Leia(n);
x <- 0;
Para i <- 1 até n faça
  Para j <- i+1 até n faça
    Para k <- 1 até j-i faça
      x  x + 1;
Imprima(x);
```

5. Perdido em uma terra muito distante, você se encontra em frente a um muro de comprimento infinito para os dois lados (esquerda e direita). Em meio a uma escuridão total, você carrega um lampião que lhe possibilita ver apenas a porção do muro que se encontra exatamente à sua frente (o campo de visão que o lampião lhe proporciona equivale exatamente ao tamanho de um passo seu). Existe uma porta no muro que você deseja atravessar. Supondo que a mesma esteja a n passos de sua posição inicial (não se sabe se à direita ou à esquerda), elabore um

algoritmo para caminhar ao longo do muro que encontre a porta em $O(n)$ passos. Considere que n é um valor desconhecido (informação pertencente à instância). Considere que a ação composta por dar um passo e verificar a posição do muro correspondente custa $O(1)$.

6. “Pseudo ou não-pseudo? Eis a questão.”

- (a) Defina os conceitos de algoritmo polinomial e algoritmo pseudopolinomial.
- (b) Forneça um exemplo de um algoritmo polinomial e outro de um algoritmo pseudopolinomial.
- (c) Prove ou refute: Todo algoritmo polinomial é pseudopolinomial.
- (d) Prove ou refute: Todo algoritmo pseudopolinomial é polinomial.

7. Era uma vez um rei que não conhecia algoritmos. Durante um fatídico verão, seu reino fora invadido por um dragão, obrigando os conselheiros da ordem omega-theta a se reunirem e decidirem por contratar o cavaleiro mais corajoso (e ambicioso) do reino que fez uma proposta de recompensa associada ao peso do animal. Considerando que o peso do dragão seja n quilos, a cobrança será feita em moedas de ouro, seguindo a regra de $k \cdot 2^{(n-k)}$ moedas de ouro adicionais para o k -ésimo quilo do dragão. Para exemplificar, um dragão com 4 quilos custaria ao reino $1 \cdot 2^3 + 2 \cdot 2^2 + 3 \cdot 2^1 + 4 \cdot 2^0 = 26$ moedas de ouro.

Elabore um algoritmo POLINOMIAL (LINEAR!!!) que, dado um inteiro positivo n , correspondente ao peso do dragão, calcule a quantidade de moedas a serem pagas ao cavaleiro. O algoritmo (projetado para ser executado pelos conselheiros da corte) deve conter apenas as operações aritméticas de soma, subtração, multiplicação e divisão de inteiros (considere que qualquer uma dessas operações é realizada em tempo constante).

8. No que se refere ao conceito de esquema de codificação de uma instância:

- (a) Prove por indução que um número inteiro positivo n ao ser codificado na base $b \geq 2$ faz uso de $(\log_b n)$ caracteres.
- (b) Prove por indução que um número inteiro positivo n ao ser codificado na base unária faz uso de (n) caracteres.
- (c) Utilizando o resultados anteriores prove que, um algoritmo que recebe como entrada um número inteiro positivo n codificado em uma base $b \geq 2$ e que executa em n passos é um algoritmo de complexidade exponencial em função do tamanho da instância.

9. Considere o seguinte problema:

[ELE] Dados um conjunto de n inteiros distintos $X = \{x_1, \dots, x_n\}$ e pesos positivos $w(x_j)$, $j = 1, \dots, n$, associados aos elementos de X , e um inteiro positivo V tal que $0 \leq V \leq W = \sum_{j=1}^n w(x_j)$; encontrar o elemento x_k tal que $\sum_{x_j < x_k} w(x_j) < V$ e $w(x_k) + \sum_{x_j < x_k} w(x_j) \geq V$.

- (a) Projete um algoritmo $O(n \log n)$ para resolver esse problema.
- (b) Utilize divisão e conquista para projetar um algoritmo $O(n)$ para resolver esse problema.
- (c) Apresente detalhadamente a análise da complexidade do item anterior.

10. Sejam u e v dois números de n bits (considere que n é potência de 2). A multiplicação tradicional de u por v utiliza $O(n^2)$ operações. Um algoritmo baseado em divisão e conquista divide os números em duas partes iguais, calculando o produto como: $uv = (a2^{n/2} + b)(c2^{n/2} + d) = ac2^n + (ad + bc)2^{n/2} + bd$. As multiplicações ac , ad , bc e bd são feitas usando este mesmo algoritmo recursivamente.
- Escreva este algoritmo
 - Determine a complexidade
 - Qual é a complexidade se $ad + bc$ é calculado como $(a + b)(c + d) - ac - bd$?
11. Verdadeiro ou Falso. Justifique.
- (a) $(\log n)^{100} = O(n^\epsilon)$, $\forall \epsilon > 0$.
- (b) $2^{n+1} = O(2^n)$.
- (c) Se $g(n, m) = m \log_d n$ onde $d = \lceil m/n + 2 \rceil$ ($\lceil x \rceil$ é o menor inteiro maior que x), $m = O(n^2)$, então $g(n, m) = O(m^{1+\epsilon})$ $\forall \epsilon > 0$.
12. Determine uma forma fechada para cada uma das seguintes recorrências:
- (a) $T(1) = 1$;
 $T(2) = 6$;
 $T(n) = T(n - 2) + 3n + 4$, $\forall n \geq 3$.
- (b) $T(1) = 1$;
 $T(2) = 6$;
 $T(n) = 2.T(n - 2) + 3$, $\forall n \geq 3$.
- (c) $T(1) = 1$;
 $T(n) = \sum_{i=1}^{n-1} (T(i) + T(n - i)) + 1$, $\forall n \geq 2$.
13. Seja $S = \{x_1, x_2, \dots, x_n\}$ uma seqüência de números reais (não necessariamente positivos). Projete dois algoritmos de complexidade $O(n)$, um iterativo e um que utilize divisão e conquista (ambos podem ser recursivos!), que determine uma subseqüência $S' = \{x_i, x_{i+1}, \dots, x_j\}$ de elementos consecutivos da seqüência S tal que o produto dos números que compõem S' é máximo dentre todas as subseqüências consecutivas possíveis.
14. Dados um multiconjunto C contendo n números reais (não necessariamente distintos) e um número real x :
- (a) Projete um algoritmo de complexidade $O(n \log n)$ que determine se existem dois elementos em C cuja soma seja igual a x .
- (b) Considere agora que os elementos do conjunto C estão ordenados crescentemente. Projete um algoritmo de complexidade $O(n)$ para resolver o mesmo problema.
15. O departamento de transporte de uma cidade deseja averiguar se seus motoristas respeitam os conceitos pregados pela técnica de direção defensiva. De acordo com o departamento, todos os veículos deveriam manter entre si uma distância maior ou igual d , definida como *distância de segurança*. Com o objetivo de realizar uma pesquisa, várias câmeras foram instaladas pela

cidade, captado as posições de inúmeros veículos. A posição de um veículo é definida por um par ordenado $(x, y) \in R^2$. Seu objetivo é projetar um algoritmo de complexidade $O(n \log n)$ que obtenha uma cena contendo as posições de n veículos captadas por uma câmera e a distância de segurança d atualmente estabelecida pelo departamento, informando ao final da execução se existem veículos que não respeitam a distância de segurança. Seu algoritmo deve responder apenas *sim* ou *não*. A distância entre dois veículos com posições (x_1, y_1) e (x_2, y_2) é dada por $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.

16. O problema da torre de Hanói generalizado consiste em mover n discos de diâmetros distintos, posicionados em um haste denominada *origem*, para uma haste denominada *destino* utilizando $m \geq 1$ hastes denominadas *de trabalho*. Inicialmente, todos os discos encontram-se empilhados na haste de origem em ordem decrescente de tamanho, de baixo para cima. As demais hastes de trabalho e destino encontram-se vazias. Durante o processo de transferência é permitida a movimentação de apenas um disco por vez. Considere ainda que nenhum disco pode ser posicionado acima de um disco com diâmetro menor que o seu. Projete um algoritmo que resolva o problema da torre de Hanói generalizado utilizando o menor número de movimentos possível. Seu algoritmo deve relatar a ordem e a quantidade de movimentos a serem realizados.
17. Considere uma *d-heap* (uma Heap onde cada nó da árvore tem d filhos ou nenhum, exceto eventualmente por um nó) com n inteiros e um valor x . A *d-heap* possui o seu maior elemento no topo e está organizada em um vetor (de n posições). Considere o seguinte um algoritmo para decidir se um valor x é maior ou igual ao k -ésimo maior elemento na *heap* ou não. Assuma que k é maior ou igual à 1.

• **Algoritmo KH**

P0 Faça $P = \emptyset$, onde P é o conjunto dos nós da Heap maiores que x , e $FP = \{Topo(Heap)\}$, onde FP é o conjunto do filhos dos nós em P que ainda não foram comparados com x . Faça também $q = 0$.

P1 Para cada nó v em FP faça: compare o valor de v com x , se for maior, inclua v em P , remova v de FP , inclua os filhos de v em FP e incremente q . Finalmente compare q com k : se for maior, Pare e responda NÃO.

Caso o valor de v não seja maior que x : remova v de FP . Se FP ficar vazio. Pare e responda SIM.

P2 Retorne ao passo P1.

Demonstre que a complexidade do algoritmo acima é $O(k)$.

18. Era uma vez um rei que não conhecia algoritmos. Durante um fatídico verão, seu reino fora invadido por um dragão, obrigando os conselheiros da ordem omega-theta a se reunirem e decidirem por contratar o cavaleiro mais corajoso (e ambicioso) do reino que fez uma proposta de recompensa associada ao peso do animal. Considerando que o peso do dragão seja n quilos, a cobrança será feita em moedas de ouro, seguindo a regra de $k \cdot 2^{(n-k)}$ moedas de ouro adicionais para o k -ésimo quilo do dragão. Para exemplificar, um dragão com 4 quilos custaria ao reino $1 \cdot 2^3 + 2 \cdot 2^2 + 3 \cdot 2^1 + 4 \cdot 2^0 = 26$ moedas de ouro.

Elabore um algoritmo POLINOMIAL (LINEAR!!!) que, dado um inteiro positivo n , correspondente ao peso do dragão, calcule a quantidade de moedas a serem pagas ao cavaleiro. O algoritmo (projetado para ser executado pelos conselheiros da corte) deve conter apenas as

operações aritméticas de soma, subtração, multiplicação e divisão de inteiros (considere que qualquer uma dessas operações é realizada em tempo constante).

Explique porque o algoritmo proposto é realmente polinomial e **NÃO** apenas pseudo-polinomial (i.e. pseudo-linear).

19. Dois conjuntos de n elementos estão disponíveis em dois computadores. Os $2n$ elementos são diferentes. Proponha um algoritmo para encontrar o n -ésimo elemento entre os $2n$. Os computadores se comunicam por mensagens. Cada mensagem contém um elemento OU um inteiro. Analise a complexidade do seu algoritmo EM TERMOS DE MENSAGENS TROCADAS entre os computadores. Projete um algoritmo de complexidade $O(\log n)$ para este problema.

Dica: Observe que, como somente as mensagens trocadas são contadas, os conjuntos de elementos em cada computador podem ser vistos como sequências **ordenadas** de inteiros.

20. Proponha um algoritmo para ordenar n inteiros no intervalo de 1 a n^3 com as seguintes complexidades:

- (a) $O(n \log n)$.
- (b) $\Theta(n^3)$.
- (c) $O(n)$.

Explique porque os algoritmos propostos são realmente polinomiais e **NÃO** apenas pseudo-polinomiais. Em particular porque o algoritmo da letra “c” é linear e não somente pseudo-linear.

21. Dado um conjunto S de n inteiros com muitas repetições. Este conjunto tem $O(\log n)$ inteiros diferentes. Considere o seguinte algoritmo:

P1: Insira os n elementos de S em uma árvore balanceada de busca (ABB) (pode ser uma árvore AVL, Vermelha e Preta, etc. Você não precisa saber como isso é feito, nem o que são essas árvores, só que cada inserção, busca e remoção leva $O(\log k)$ operações, onde k é o número de elementos na ABB). Caso o elemento já exista na ABB, incremente um contador de quantas vezes o elemento aparece no conjunto.

P2: Retire sucessivamente o menor elemento da ABB e imprima o inteiro retirado tantas vezes quanto estiver no contador a ele associado. (Encontrar o menor elemento também pode ser feito em $O(\log k)$)

Este algoritmo ordena o conjunto S . Qual a sua complexidade ? (em função de n).

22. Sejam S_1 e S_2 conjuntos cujos elementos são números inteiros e onde $|S_1| + |S_2| = n$. Dado um inteiro x , proponha um algoritmo para determinar se existem $e_1 \in S_1$ e $e_2 \in S_2$ tais que $e_1 + e_2 = x$.

- (a) Proponha um algoritmo de complexidade $\Theta(n^2)$ e apresente a análise que permite concluir esta complexidade.
- (b) Proponha um algoritmo de complexidade $O(n \log n)$ e apresente a análise que permite concluir esta complexidade.
- (c) Suponha agora que S_1 e S_2 estejam ordenados. Proponha um algoritmo de complexidade $O(n)$ e apresente a análise que permite concluir esta complexidade.

23. Sejam S_1 e S_2 conjuntos cujos elementos são números inteiros e onde $|S_1| = |S_2| = n$.
- Escreva um algoritmo para determinar a união de S_1 e S_2 . Nenhum elemento deve aparecer mais de uma vez. Analise sua complexidade em função de n (melhor caso e pior caso).
 - Este algoritmo deve executar em $O(n \log n)$.
24. Escreva um algoritmo de "busca ternária" que primeiro testa se o elemento da posição $n/3$ é igual ao elemento procurado x e então, se necessário, testa o elemento da posição $2n/3$ para verificar a igualdade com o elemento x e, possivelmente, reduzir o tamanho do conjunto para um terço do original. Escreva a relação de recorrência do algoritmo, analise a sua complexidade e compare o algoritmo de busca binária. Qual faz menos comparações no pior caso ?
25. O controle de qualidade de uma fábrica de frascos de vidro precisa determinar o maior nível de impacto que os seus frascos podem receber sem quebrar. Para este fim a seção de controle de qualidade possui uma escada. O degrau da escada (x) a partir do qual o frasco quebra é o que determina a resistência de cada tipo de produto oferecido pela fábrica. Considere que esta escada possui n degraus. Se para determinar x , onde $1 \leq x \leq n$, somente um frasco está disponível um único algoritmo garante a determinação de x . Este algoritmo consiste em provocar a queda do frasco do degrau 1, depois do 2, e assim por diante até o frasco quebrar com a sua queda. O número de quedas provocadas neste algoritmo é $O(n)$. Responda ao itens abaixo:
- Considere que 2 frascos estão disponíveis. Proponha para este caso um algoritmo onde o número de quedas provocadas dos frascos é $O(\sqrt{n})$ e que a determinação do valor x é garantida.
 - Mostre que para k frascos disponíveis é possível propor um algoritmo onde o número de quedas é, no pior caso, $k \cdot n^{\frac{1}{k}}$. Dica: Apresente os algoritmos para $k = 2$, e $k = 3$. Em seguida generalize.
 - Suponha que você utilize a busca binária para determinar x . Apresente (e demonstre que é) um limite superior para o número de frascos que serão necessários para garantir que x é determinado.
26. Seja $f_i(n)$ o número de operações realizadas pelo programa i abaixo. Seja $f_i(n)$ a função de n que estima o número de operações elementares executadas pelo algoritmo. Encontre $u_i(n)$ tal que $f(n) = O(u_i(n))$. Caso o algoritmo seja recursivo, escreva sua relação de recorrência. Caso seja iterativo e apareçam somatórios, escreva-os.

V[n] - vetor de inteiros (acessível para todos os procedimentos)

```
***** A1 *****
Chamada: A1 (n)

void A1 (int n)
{
  int i,j,soma;
  soma = 0;
  i = 1;
  while (i <= n)
  {
    j = 1;
    while (j <= i)
    {
      soma = soma + V[j];
      j = j + 1;
    }
    i = i + 1;
  }
}
***** A3 *****
Chamada: A3 (1, n)

void A3 (int i, int n)
{
```

```

int j, continue;
if (i <= n)
{
    continue = 1;
    j = 1;
    while ( (j <= i) && continue)
    {
        if (V[i] > V[j])
        {
            V[i] = V[j];
            continue = 0;
        }
        j = j + 1;
    }
    A3 (i+1, n);
}
}
***** A8 *****
Chamada: A8 (1, n)

void A8 (int i, int f)
{
    int j;
    if (i <= f-2 )
    {
        j = i;
        while (j <= f)
        {
            if (V[i] > V[j])
            {
                V[i] = V[j];
            }
            j = j + 1;
        }
        m = (int)( (i+f)/2 );
        if ( V[m] < V[i] )
        {
            A8 (i, m);
        }
        else
        {
            A8 (m+1, f);
        }
    }
}
}

```

27. Seja uma função convexa $f(x)$ definida no intervalo $[0, U]$. Dado um $\epsilon > 0$, deseja-se encontrar x^0 de modo que o intervalo $[x^0 - \epsilon, x^0 + \epsilon]$ contenha x^* onde $f(x^*)$ é o menor valor de f no intervalo dado.
- Suponha que, para um dado x e um dado U , o cálculo de $f(x)$ se faz em tempo constante. Projete um algoritmo de complexidade $O(\log(U/\epsilon))$.
Dica: Lembre que a função é convexa e, para iterativamente se aproximar do mínimo, determine o valor da função em 4 pontos. O que você pode concluir sobre onde está o mínimo de f ?
 - A complexidade deste algoritmo é polinomial ? Justifique.
28. Considere que os n itens de uma busca arqueológica estão organizados nos andares de uma pirâmide de modo que o item de maior valor está no andar mais alto, os 2 itens de maior valor seguintes estão no andar abaixo, os 4 itens de valores menores no andar abaixo do que tem 2 itens, e assim por diante. Isto é, sendo v_1, v_2, \dots, v_n e que $v_1 \geq v_2 \geq v_3 \geq \dots \geq v_n$, o item 1 está no andar mais alto, 2 e 3 no seguinte, os itens 4, 5, 6 e 7 logo abaixo, e assim por diante. Responda:
- Quantos andares tem a pirâmide que guarda os itens ?
 - Utilizando um vetor para armazenar o valor do item de maior valor de cada andar da pirâmide, proponha um algoritmo $O(\log \log n)$ para determinar o andar em que se encontra um item dado.
29. Considere uma d -heap (uma Heap onde cada nó da árvore tem d filhos ou nenhum, exceto eventualmente por um nó) com n inteiros e um valor x . A d -heap possui o seu maior elemento

no topo e está organizada em um vetor (de n posições). Considere o seguinte um algoritmo para decidir se um valor x é maior ou igual ao k -ésimo maior elemento na *heap* ou não. Assuma que k é maior ou igual à 1.

- **Algoritmo KH**

P0 Faça $P = \emptyset$, onde P é o conjunto dos nós da Heap maiores que x , e $FP = \{Topo(Heap)\}$, onde FP é o conjunto do filhos dos nós em P que ainda não foram comparados com x . Faça também $q = 0$.

P1 Para cada nó v em FP faça: compare o valor de v com x , se for maior, inclua v em P , remova v de FP , inclua os filhos de v em FP e incremente q . Finalmente compare q com k : se for maior, Pare e reponda NÃO.

Caso o valor de v não seja maior que x : remova v de FP . Se FP ficar vazio. Pare e responda SIM.

P2 Retorne ao passo P1.

Demonstre que a complexidade do algoritmo acima é $O(k)$.