1. Algoritmo de descida num monte de máximo ("heap"):

```
=======================================
Algoritmo: descer_monte_máx
Entrada: vetor A, natural t, natural i
Saída: nenhuma
---------------------------------------
1.  e := 2*i, d := e + 1
2.  SE e > t
3.  | RETORNE
4.  SE d > t OU A[e] >= A[d]
5.  | m := e
6.  SENÃO
7.  | m := d
8.  SE A[m] > A[i]
9.  | aux := A[m], A[m] := A[i], A[i] := aux
10. | descer_monte_máx(A,t,m)
=======================================
```

2. Algoritmo de extração do máximo:

```
=======================================
Algoritmo: extrair_máx
Entrada: vetor A, natural t
Saída: um elemento de A
---------------------------------------
1. m := A[1]
2. SE t > 1
3. | A[1] := A[t]
4. | --t
5. | descer_monte_máx(A,t,1)
6. RETORNE m
=======================================
```

3. Algoritmo de construção de um monte:

```
=======================================
Algoritmo: construir_monte_máx
Entrada: vetor A, natural t // t >= 1
Saída: nenhuma
---------------------------------------
1. PARA i DE t A 1
2. | descer_monte_máx(A,t,i)
=======================================
```

3.2. EXERCÍCIO: as primeiras chamadas de "descer_monte_máx" no algoritmo acima são desnecessárias, pois A[i] é uma folha e a chamada em questão certamente não alterará o vetor. Altere, então, o valor inicial de "i", de forma que A[i] seja, na primeira iteração do laço, o primeiro nó interno da árvore induzida pelo monte em questão.

4. Algoritmo de ordenação por monte:

```
=======================================
Algoritmo: ord_monte   // "heapsort"
Entrada: vetor A[1..n]
Saída: nenhuma
---------------------------------------
1. construir_monte_máx(A,n)
2. PARA t DE n A 2
```

3. | A[t] := extrair_máx(A,t)
==========

5. Tempo de execução do algoritmo de descida:

T(t,i)  = a,           se 2i  > t, para algum a > 0 (real)
T(t,i) <= T(t,2i) + b, se 2i <= t, para algum b > 0 (real)

6. Observação: a altura de A[i] num monte de t elementos é:

h(t,i) = piso(lg t) - piso(lg i)   (supõe naturais 0 < i <= t).

Uma maneira de explicar a fórmula acima é: a profundidade de A[i] é dada por
piso(lg i), e a altura de A[i] é igual à profundidade de A[t] menos a de A[i].

7. EXERCÍCIO: mostre que T(t,i) <= b*h(t,i) + a,
para todos i e t naturais tais que 0 < i <= t.
(Dica: tente provar por indução em h(t,i), ou t-i+1.)
(Uma solução do exercício segue abaixo, mas você deve tentar fazê-lo
antes de olhá-la.)

8. COMPLEXIDADES DOS TEMPOS DE EXECUÇÃO DOS ALGORITMOS:

a) extrair_máx: como T(t,1) <= b*h(t,1) + a = b*piso(lg t) + a,
então extrair_máx executa em tempo O(lg t).

b) construir_monte_máx: como cada chamada a descer_monte_máx executa em O(lg t),
então é imediato que construir_monte_máx executa
tem tempo O(t * lg t). (Uma análise mais cuidadosa mostra que o algoritmo
na verdade executa em tempo O(t); veja a sequência desta nota de aula.)

c) ord_monte: como construir_monte_máx(A,n) executa em tempo O(n * lg n)
e cada chamada a extrair_máx executa em tempo O(lg n),
então o algoritmo executa em tempo O(n * lg n).

9. EXERCÍCIO: Mostre que, para todo k > 0 natural, se t = 2^k - 1, então

SOMATÓRIO_{i=1}^{t} (1 + h(t,i)) <= 2t.

(Dica: por indução em k.)
(Uma solução segue abaixo, mas tente fazer o exercício antes de olhá-la.)

10. EXERCÍCIO: Utilize o resultado do lema anterior para mostrar que
o algoritmo construir_monte_máx executa em tempo O(t).

11. EXERCÍCIO: em relação ao algoritmo de ordenação por entrelaçamento
("merge sort"), o algoritmo de ordenação por monte ("heapsort")
tem a vantagem de precisar de apenas O(1) de memória auxiliar (em comparação
com a necessidade de θ(n) da versão imediata do primeiro algoritmo).
Por outro lado, o primeiro algoritmo é estável. O algoritmo de ordenação
por monte acima escrito é estável? Se não, você consegue modificá-lo
de forma que ele passe a sê-lo?

12. Seguem abaixo as soluções dos exercícios 7 e 9 acima.

```
================================================================================
----------------------- SOLUÇÃO DO EXERCÍCIO 7 -----------------------
Queremos mostrar que T(t,i) <= b*h(t,i) + a, para todos i e t naturais
  tais que 0 < i <= t.

Por indução em k = t-i+1:

* BASE (k = 1): Como k = 1, então t = i. Temos então:

  T(t,i) = a = b*0 + a = b*h(t,i) + a, CQD.

* HI: T(t',i') <= b*h(t',i') + a, para todo k' < k.

* PASSO (k > 1): Temos:

  T(t,i) <= T(t,2i) + b
         <= ( b*h(t,2i) + a ) + b  --> pela HI, pois t-2i+1 < t-i+1 (já que i >=
1).
          = b*( piso(lg t) - piso(lg 2i)) + a + b
          = b*( piso(lg t) - ( piso(lg i) + 1 ) ) + a + b
          = b*( piso(lg t) - piso(lg i) ) - b + a + b
          = b*( piso(lg t) - piso(lg i) ) + a
          = b*h(t,i) + a, CQD.

----------------------- SOLUÇÃO DO EXERCÍCIO 7 -----------------------
================================================================================


================================================================================
----------------------- SOLUÇÃO DO EXERCÍCIO 9 -----------------------
Queremos mostrar que, para todo k > 0 natural, se t = 2^k - 1, então

   SOMATÓRIO_{i=1}^{t} (1 + h(t,i)) <= 2t.

Por indução em k:

* BASE (k=1): logo, t = 2^1 - 1 = 1. Temos:

   SOMATÓRIO_{i=1}^{t} (1 + h(t,i))
    = 1 + h(t,1)
    = 1
   <= 2
    = 2*t, CQD.

* HI: SOMATÓRIO_{i=1}^{t'} (1 + h(t,i)) <= 2t', para todo t' = 2^k' - 1,
                                                com k' < k.

* PASSO (k > 1):

   SOMATÓRIO_{i=1}^{t} (1 + h(t,i))

    = SOMATÓRIO_{i=1}^{2^k - 1} (1 + piso(lg (2^k - 1)) - piso(lg i))

    = SOMATÓRIO_{i=1}^{2^k - 1} (1 + (k-1) - piso(lg i))   --> piso(lg (2^k -
1)) = k-1

    = SOMATÓRIO_{i=1}^{2^(k-1) - 1} (1 + (k-1) - piso(lg i))

      + SOMATÓRIO_{i=2^(k-1)}^{2^k - 1} (1 + (k-1) - piso(lg i))  --> separando
o intervalo
                                                                        do
somatório em 2

    = SOMATÓRIO_{i=1}^{2^(k-1) - 1} (1 + (k-2) - piso(lg i))      --> k-1 = (k-
```

```
2) + 1

      + SOMATÓRIO_{i=1}^{2^(k-1) - 1} 1

      + SOMATÓRIO_{i=2^(k-1)}^{2^k - 1} (1 + (k-1) - (k-1))       --> piso(lg i)
= k-1

    = SOMATÓRIO_{i=1}^{2^(k-1) - 1} (1 + piso(lg (2^(k-1) - 1)) - piso(lg i))
                                                                  '
                                                                  `--> piso(lg (2^(k-1)
- 1)) = k-2

      + SOMATÓRIO_{i=1}^{2^(k-1) - 1} 1

      + SOMATÓRIO_{i=2^(k-1)}^{2^k - 1} (1 + (k-1) - (k-1))       --> piso(lg i)
= k-1

   <= 2*(2^(k-1) - 1)                                             --> pela HI

      + SOMATÓRIO_{i=1}^{2^(k-1) - 1} 1

      + SOMATÓRIO_{i=2^(k-1)}^{2^k - 1} 1

    = 2^k - 2 + SOMATÓRIO_{i=1}^{2^k - 1} 1

    = 2^k - 2 + (2^k - 1)*(1+1)/2

    = 2^k - 2 + 2^k - 1

    < 2*(2^k - 1), CQD.

------------------------ SOLUÇÃO DO EXERCÍCIO 9 ------------------------
=========================================================================
```