

- 1) Como ficará o arranjo *vet* após a execução da chamada de *buildHeap* (*vet*, 8). onde $\text{int vet[]} = \{1, 6, 5, 3, 7, 8, 4, 2\}$;

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 vetor forem iguais? Explique sucintamente cada passo do cálculo dessas complexidades.

A função *heapify* restabelece a propriedade de *heap* da posição passada como parâmetro, a complexidade de tempo é $O(\log n)$. A função *buildHeap* constrói um *heap* no vetor, a complexidade de tempo 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/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);
```

```
    }
```

```
}
```