Questão 1

a)

Entrada: array A, de tamanho T

```
build_max_heap(A)

n = T
while (n > 0):
      swap(a[0],a[n-1])
      bubble_down(a[0])
      n = n-1
```

Primeiramente, o build_max_heap transforma o array em um max heap.
Em seguida, põe o elemento máximo na última posição do array.
É garantido que o elemento máximo está na posição 0 pois o array das posições [0:n-1] é um max heap.
No entanto, ao pôr o elemento que antes estava na posição n-1 no topo do heap, a propriedade de heap não é garantida.
Uma chamada ao algoritmo bubble_down é necessária para 'descer' o valor pela árvore, até que a propriedade de max heap seja garantida.
Em seguida, o próximo elemento máximo vai ser retirado e posto na posição correta, e assim por diante até que 'n', o tamanho do heap
que ocupa as primeiras posições do array A, ser reduzido a zero (todos os elementos foram ordenados com sucesso).

b)

Uma observação crucial torna possível a construção de um heap em O(n), quando construído de forma offline: a maior parte dos nós está perto da base, então é interessante deslocar os nós para baixo, buscando manter a propriedade de um max heap, ao invés de deslocar os nós de baixo para cima.

No pior caso, as folhas descerão 0 níveis (são, no pior caso, n/2 folhas). Em seguida, os nós com altura 1 descerão 1 nível cada (são n/4 nós no primeiro nível, no máximo), depois n/8 nós que poderão descer 2 níveis, e assim por diante.

Essa é a intuição. Formalmente, temos que no pior caso:

número de operações = i * n/(2^i) para todo i = 1... H

Isso converge para O(n).

c)

inicialmente, temos:

```
    1
 7      9
3  0   2
```

Após chamar heapify no nó de valor 9 e valor 7, nenhuma troca foi necessária.
Após chamar heapify no nó de valor 1, uma troca foi necessária:

```
    9
 7      1
3  0   2
```

Após isso, o 1 sofre um heapify recursivamente, e mais uma troca é necessária:

```
    9
 7      2
3  0   1
```

OK: a árvore tem propriedade de max heap.

Questão 2

a1) theta(nlogn) ou O(n logn)
a2) theta(nlogn) ou omega(n logn)

obs: com algumas adaptações, você poderia ter comportamento O(n) no melhor caso: fazendo uma varredura linear e encerrando caso o vetor já estivesse ordenado, mas vamos considerar o mergesort padrão, que vai ter o mesmo comportamento assintótico independente da entrada.

b)

Todos esses seriam respostas válidas:

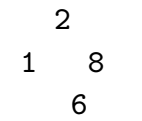
```
1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1
1 1 1 1 1 1 1 1 1
```

Obs: "vetor de entrada" teria sido mais preciso que "vetor desordenado" - é possível que o vetor já esteja ordenado.

Questão 3

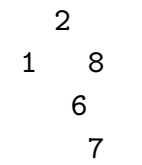
a1) O(n)
a2) O(logn)
a3) O(n)
a4) O(logn)

b)

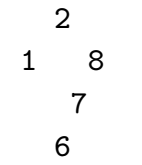


c)

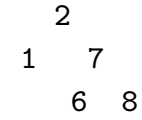
Início:



Rotação esquerda:



Rotação direita:



OK