Complexidade de Algoritmos, 2020/2
Paulo Roberto Albuquerque

1) Como ficará o arranjo `vet` após a execução da chamada de buildHeap (vet, 8), onde:

```
int vet[] = {1, 6, 5, 3, 7, 8, 4, 2};
```

Qual a complexidade de tempo e a complexidade de espaço para o pior caso de execução da função `heapSort`? Qual a complexidade de tempo se todos elementos do arranjo forem iguais? Explique sucintamente cada passo do cálculo dessas complexidades.

A função `heapify` restabelece a propriedade de heap da posição do arranjo passada como parâmetro, a complexidade de tempo no pior caso é O(log n). A função `buildHeap` constrói um heap no arranjo, a complexidade de tempo no pior caso é O(n).

```
int esquerda(int i) { return (2 * i + 1); }

int direita(int i) { return (2 * i + 2); }

void heapify(int *a, int n, int i) {
  int e, d, maior, aux;
  e = esquerda(i);
  d = direita(i);
  if(e < n && a[e] > a[i])
      maior = e;
  else
      maior = i;
  if(d < n && a[d] > a[maior]) maior = d;
  if(maior != i) {
      aux = a[i];
      a[i] = a[maior];
      a[maior] = aux;
      heapify(a, n, maior);
  }
}

void buildHeap(int *a, int n) {
  int i;
  for(i = (n - 1) / 2; i >= 0; i--) heapify(a, n, i);
}

void heapSort(int *a, int n) {
  int i, aux;
  buildHeap(a, n);
  for(i = n - 1; i > 0; i--) {
      aux = a[0];
      a[0] = a[i];
      a[i] = aux;
      heapify(a, i, 0);
  }
}
```

Respostas:
Após a execução de `buildHeap(vet, 8)`, o vetor será ordenado de tal maneira que forme uma Heap Máxima, que é um tipo de árvore binária onde o maior elemento sempre é a raiz. O vetor fica ordenado de tal forma que o pai de qualquer elemento é seu `índice/2`, e seus filhos (se existirem) estão nas posições `índice*2` e `índice*2 + 1`. Logo o maior elemento estará na posição de `índice = 0`.
`vet_após_buildHeap = [8, 7, 5, 3, 6, 1, 4, 2]`

A complexidade de tempo no pior caso será **O(n log n)**, pois, `buildHeap()` tem complexidade de O(n), além disso, executa-se um laço `for` n vezes com a chamada de `heapify()` dentro, que possui complexidade de O(log n).
Somando O(n) + n*O(log n) = O(n) + O(n log n) = **O(n log n)**.

A complexidade de espaço no pior e melhor caso e no caso médio são iguais: **O(n).**
Vale ressaltar que o HeapSort é um algoritmo chamado de *in-place*, ou seja, ele não faz uso de estruturas de dados auxiliares além de variáveis fixas que tem complexidade de espaço igual a **O(1)**.

A complexidade de tempo se todos os elementos forem iguais será **O(n)**.
Isso se deve ao fato de que a construção da heap tem complexidade O(n), e todas as chamadas de `heapify()` dentro de `heapSort()` serão, no total, de complexidade O(n), pois o laço `for` executará n vezes, porém, nenhuma dessas chamadas entrará na chamada recursiva, já que `maior` sempre será igual a `i`.