O PROBLEMA DA ORDENAÇÃO

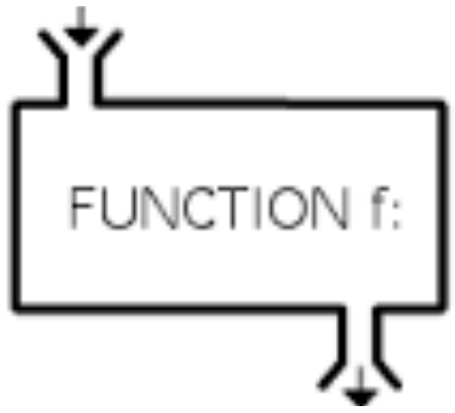
Bubblesort
Insertion Sort
Heaps binários; Heapsort
Merge sort

27-Maio-2021

Ordenação

- Pretende-se, dada uma sequência, obter a sequência equivalente mas ordenada;

45;-20;40;13;10;7;21



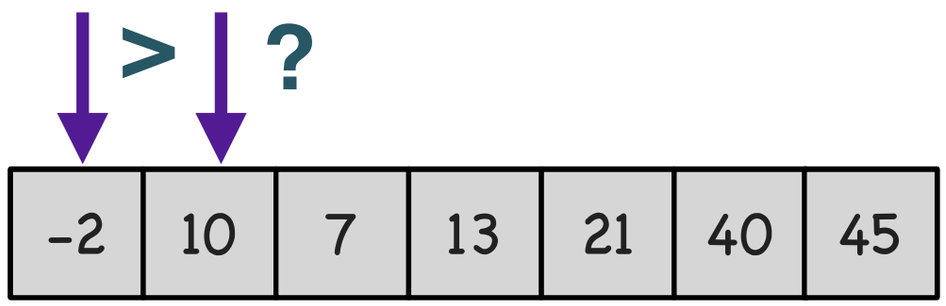
-20;7;10;13;21;40;45

- Todos os algoritmos apresentados receberão um array de inteiros e ordenam-no;

- Todos conhecemos algum método (algoritmo) de ordenação

BubbleSort

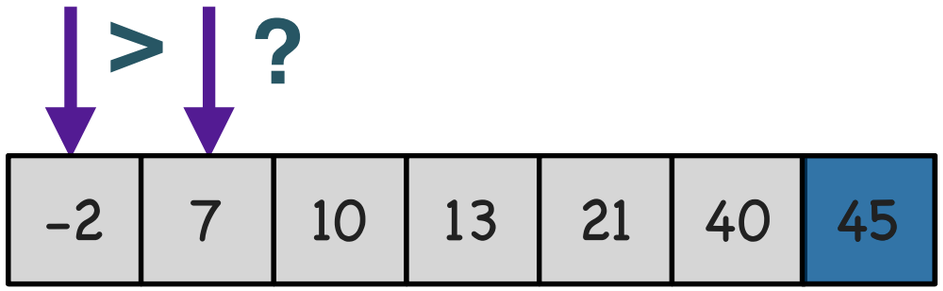
- Ordenamento por bolha;
- http://www.youtube.com/watch?v=MtcrEhrt_K0&feature=related
- Basta comparar um elemento com o seguinte e trocar se for maior



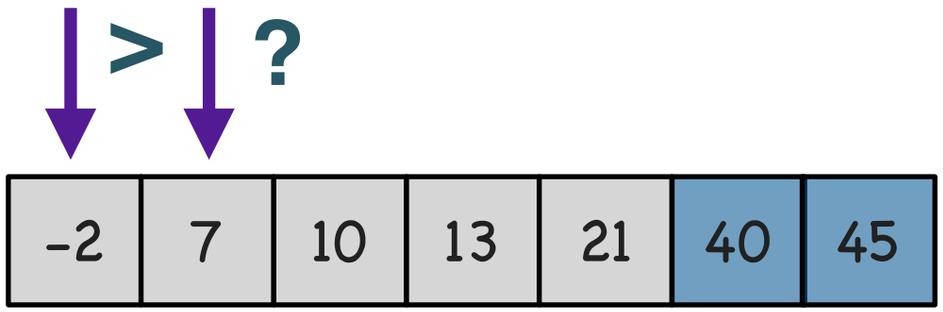
- Após uma primeira passagem:
  - O array não está ordenado mas **o maior elemento está no fim**

BubbleSort (cont.)

- 2ª Passagem

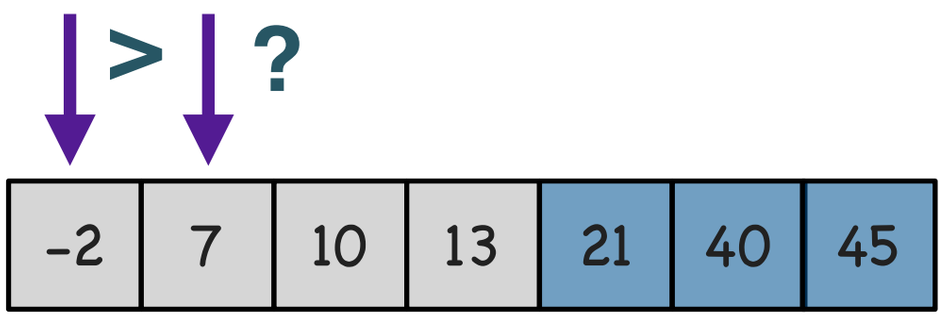


- 3ª Passagem

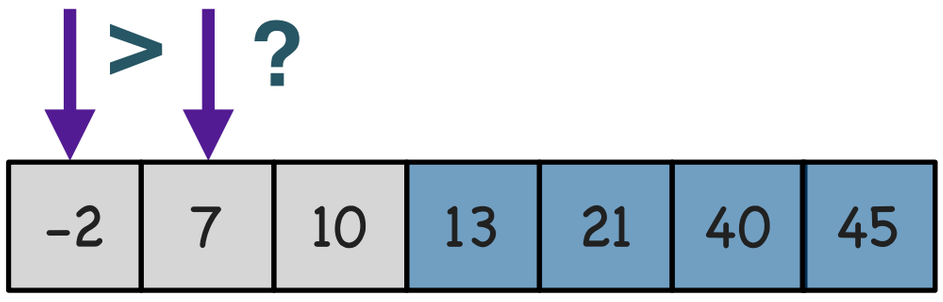


BubbleSort

- 4ª Passagem

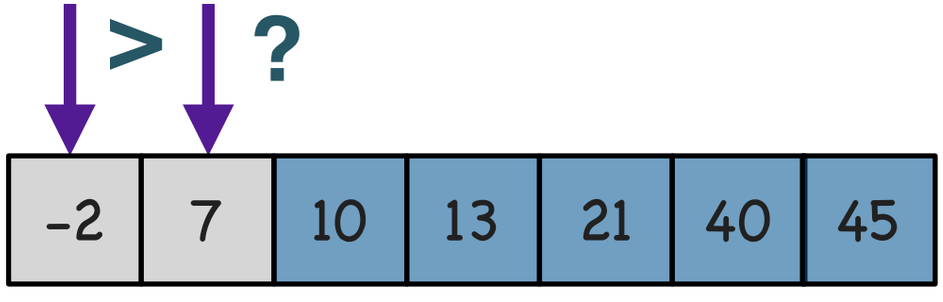


- 5ª Passagem



Array ordenado

- 6ª Passagem



Código - Bubblesort

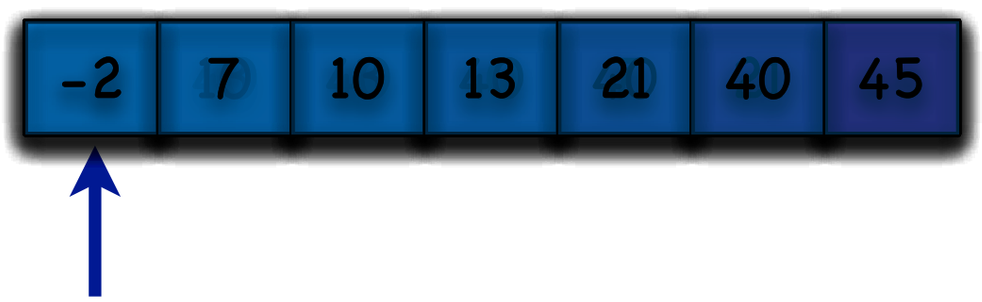
```
void Bubblesort(ElementType A[], int size){

    for (int i=1;i<size;i++){
        printf("Passagem %d: \n", i);
        for (int j=0;j<size-i;j++)
        if (A[j]>A[j+1]){
            printf("troca %d com %d\n",A[j],A[j+1]);
            Troca(&A[j],&A[j+1]);
        }
    }
}
```

$$T = O(size^2)$$

```
void Troca( ElementType *Lhs, ElementType *Rhs ){
    ElementType Tmp = *Lhs;
    *Lhs = *Rhs;
    *Rhs = Tmp;
}
```

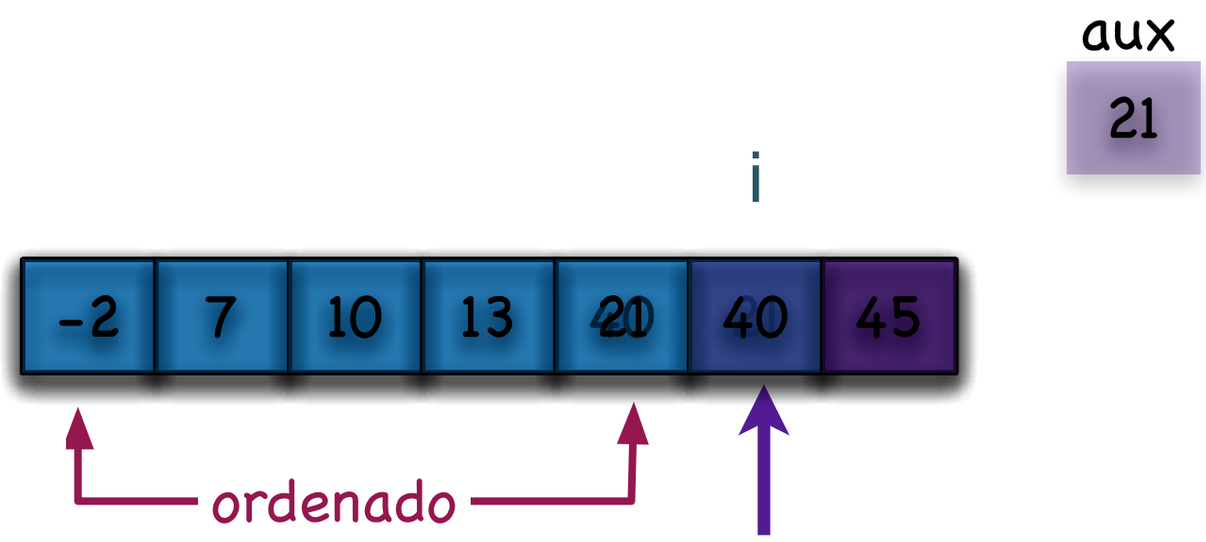
Insertion Sort

- O algoritmo que usamos quando ordenamos:
  - cartas de jogar
    - **https://www.youtube.com/watch?v=uMqVuEEWJv4**
    - **http://www.youtube.com/watch?v=FrOSmtNOIJM**
  - testes dos alunos



Insertion sort

- Caso Geral:




```
ElementType Aux=A[i];
int j=i;
while (j>0 && Aux<A[j-1]){
     A[j]=A[j-1];
      j--;
}
A[j]=Aux;
```

Código

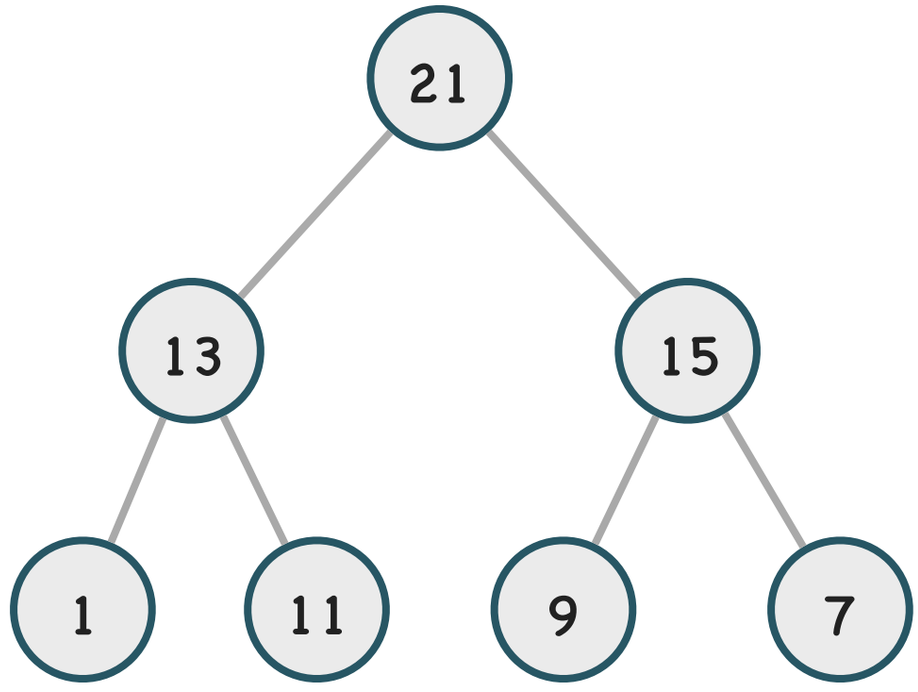
```
void InsertionSort(ElementType A[], int size){
    for (int i=1; i<size;i++){
        printf("Passagem %d \n",i);
        ElementType Aux=A[i];
        int j=i;
        while (j>0 && Aux<A[j-1]){
            A[j]=A[j-1];
            j--;
        }
        A[j]=Aux;
        printArrayElementType(A, i+1);
    }
}
```

$O(size^2)$ - pior dos casos

$O(size)$- melhor dos casos

Heapsort

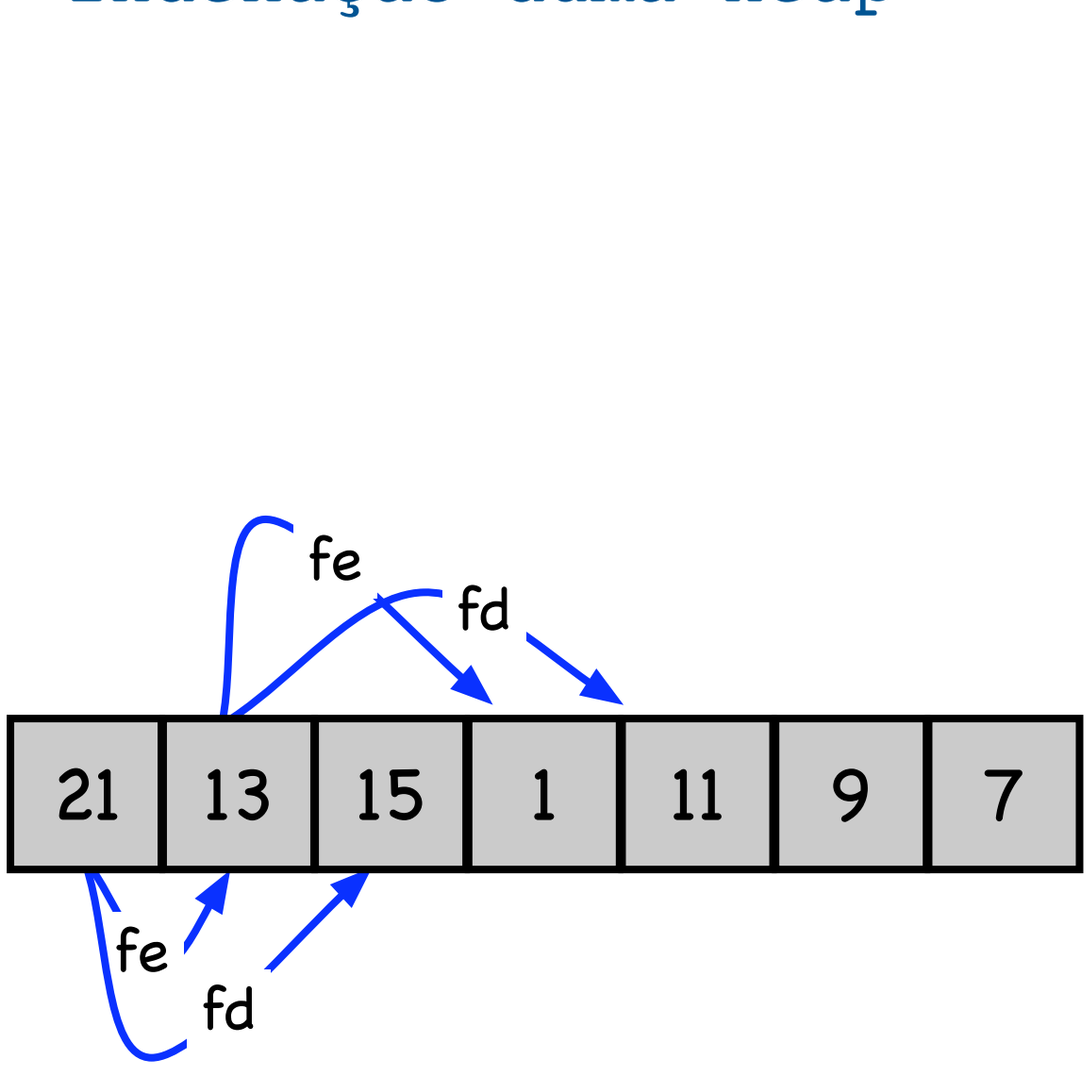
- Definição de Heap: árvore binária completa, com excepção para o nível de profundidade máximo;
  - O pai é maior que os filhos:
    - valor(FE(x))< valor(x)
    - valor(FD(x))< valor(x)
- Representação em array

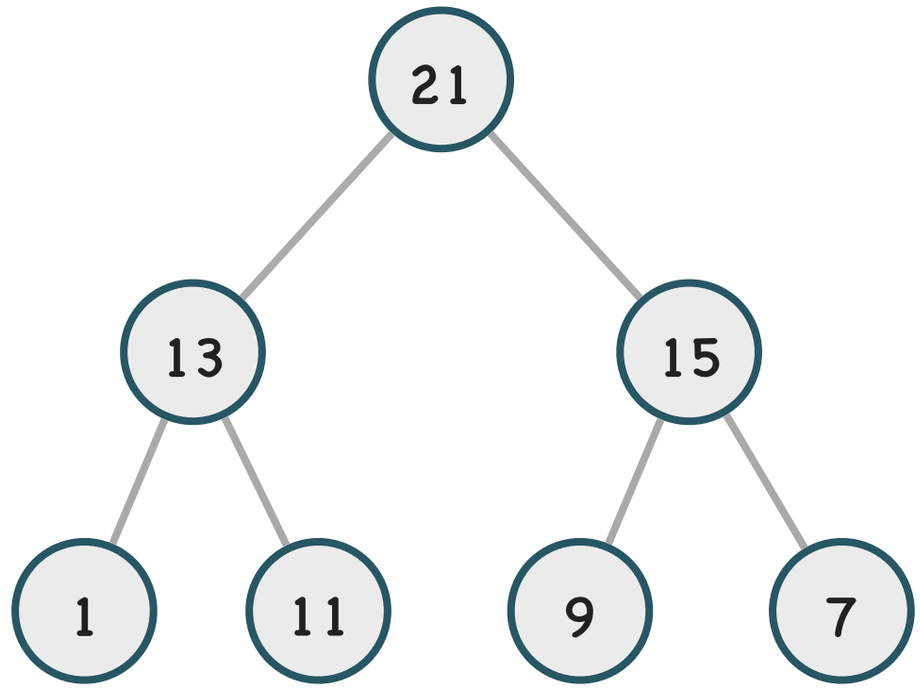


21	13	15	1	11	9	7

Heap

- Indexação duma heap





fe(x)=2x

fd(x)=2x+1

pai(x)= ⌊x/2⌋

Heapsort

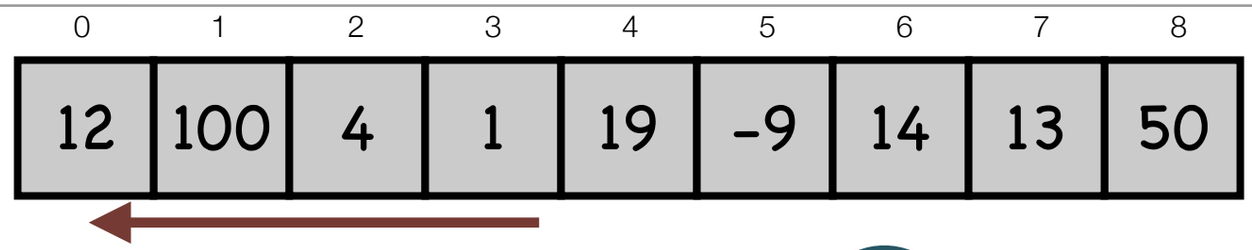
- Definir a operação "heapificar":
  - Substituir o pai pelo maior dos filhos, e heapificar o filho

- Dado um array como construir uma heap ?

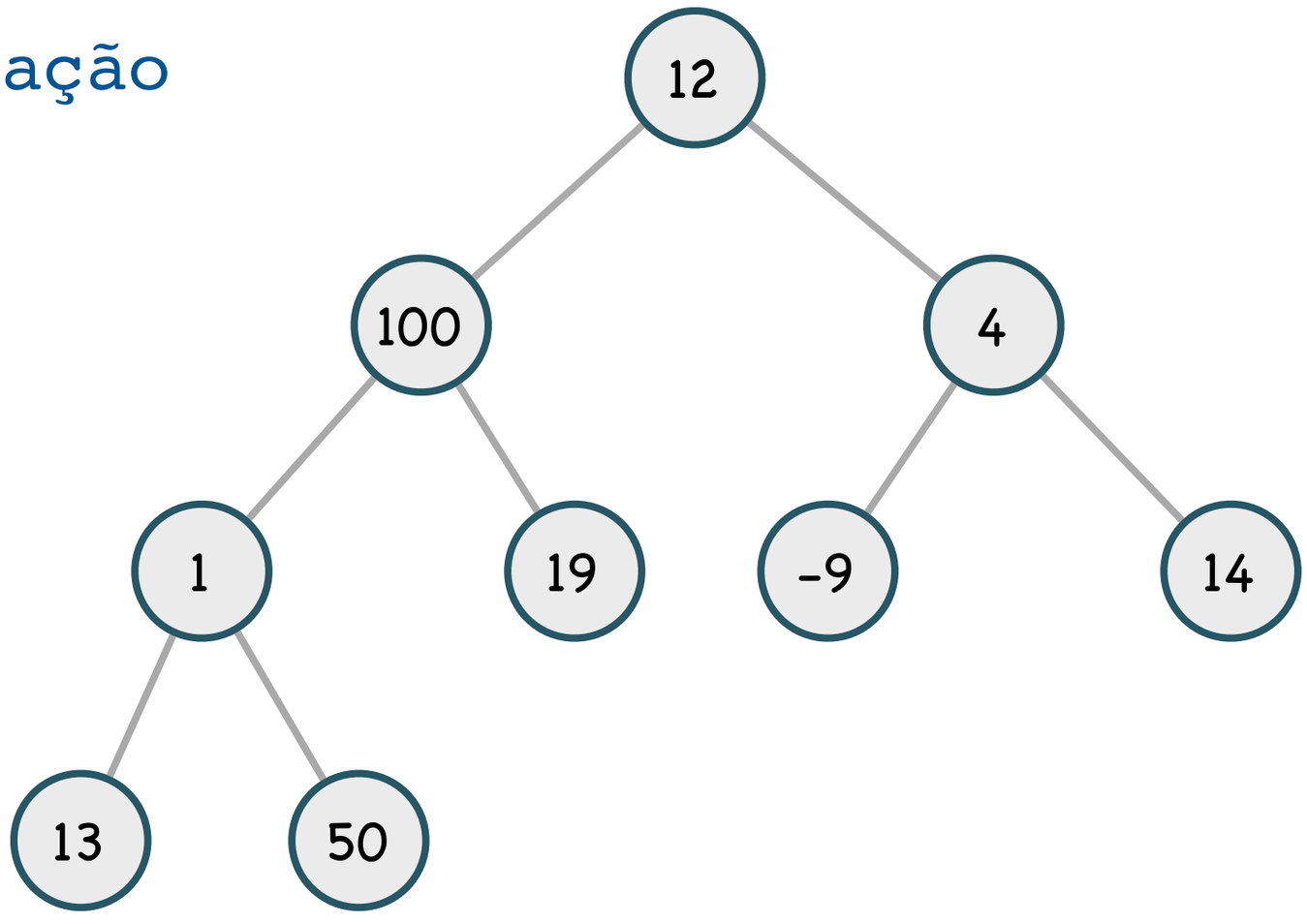
12	100	4	1	19	-9	14	13	50

Construir a Heap

- O array A:



- tem a visualização

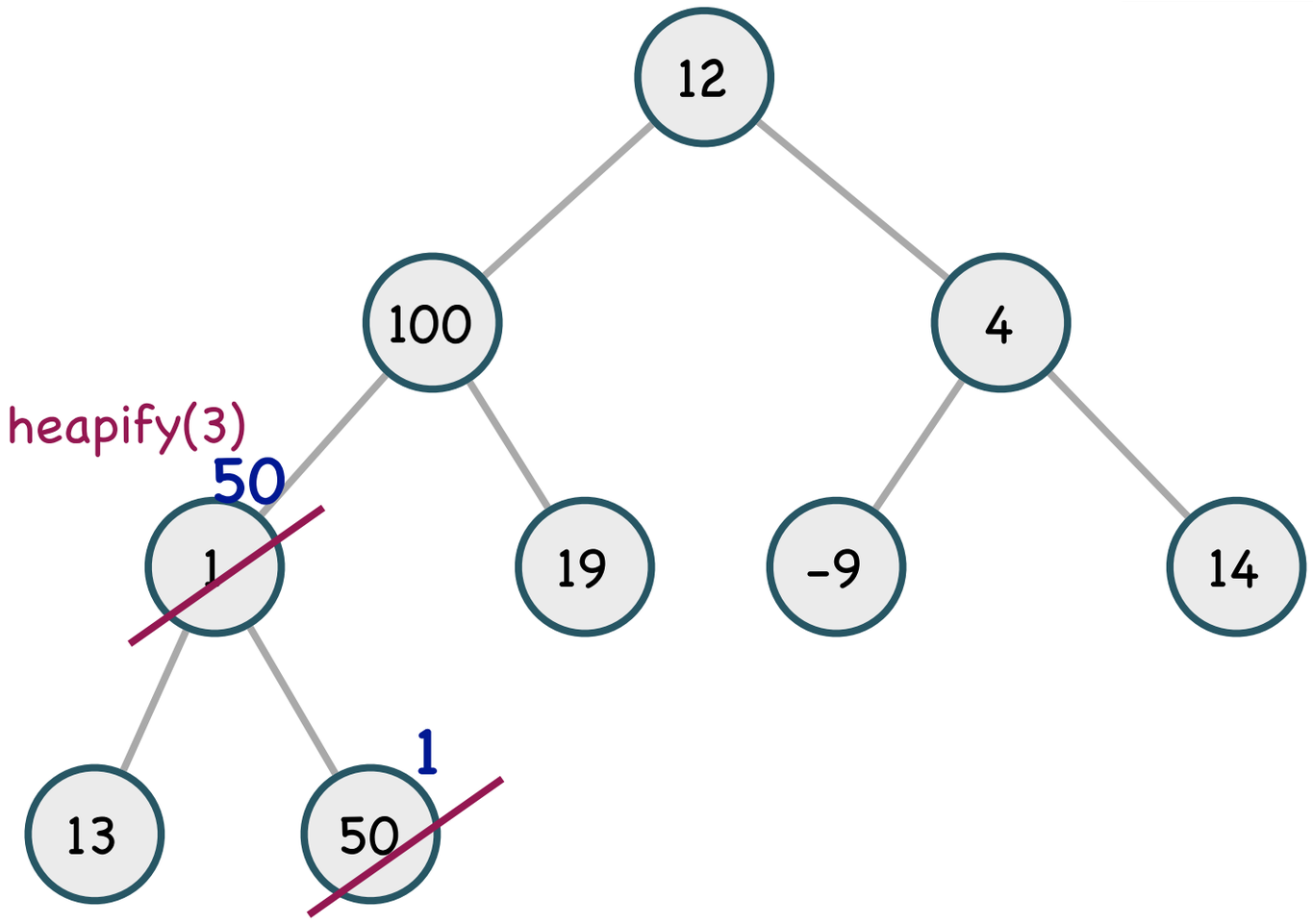


- Vamos heapificar todos nós (A[índice no array]), começando pelos nós mais baixos(de maior índice no array) desde que tenham filhos!

Construir a heap

- heapificar índice 3, índice 2, índice 1, índice 0

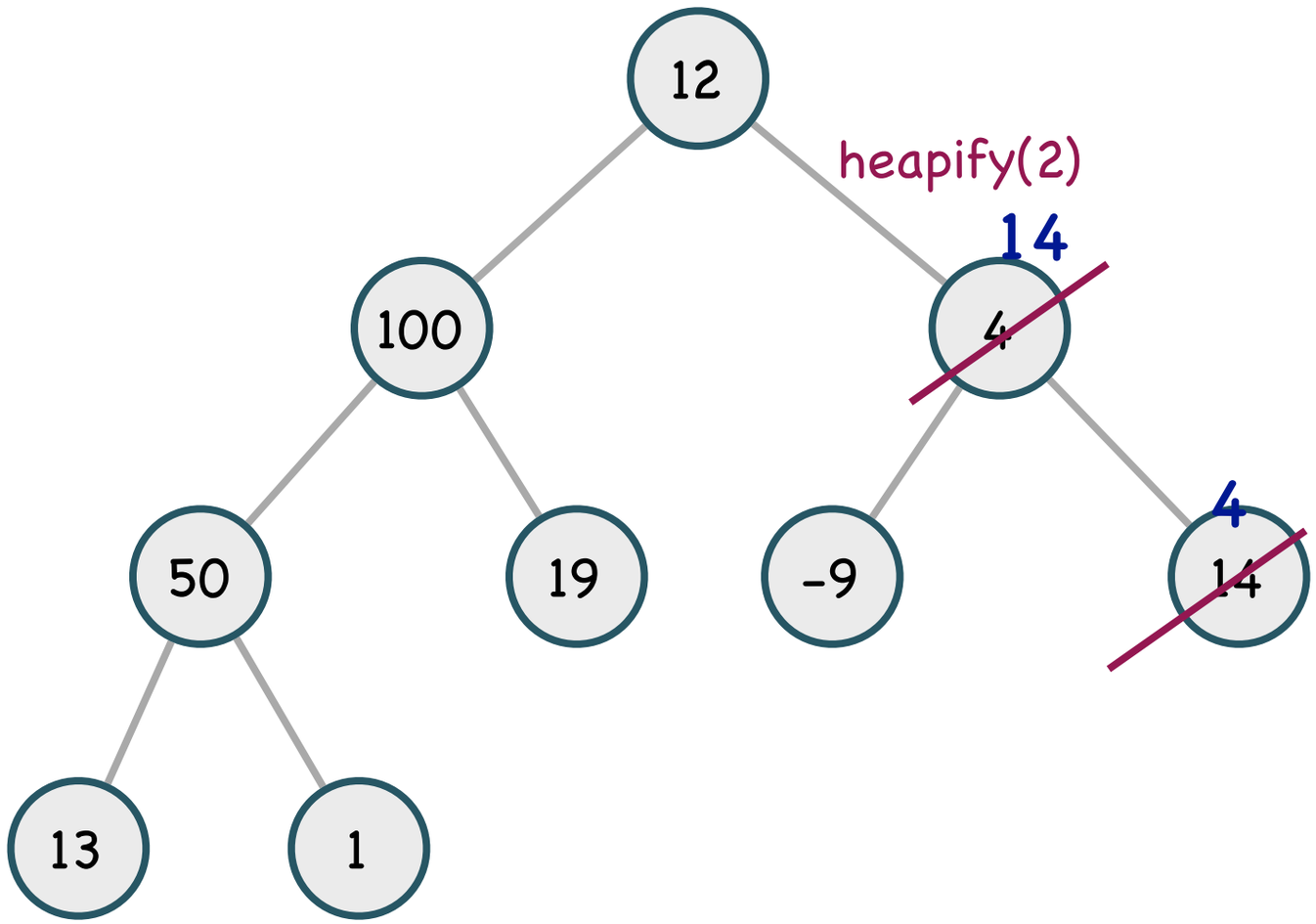
0	1	2	3	4	5	6	7	8
12	100	4	1	19	-9	14	13	50



Construir a heap

0	1	2	3	4	5	6	7	8
12	100	4	50	19	-9	14	13	1

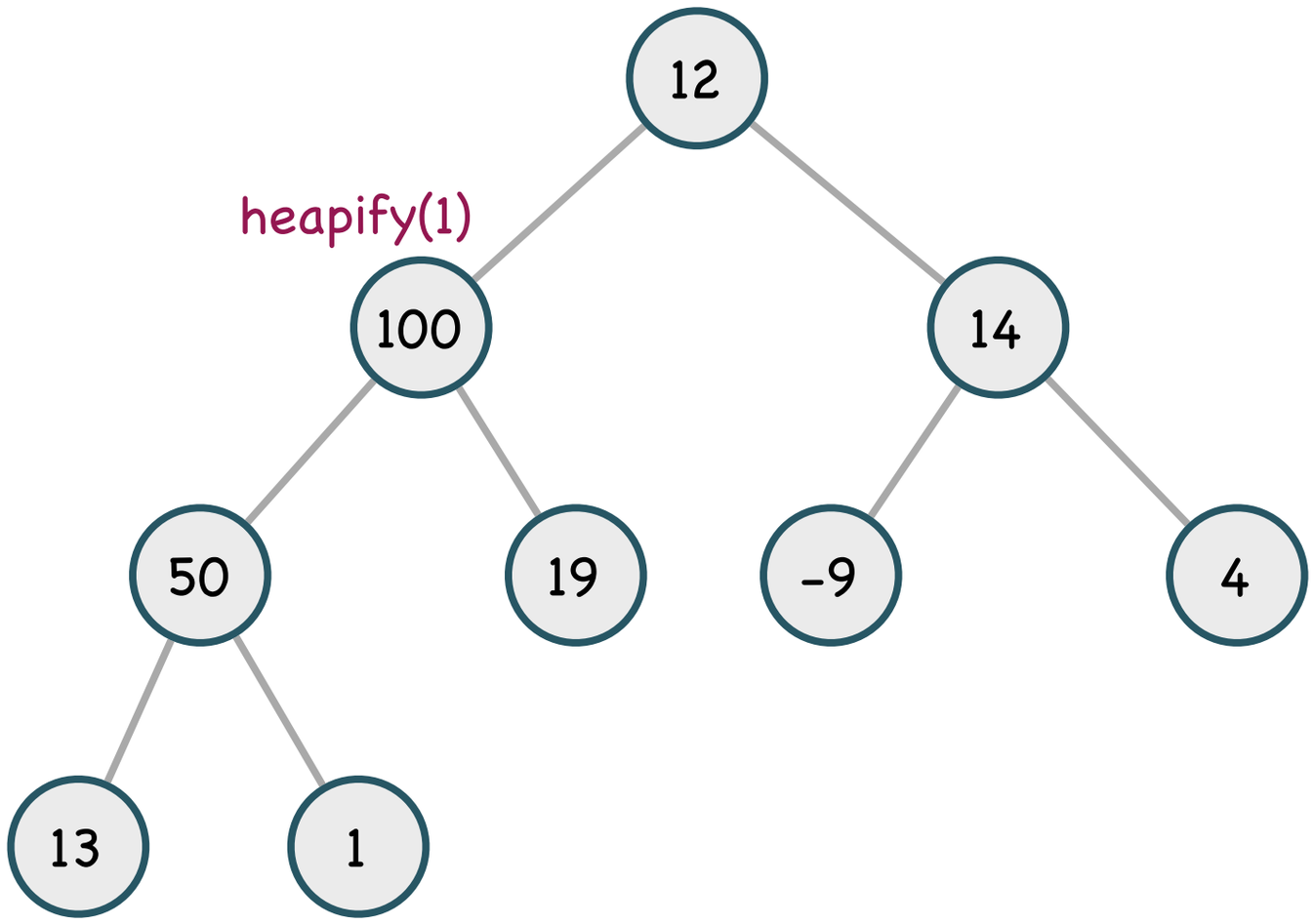
- Após heapificar(3)



Construir a heap

0	1	2	3	4	5	6	7	8
12	100	14	50	19	-9	4	13	1

- Após heapificar(2)

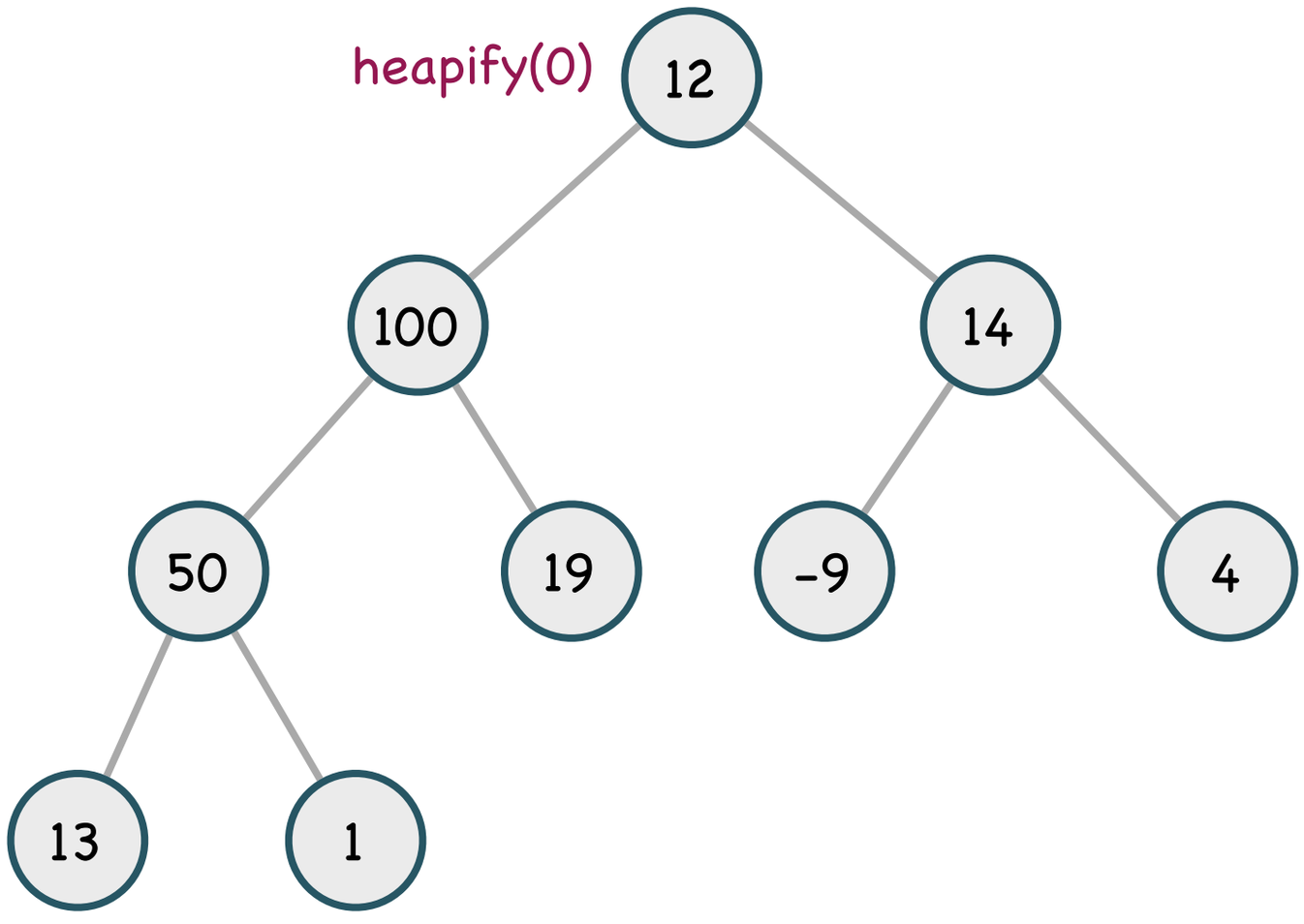


- e vamos heapificar(1)

Construir a heap

0	1	2	3	4	5	6	7	8
12	100	14	50	19	-9	4	13	1

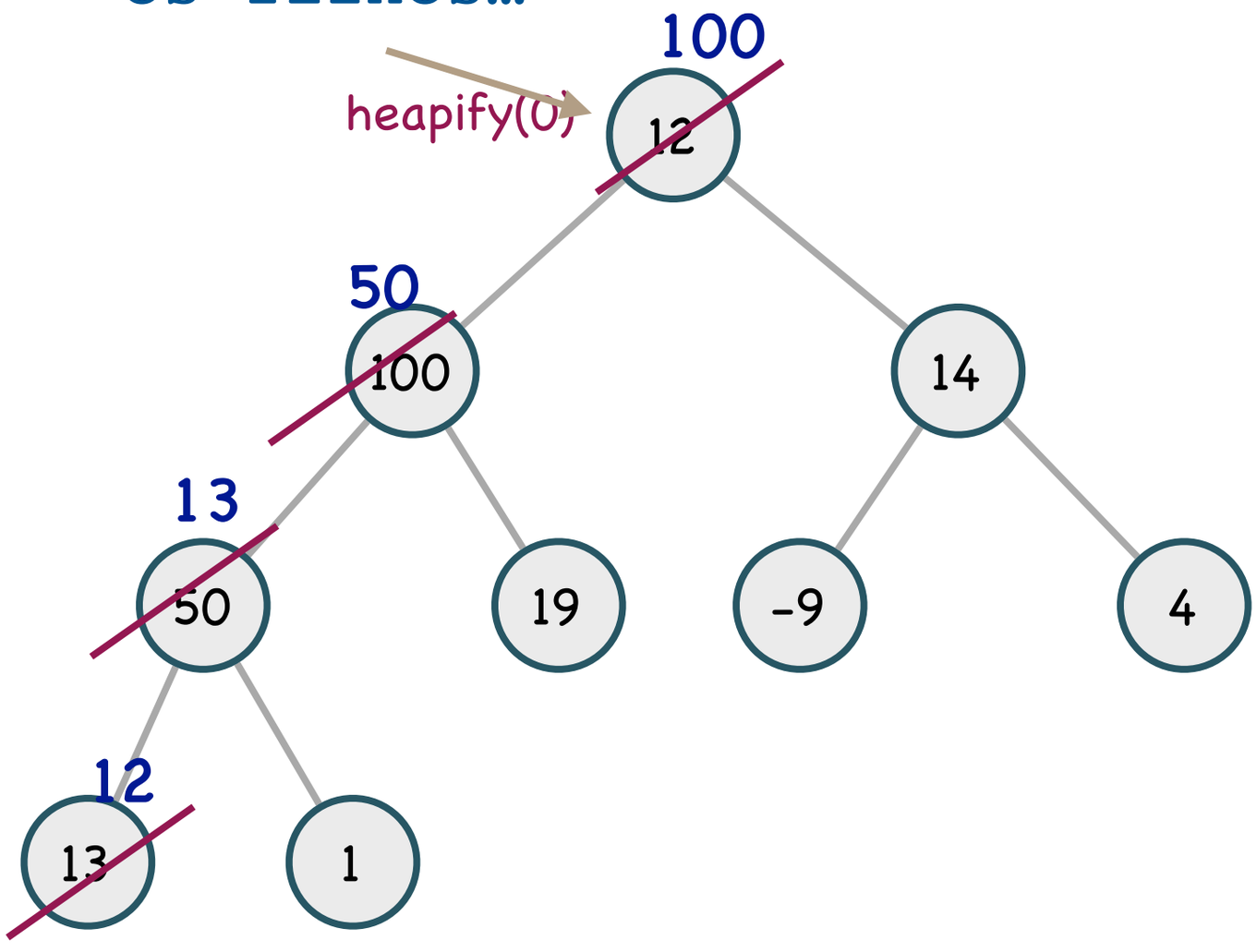
- Após heapificar(1), não muda nada o Pai é maior que os filhos…



- e vamos heapificar(0)

Construir a heap

- Após heapificar(1), não muda nada o Pai é maior que os filhos…



0	1	2	3	4	5	6	7	8
100	50	14	13	19	-9	4	12	1

heapify(0)
Guardar VPai=12

MaiorFilhos? 100

100 > 12 ?

MaiorFilhos? 50

50 > 12 ?

MaiorFilhos? 13

13 > 12 ?

Actualizo index com VPai

Heapsort

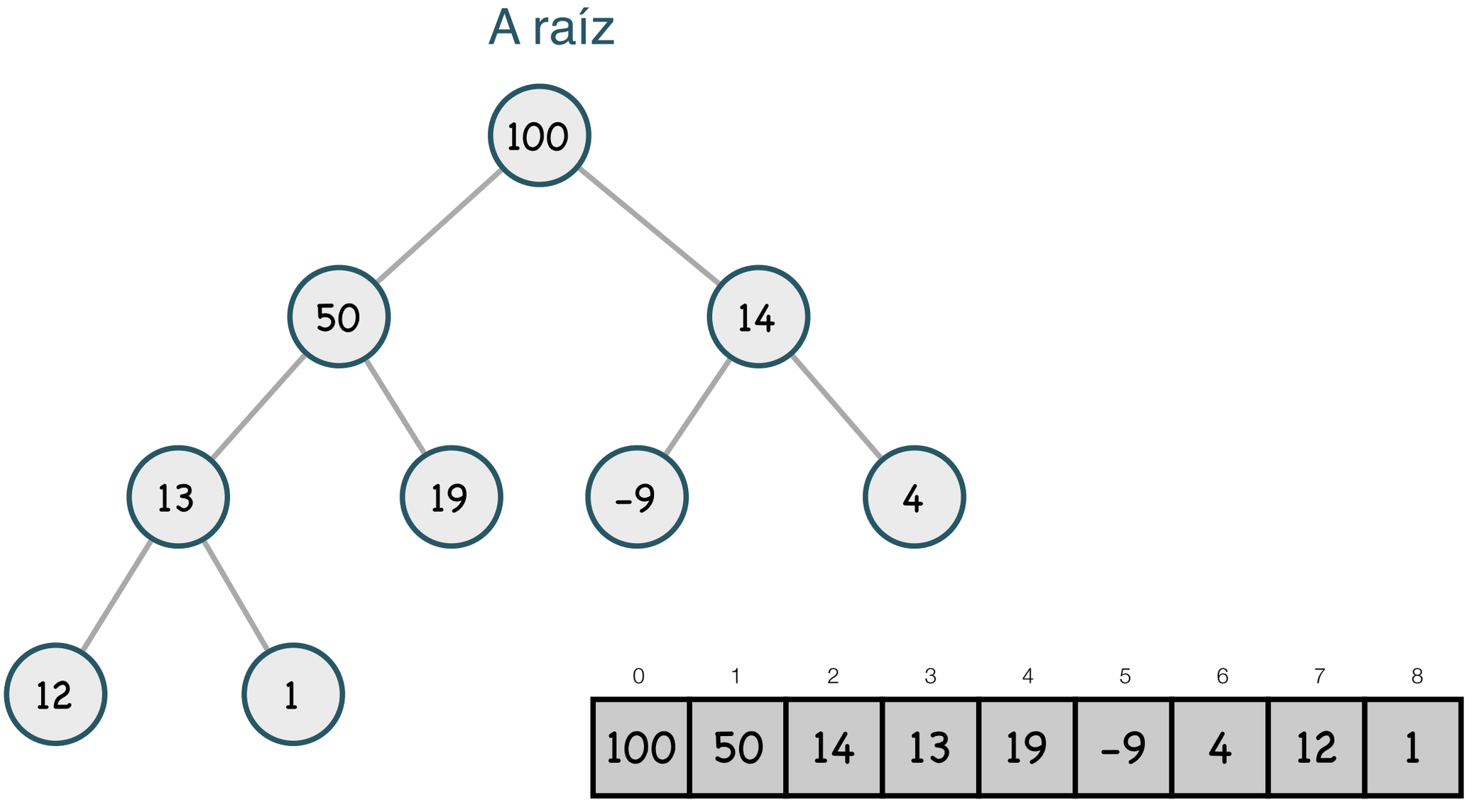
```
#define fesq( i )  ( 2 * ( i ) + 1 )
```

$O(log(N))$

```
void heapify2( ElementType A[ ], int i, int N ){
    ElementType VPai;
    int iFilho;
    VPai=A[i];
    while(fesq(i)<N){
        iFilho=fesq(i);
        if(iFilho!=N-1 && A[iFilho+1]>A[iFilho] )
            iFilho++; //índice do maior dos filhos

        if(VPai<A[iFilho])
           A[i]=A[iFilho];
        else
            break;
        i=iFilho;
    }
    A[i]=VPai;
}
```

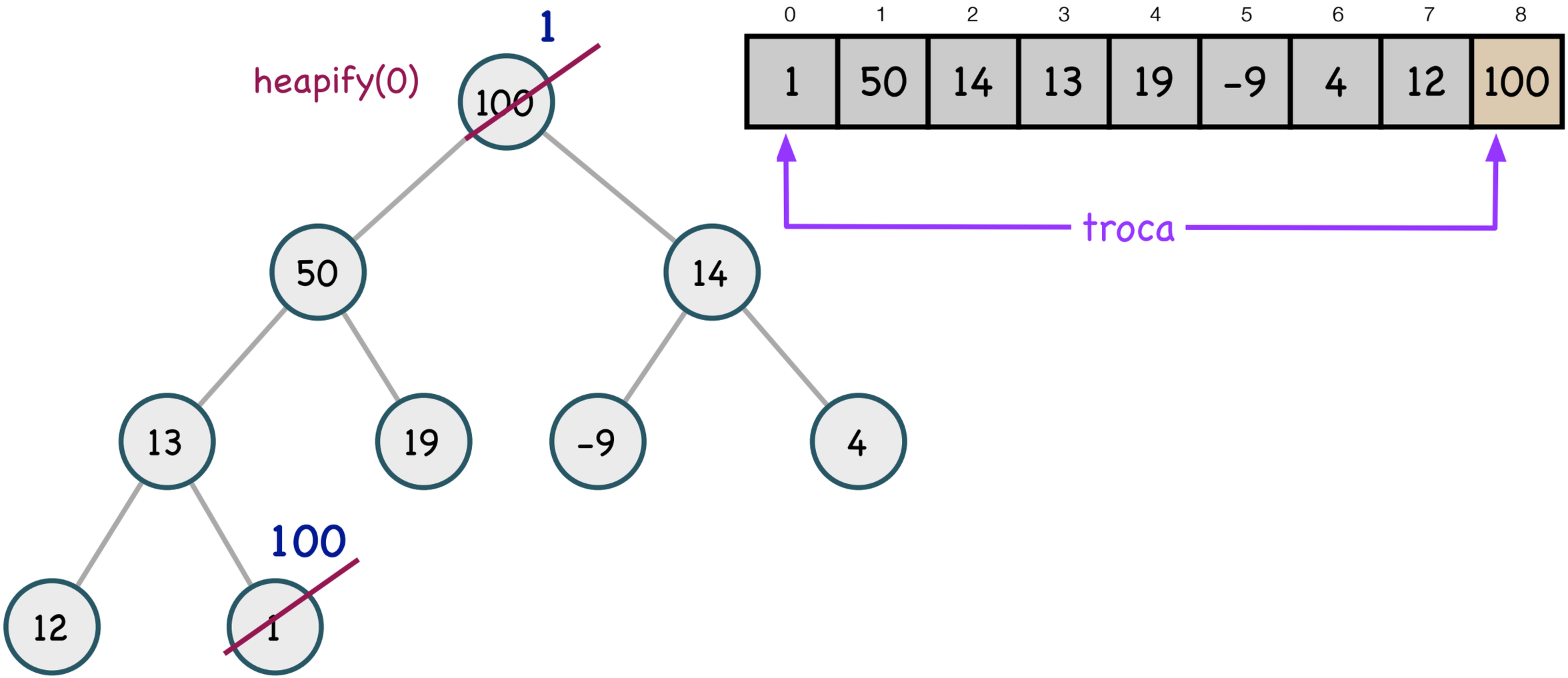
A HEAP

- Que posição ocupa o maior elemento da heap ?



Ordenar usando a heap?

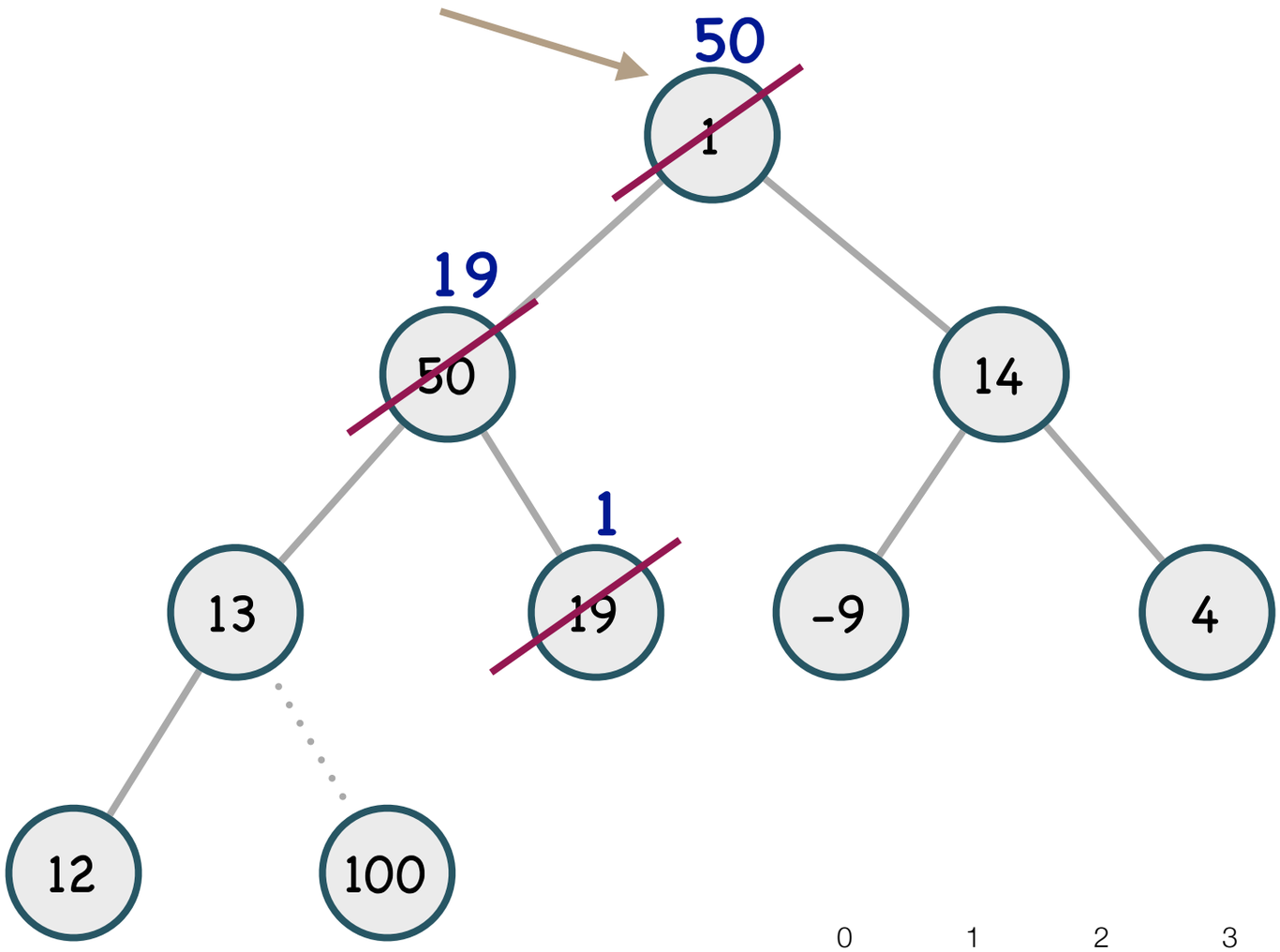
- Trocar o maior elemento (raiz da heap) com o último. Heapificar a nova raíz.



Heapsort

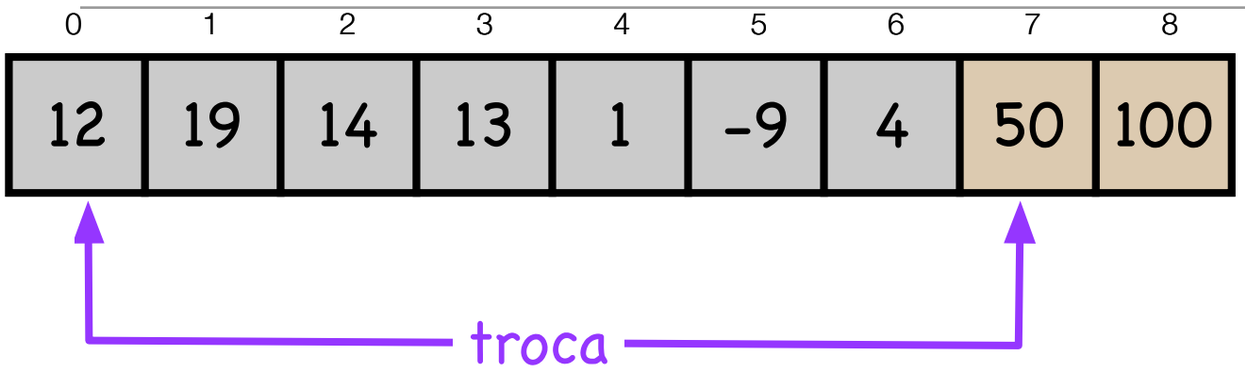
0	1	2	3	4	5	6	7	8
1	50	14	13	19	-9	4	12	100

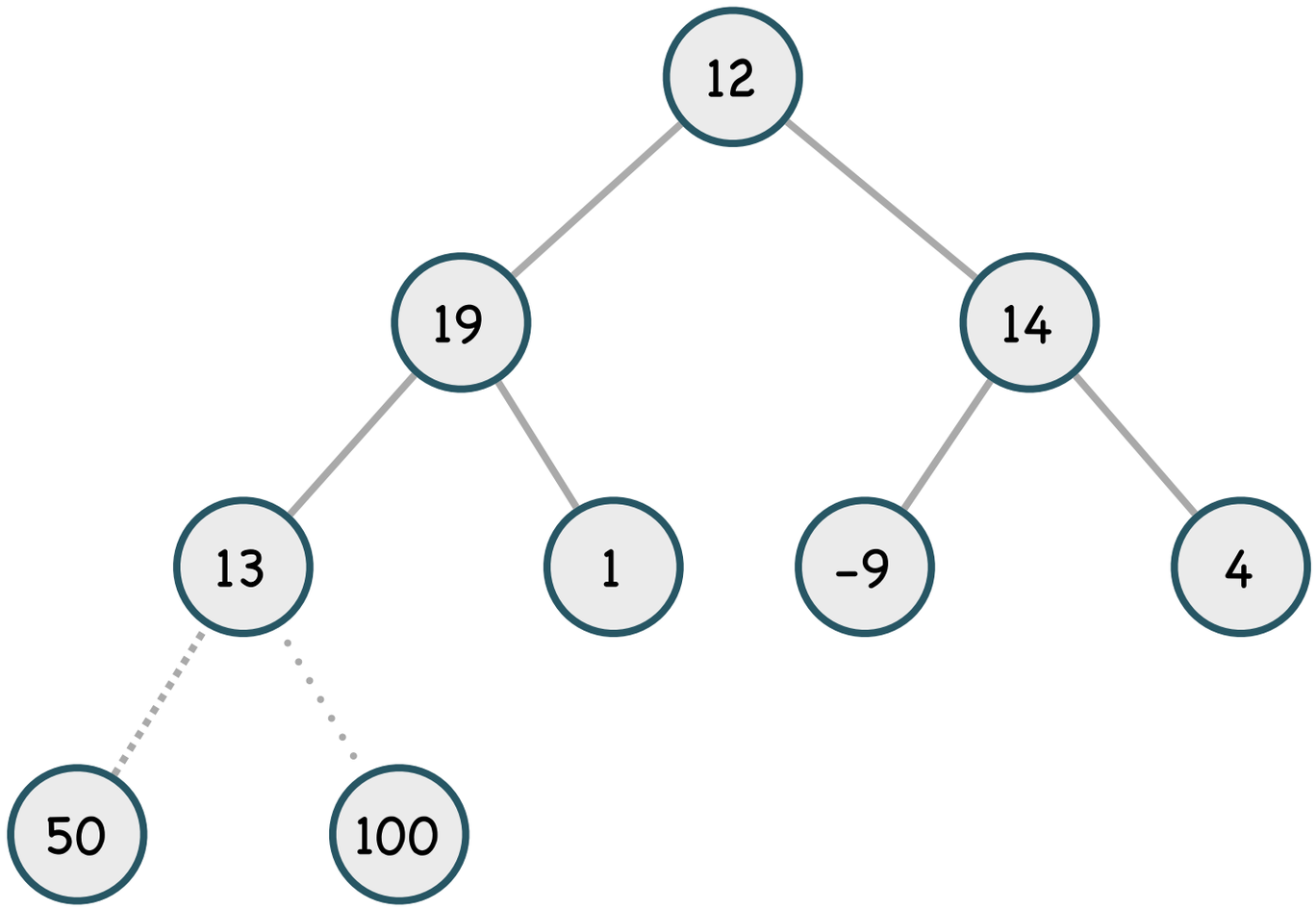
heapify(0)
Guardar VPai=1



0	1	2	3	4	5	6	7	8
50	19	14	13	1	-9	4	12	100

Heapsort

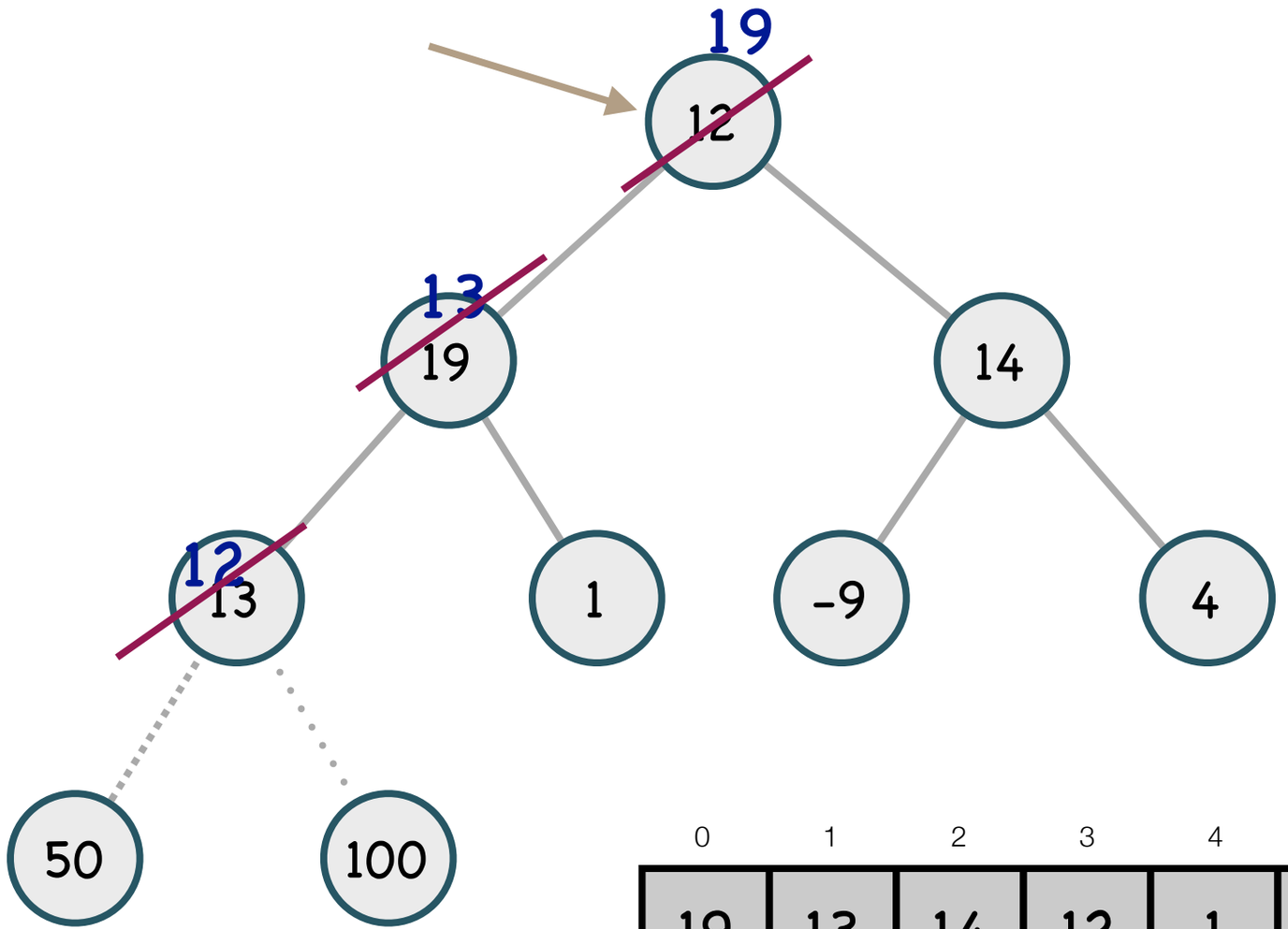




Heapsort

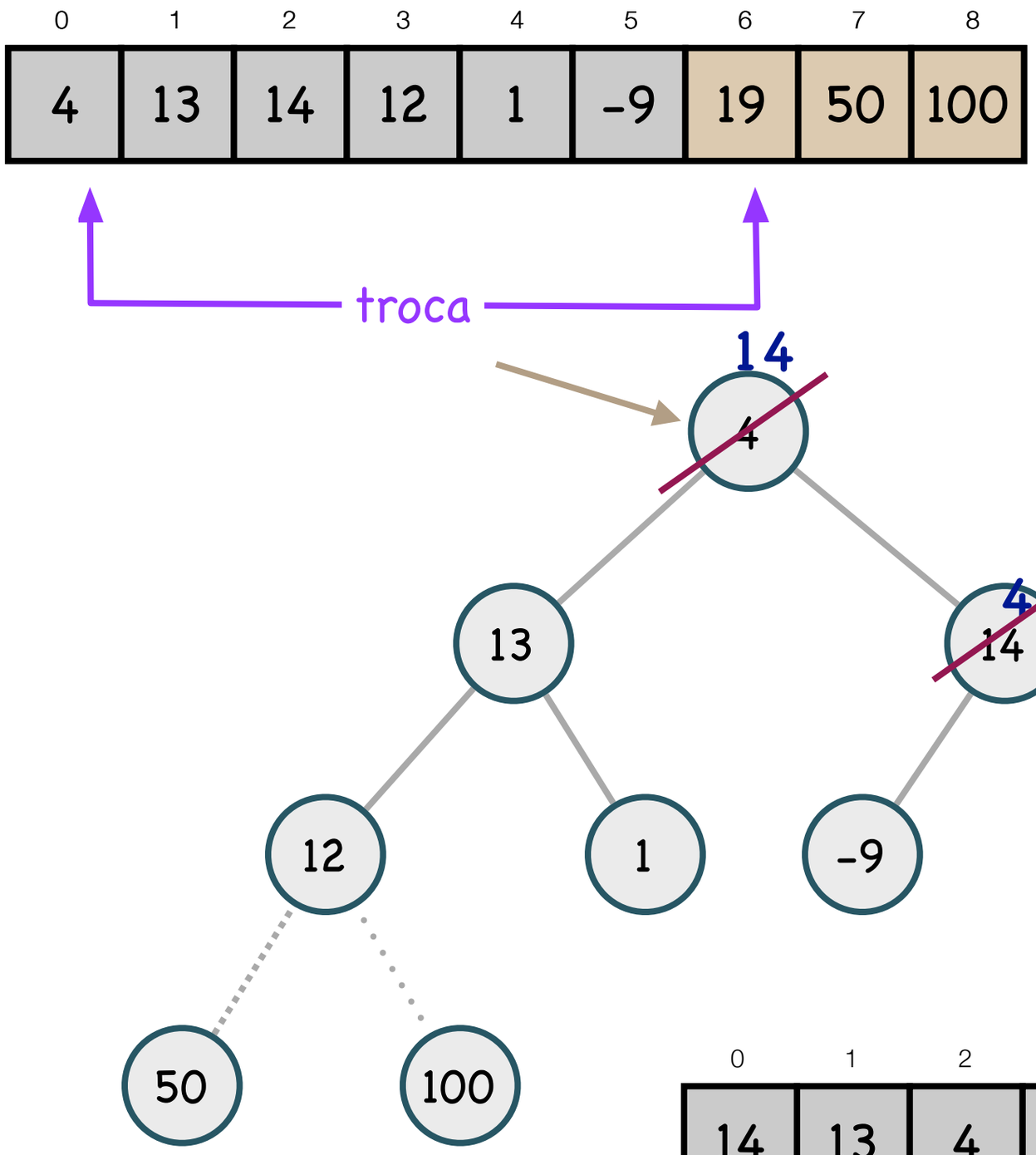
0	1	2	3	4	5	6	7	8
12	19	14	13	1	-9	4	50	100

heapify(0)
Guardar VPai=12



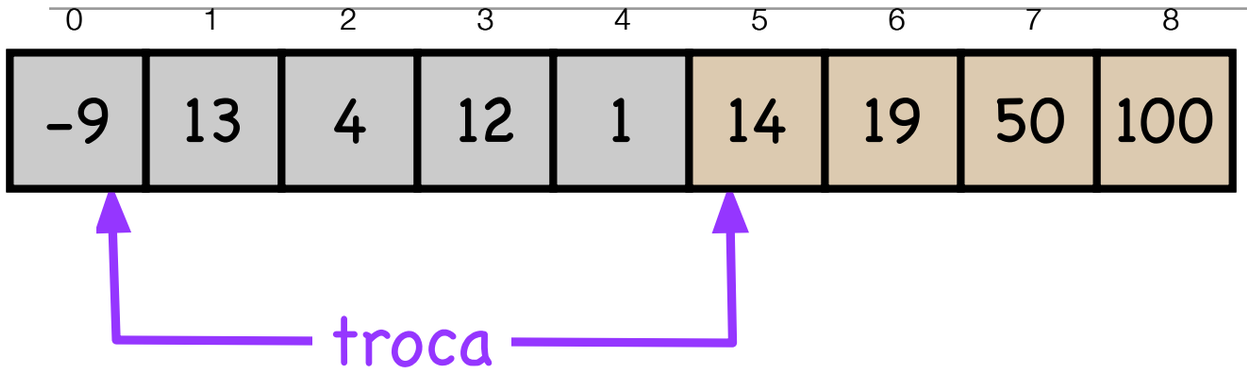
0	1	2	3	4	5	6	7	8
19	13	14	12	1	-9	4	50	100

Heapsort



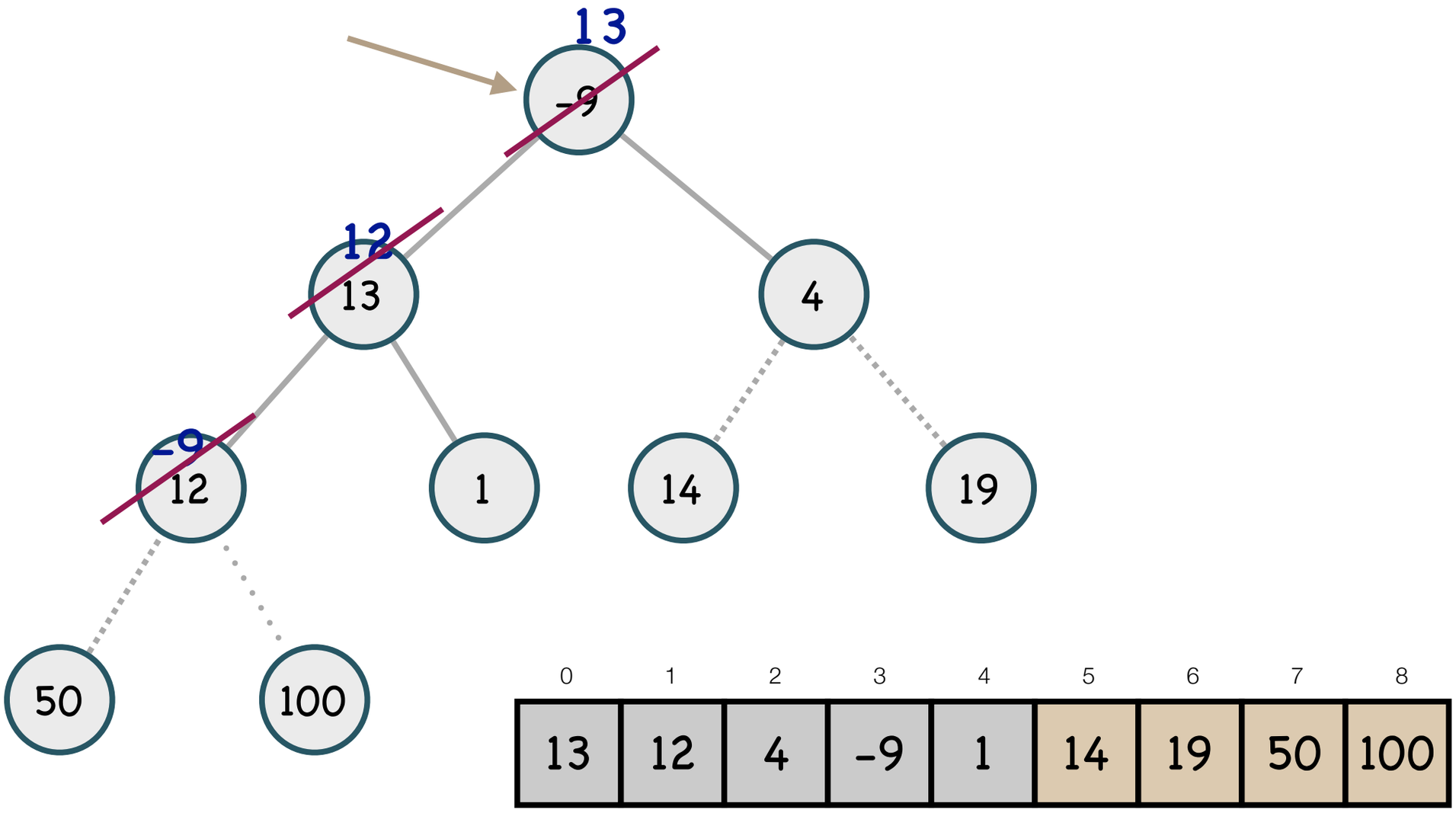
heapify(0)
Guardar VPai=4

Heapsort

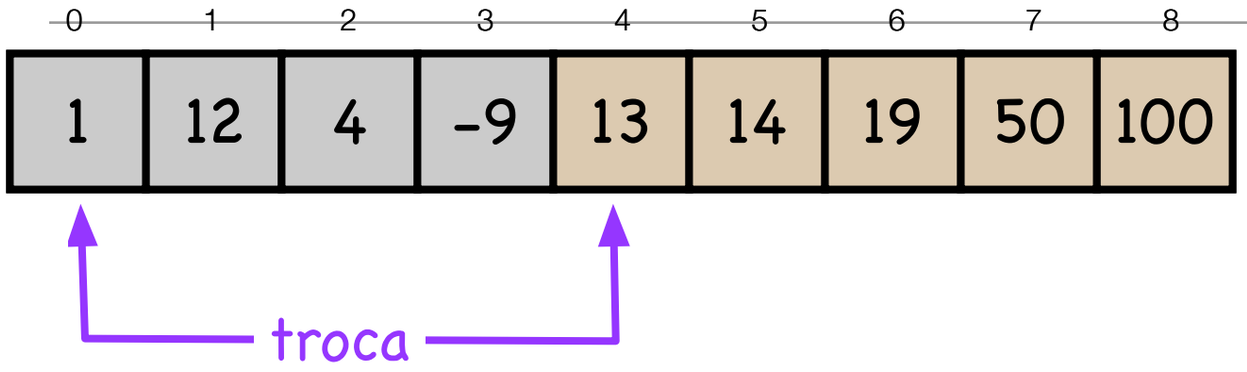


heapify(0)

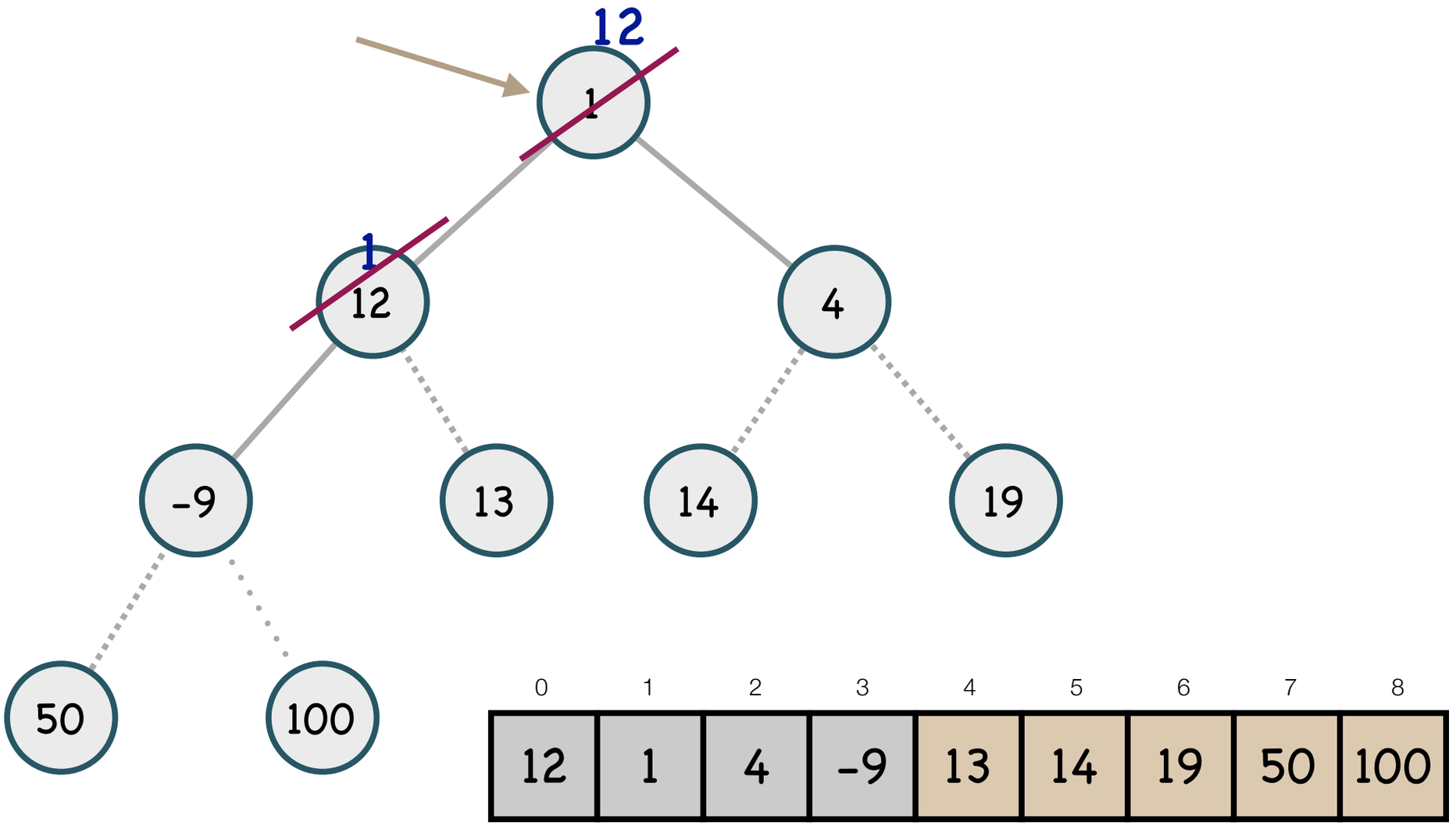
Guardar VPai=-9



Heapsort



heapify(0)
Guardar VPai=1

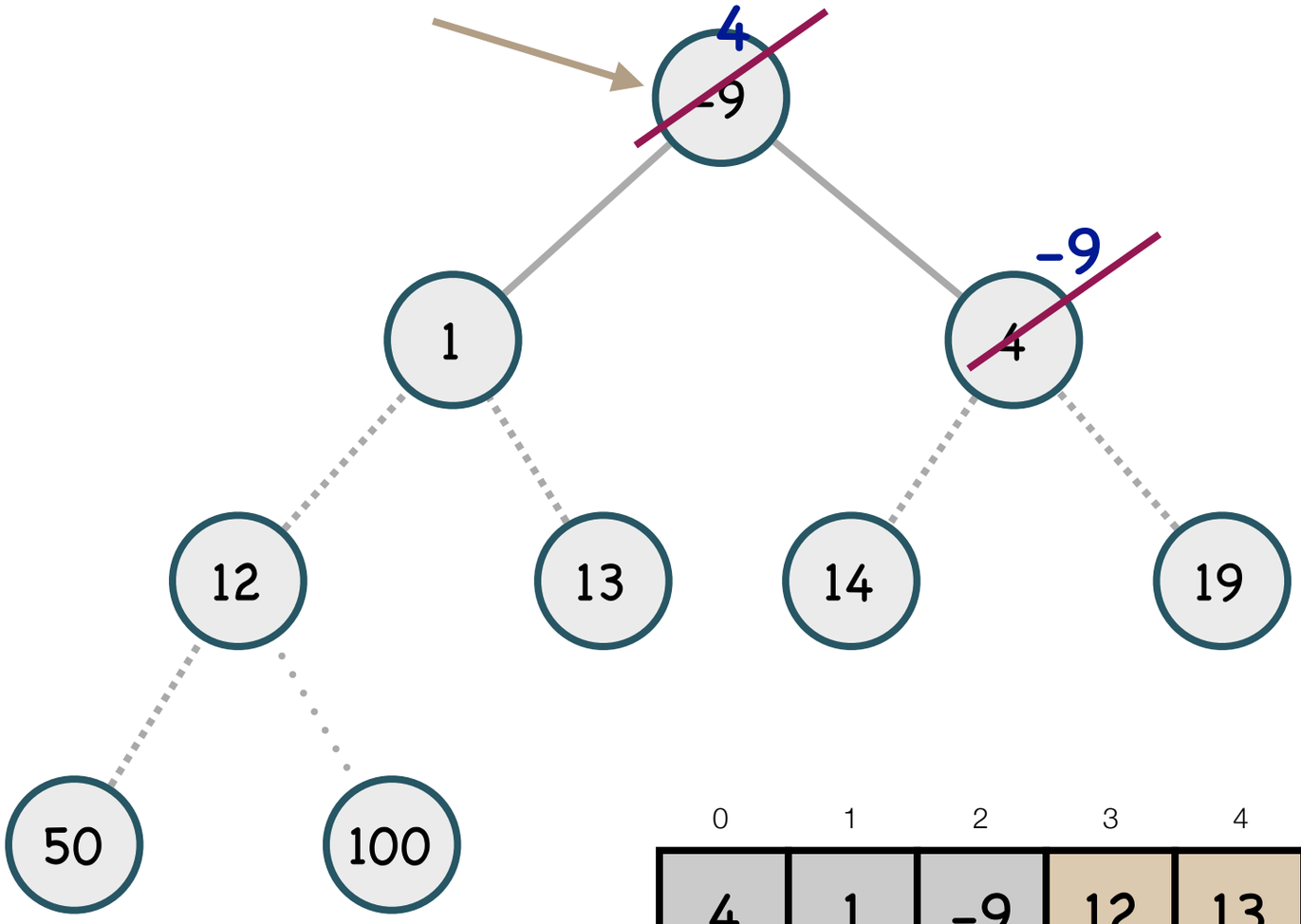


HEAPSORT

0	1	2	3	4	5	6	7	8
-9	1	4	12	13	14	19	50	100

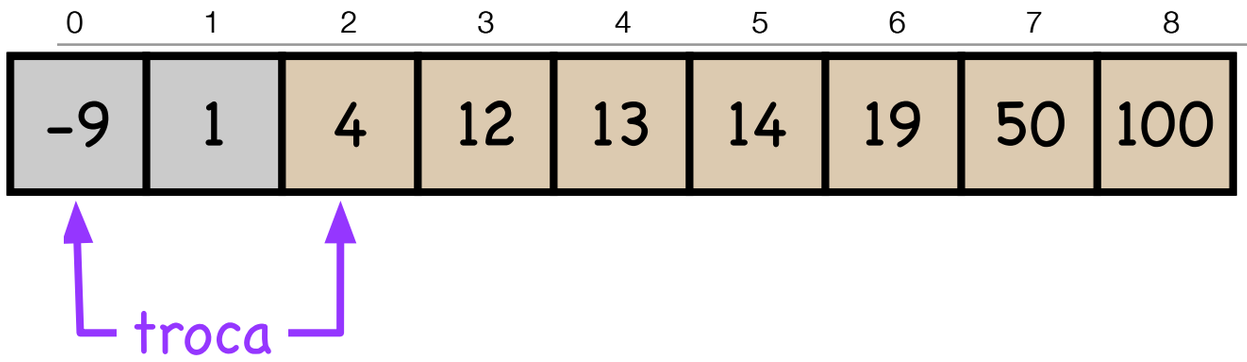
troca

heapify(0)
Guardar VPai=-9

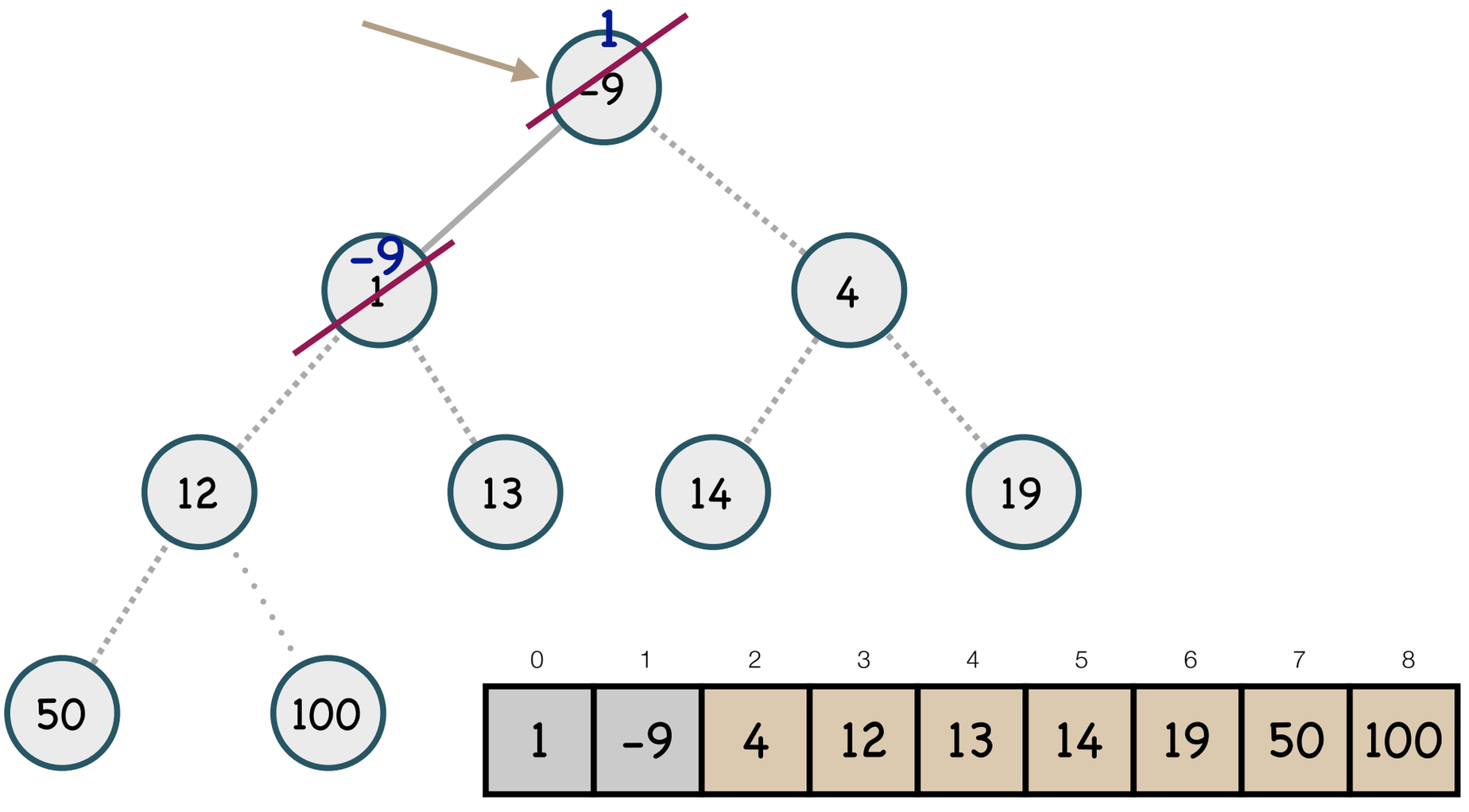


0	1	2	3	4	5	6	7	8
4	1	-9	12	13	14	19	50	100

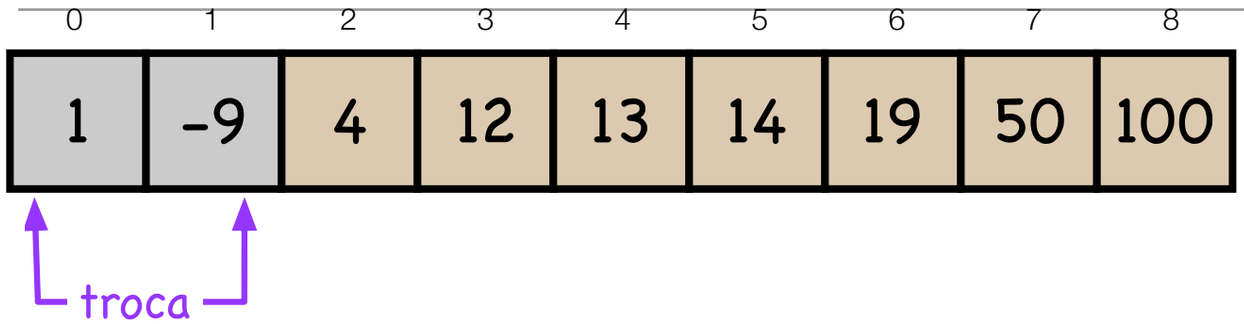
Heapsort

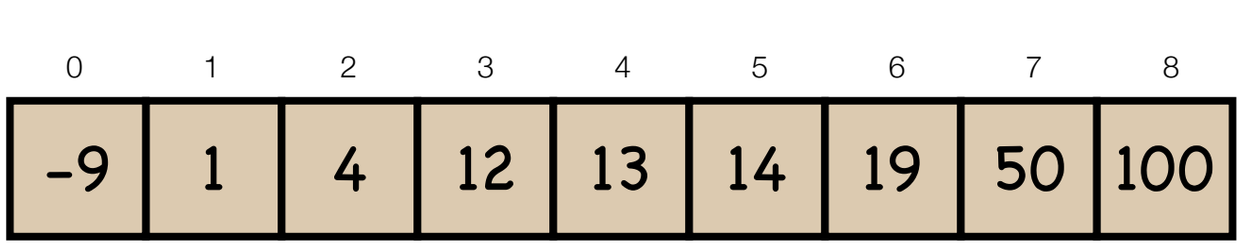


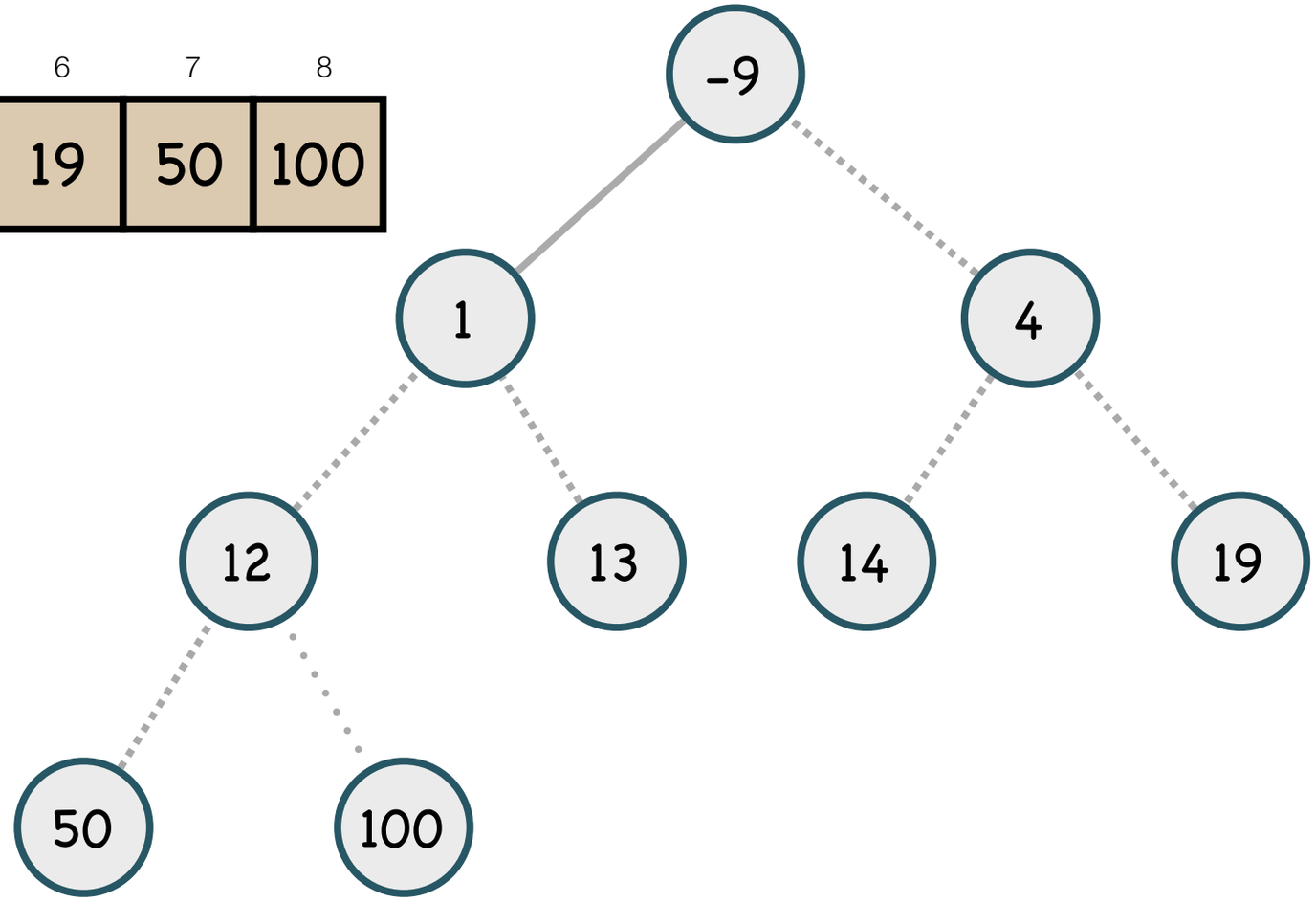
heapify(0)
Guardar VPai=-9



Heapsort


0 1 2 3 4 5 6 7 8
1 -9 4 12 13 14 19 50 100
troca
0 1 2 3 4 5 6 7 8
-9 1 4 12 13 14 19 50 100


0 1 2 3 4 5 6 7 8
-9 1 4 12 13 14 19 50 100


-9
1
4
12
13
14
19
50
100

Heapsort

$O(Nlog(N))$

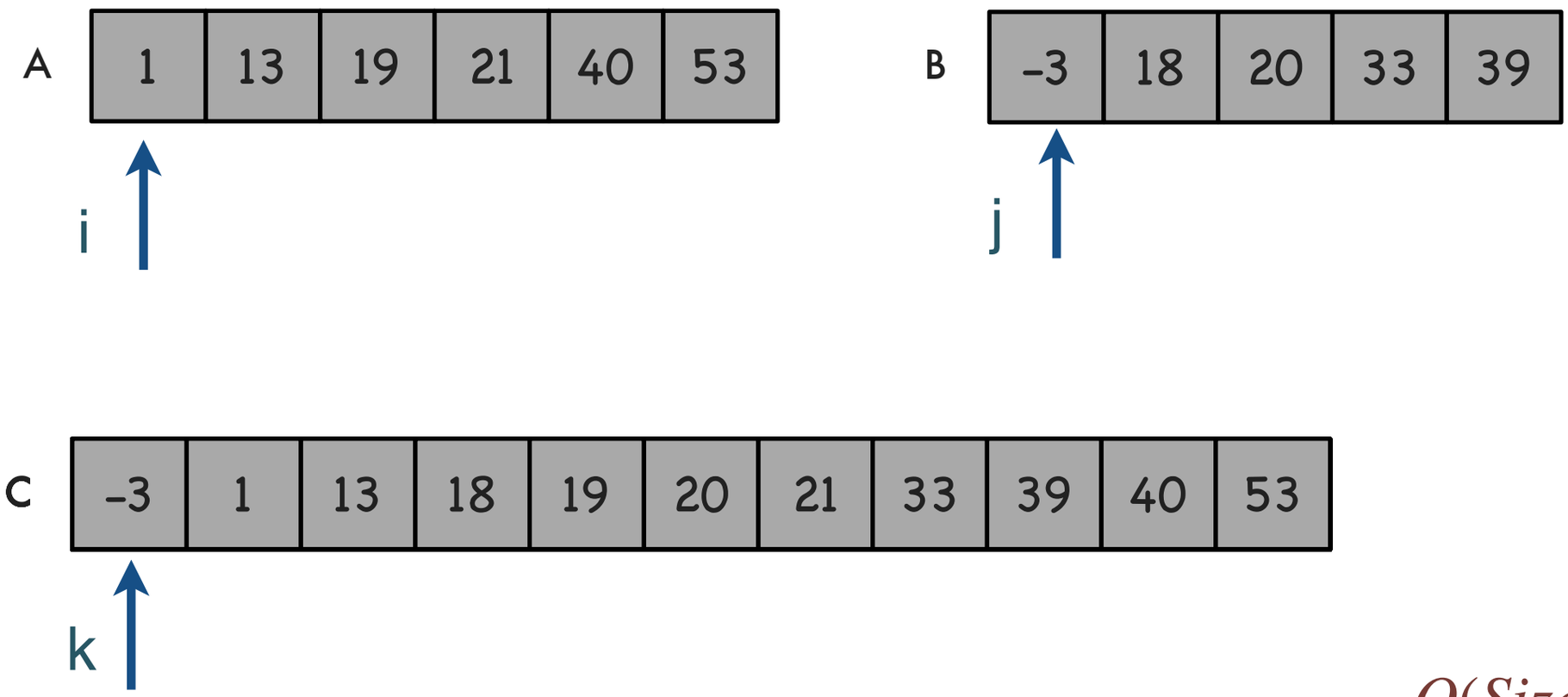
```
void Heapsort(ElementType A[], int N){
    int i;

    for( i = N / 2; i >= 0; i-- )  /* BuildHeap */
        heapify( A, i, N );

    for( i = N - 1; i > 0; i-- ){
        Troca( &A[ 0 ], &A[ i ] );  /* DeleteMax */
        heapify( A, 0, i );
    }
}
```

$O(Nlog(N))$

$O(Nlog(N))$

Mergesort

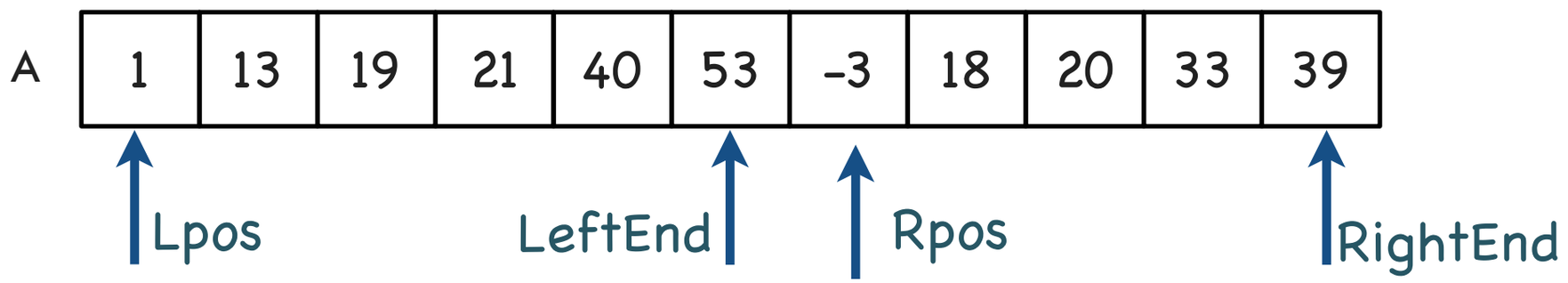
- Merge significa “unir”.
- Ideia geral:
  - Fusão de duas listas ordenadas.

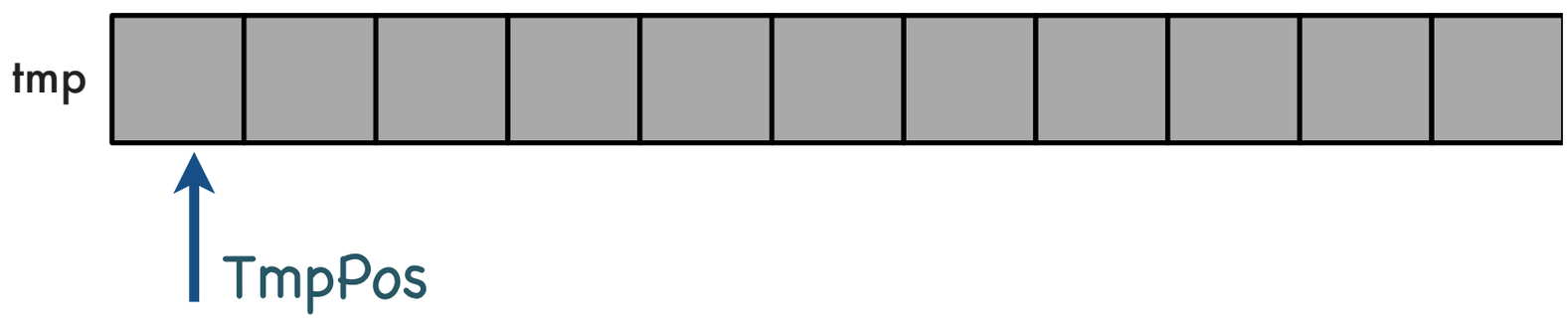


$O(Size(A) + Size(B))$

Mergesort

- Suponhamos que ambas as listas estão no mesmo array, e são contíguas:



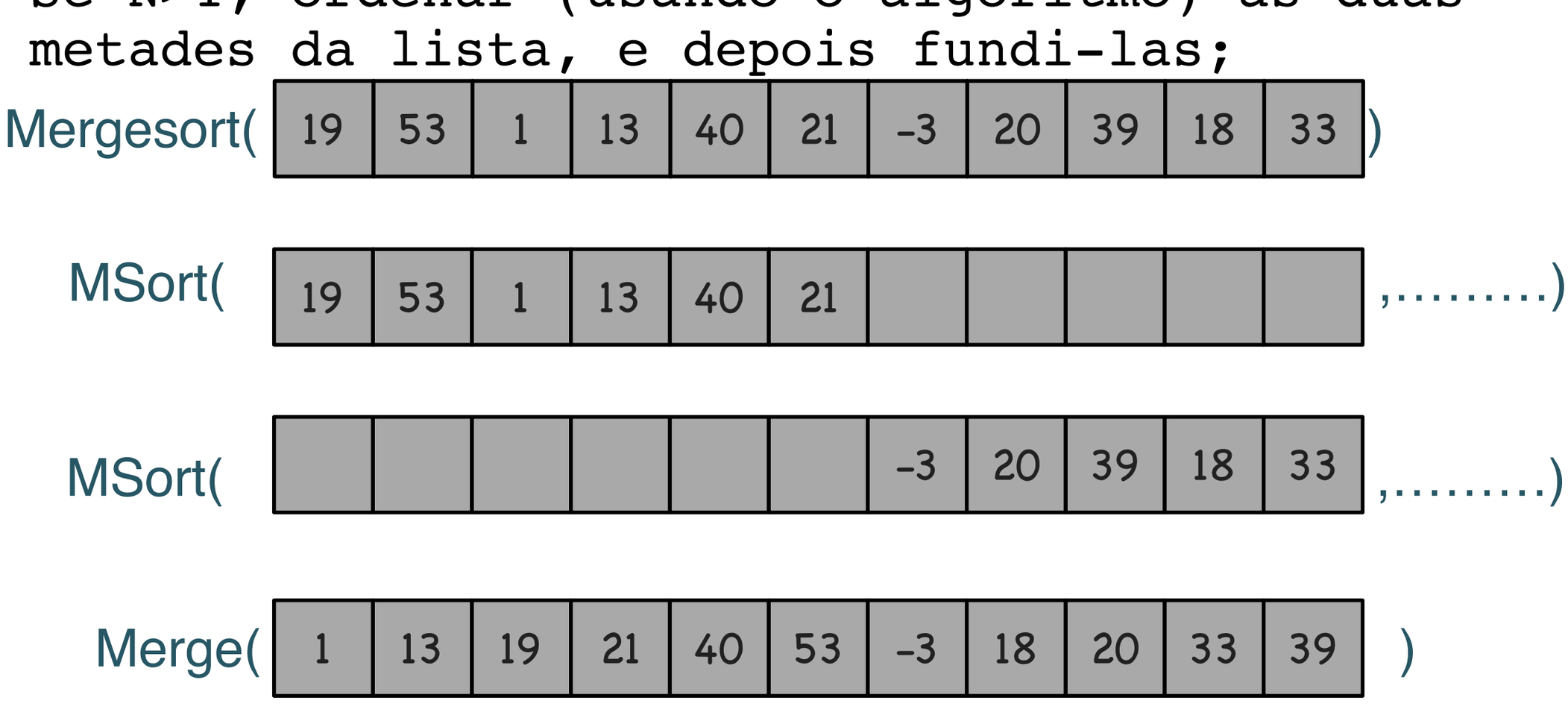


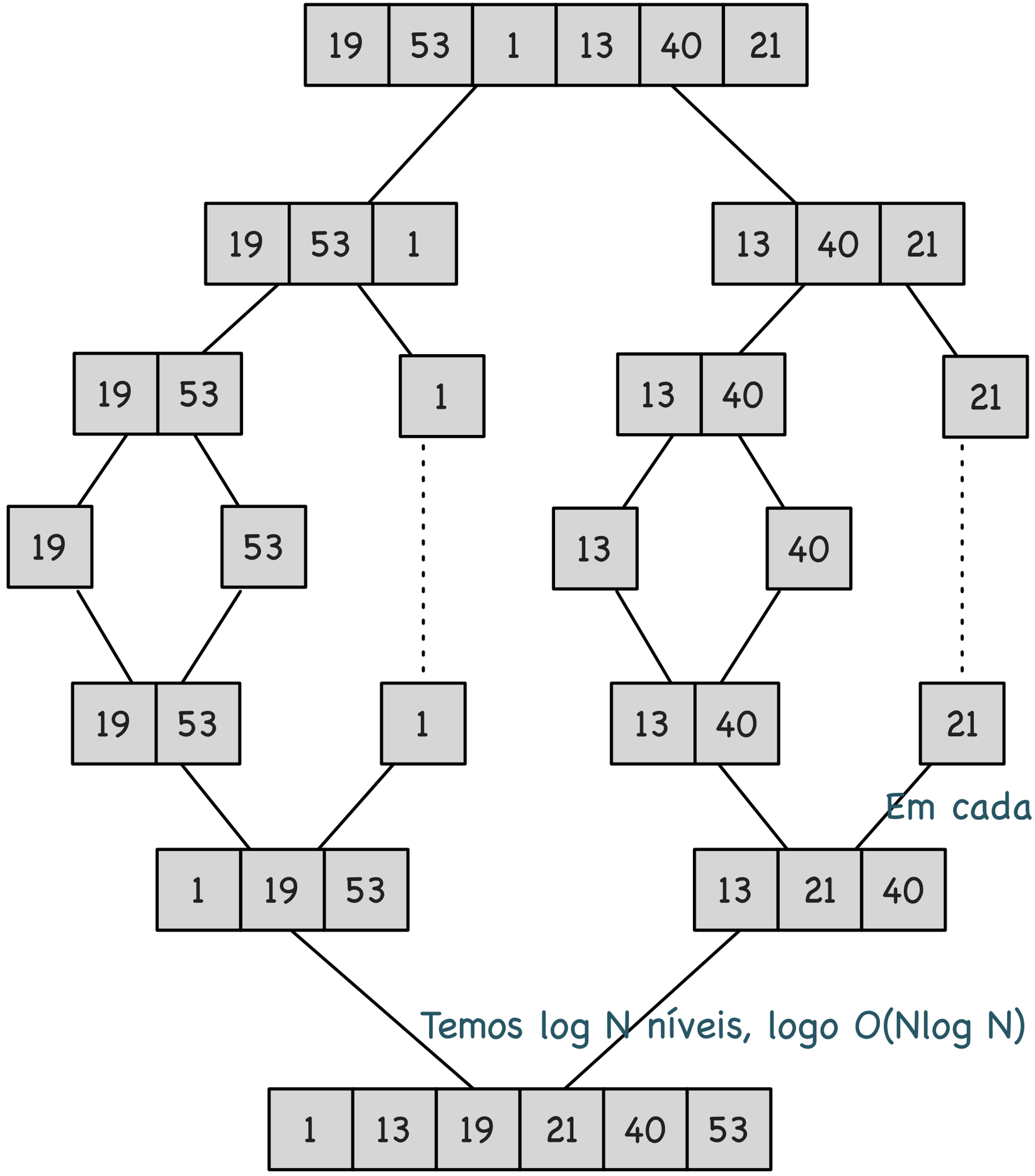
Código Fusão

```
void Merge( ElementType A[ ], ElementType TmpArray[ ],int Lpos, int Rpos,
int RightEnd ){
    int i, LeftEnd, NumElements, TmpPos;

    LeftEnd = Rpos - 1;
    TmpPos = Lpos;
    NumElements = RightEnd - Lpos + 1;

    /* main loop */
    while( Lpos <= LeftEnd && Rpos <= RightEnd )
        if( A[ Lpos ] <= A[ Rpos ] )
            TmpArray[ TmpPos++ ] = A[ Lpos++ ];
        else
            TmpArray[ TmpPos++ ] = A[ Rpos++ ];

    while( Lpos <= LeftEnd )  /* Copy rest of first half */
          TmpArray[ TmpPos++ ] = A[ Lpos++ ];
    while( Rpos <= RightEnd ) /* Copy rest of second half */
          TmpArray[ TmpPos++ ] = A[ Rpos++ ];

    /* Copy TmpArray back */
    for( i = 0; i < NumElements; i++, RightEnd-- )
        A[ RightEnd ] = TmpArray[ RightEnd ];
}
```

Mergesort

- Algoritmo recursivo:
  - Se N=1 já está (lista com 1 elemento está ordenada);
  - Se N>1, ordenar (usando o algoritmo) as duas metades da lista, e depois fundi-las;




19 53 1 13 40 21
19 53 1
13 40 21
19 53
1
13 40
21
19
53
13
40
19 53
1
13 40
21
1 19 53
13 21 40
1 13 19 21 40 53
Em cada nível O(N)
Temos log N níveis, logo O(Nlog N)

Código

```
void MSort( ElementType A[ ], ElementType TmpArray[ ],int Left,
            int Right ){
     int Center;

     if( Left < Right ){
        Center = ( Left + Right ) / 2;
        MSort( A, TmpArray, Left, Center );
        MSort( A, TmpArray, Center + 1, Right );
        Merge( A, TmpArray, Left, Center + 1, Right );
      }
}
```

```
 void Mergesort( ElementType A[ ], int N ){
     ElementType *TmpArray;

     TmpArray = malloc( N * sizeof( ElementType ) );
     if( TmpArray != NULL ){
         MSort( A, TmpArray, 0, N - 1 );
         free( TmpArray );
     }
     else
        FatalError( "No space for tmp array!!!" );
}
```

Exercício

1. Considere o array:

1	34	7	21	15	18	2	40	31	19	11

Ordene-o usando os métodos:

A. insertion sort

B. heapsort

C. merge sort

Quicksort

- http://www.youtube.com/watch?v=cNB5JCG3vts&feature=related
- http://www.youtube.com/watch?v=vxENKlcs2Tw