Diagrama de execução do MergeSort:

Entrada: Vetor A[5] = [5,4,3,2,1]

MergeSort(A, 0 ,5)

P=0 R=5

P<R-1

Q= (0+5)/2 = 2

MergeSort(A, 0, 2) [5,4]

P=0 R=2

P<R-1

Q= (0+2)/2 = 1

MergeSort(A, 0, 1) [5]

P=0 R=1

P<R-1

MergeSort(A, 1, 2) [4]

P=1 R=2

P<R-1

Intercala(A, 0, 1, 2) [4,5]

MergeSort(A, 2, 5) [3,2,1]

P=2 R=5

P<R-1

Q = (2+5)/2 = 3

MergeSort(A, 2, 3)[3]

P=2 R=3

P<R-1

MergeSort(A, 3, 5)[2,1]

P=3 R=5

P<R-1

Q=(3+5)/2 = 4

MergeSort(A, 3, 4)[2]

P=3 R=4

P<R-1

MergeSort(A, 4, 5)[1]

P=4 R=5

P<R-1

Intercala(A, 3, 4, 5)[1,2]

Intercala(A, 2, 3, 5)[1,2,3]

Intercala(A, 0, 2, 5) [1,2,3,4,5]

Diagrama de execução do QuickSort:

Entrada: Vetor A[5] = [3,7,2,15,1]

```
        partition(A, p, r)
        3       1       2       15      7
        PIVO            j       i
        troca(A[p], A[j])
        2       1       3       15      7
        retorna 2
Q = 2
```

QuickSort(A, 0 ,4)

P=0 R=4
P<R
Q= Partition(A, 0, 4) = 2
QuickSort(A, 0, 1)

P= 0 R=1
P<R
Q=Partition(A, 0, 1) = 1 [1,2]
QuickSort(A, 0, 0)

P=0 R=0
P<R

QuickSort(A, 2, 1)

P=2 R=1
P<R

[1,2,3,15,7]
QuickSort(A, 3, 4)

P=3 R=4
P<R
Q=Partition(A, 3, 4) = 4 [7,15]
QuickSort(A, 3, 3)

P=3 R=3
P<R

QuickSort(A, 5, 4)

P=5 R=4
P<R

[1,2,3,7,15]

```
pivo = A[p]
i = p+1
j = r
RODA ENQUANTO i <=j
        se ( A[i] <= pivo) i++
                senão
                        se (A[j] > pivo) j--
                        senão troca(A[i], A[j]) i++ j--
TROCA (A[p], A[j])
retorna j
------------------------------------------------------------
(A, 0, 4)
pivo = 3
i = 1
j = 4

        3       7       2       15      1
senão troca(A[i], A[j]) i++ j--
i = 2
j = 3
        3       1       2       15      7
se ( A[i] <= pivo) i++
i = 3
j = 3
        3       1       2       15      7
se (A[j] > pivo) j--
i = 3
j = 2
PARA POIS i > j
TROCA (A[p], A[j])
        2       1       3       15      7
retorna j = 2
```

```
pivo = A[p]
i = p+1
j = r
RODA ENQUANTO i <=j
        se ( A[i] <= pivo) i++
                senão
                        se (A[j] > pivo) j--
                        senão troca(A[i], A[j]) i++ j--
TROCA (A[p], A[j])
retorna j
------------------------------------------------------------
(A, 3, 4)
pivo = 15
i = 4
j = 4

        1       2       3       15      7
se ( A[i] <= pivo) i++
i = 5
j = 4
        1       2       3       15      7
PARA i > j
TROCA (A[p], A[j])
        1       2       3       7       15
retorna j = 4
```

```
pivo = A[p]
i = p+1
j = r
RODA ENQUANTO i <=j
        se ( A[i] <= pivo) i++
                senão
                        se (A[j] > pivo) j--
                        senão troca(A[i], A[j]) i++ j--
TROCA (A[p], A[j])
retorna j
------------------------------------------------------------
(A, 0, 1)
pivo = 2
i = 1
j = 1

        2       1
se ( A[i] <= pivo) i++
PARA POIS i > j
TROCA (A[p], A[j])
        1       2
retorna j = 1
```

HeapSort

Um heap é uma **lista linear** composta de elementos com chaves $s_1, \ldots, s_n$, considere que o lista linear possui **M** posições disponíveis e satisfaz a **seguinte propriedade**:

$$\text{propriedade(i)} = s_i \leq s_{\lfloor i/2 \rfloor} \text{ para } 1 < i \leq n.$$

Algoritmo arranjar(n)
início
  para $i \leftarrow \lfloor n/2 \rfloor$ **até** 1 **faça**
    descer(i, n)
  fim-para
fim.

Algoritmo descer(i, n)
início
  j$\leftarrow 2 \times i$
  se j$\leq n$ **então**
    se j$< n$ **então** # tem filho a direita
      se T[j + 1] > T[j] **então** # filho direito
        j$\leftarrow$j + 1
      fim-se
    fim-se
    se T$[i]$ < T[j] **então**
      troca(T$[i]$,T[j])
      descer(j, n)
    fim-se
  fim-se
fim.

****Filhos precisam ser <= do que seu pai!**

Vetor = [4, 12, 1, 7, 20, 14, 5, 4]

n=8

1 -> 8

Arranjar(8)

n=8 i=4

Descer[4,8]

J=8
J<=n
[4, 12, 1, 7, 20, 14, 5, 4]

i=i-1 i=3

Descer(3,8)

J=6
J<=n
[4, 12, 14, 7, 20, 1, 5, 4]

Descer(6, 8)

j=12
J<=n

i=i-1 i=2

Descer(2,8)

J=4
J<=n
J=J+1
J=5
[4, 20, 14, 7, 12, 1, 5, 4]

Descer(5, 8)

j=10
J<=n

i=i-1 i=1

Descer(1, 8)

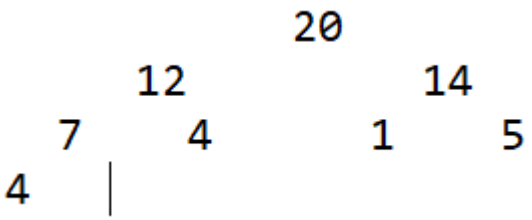
J=2
J<=n
[20, 4, 14, 7, 12, 1, 5, 4]

Descer(2,8)

J=4
J<=n
J=J+1 J=5
[20, 12, 14, 7, 4, 1, 5, 4]
Descer(5,8)

J=10
J<=n

[20, 12, 14, 7, 4, 1, 5, 4]



Inserindo um valor no final do vetor

```
Algoritmo insere(novo)
início
  se n < M então
    T[n+1] ← novo
    n ← n + 1
    subir(n)
  senão
    overflow
  fim-se
fim.
```

```
Algoritmo subir(i)
início
  j ← ⌊i/2⌋
  se j ≥ 1 então
    se T[i] > T[j] então
      troca(T[i],T[j])
      subir(j)
    fim-se
  fim-se
fim.
```

[20, 12, 14, 7, 4, 1, 5, 4]

n=8

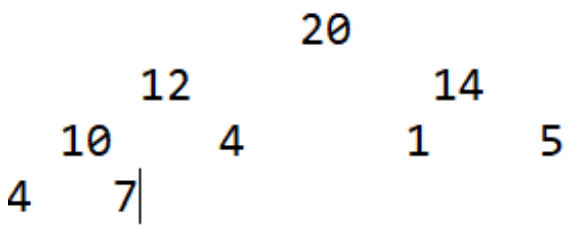
Insere(10)

T[n+1] = 10
[20, 12, 14, 7, 4, 1, 5, 4, 10]
n++ n=9
Subir(9)

i=9
J=4
J>=1
T[i]>T[J]
[20, 12, 14, 10, 4, 1, 5, 4, 7]
Subir(4)

i=4
j=2
J>=1
T[i]>T[J]

[20, 12, 14, 10, 4, 1, 5, 4, 7]



Remoção de um elemento do vetor

```
Algoritmo remove()
início
  se n ≠ 0 então
    agir(T[1])
    T[1] ← T[n]
    n ← n − 1
    descer(1, n)
  senão
    underflow
  fim-se
fim.
```

```
Algoritmo descer(i, n)
início
  j ← 2 × i
  se j ≤ n então
    se j < n então # tem filho a direita
      se T[j + 1] > T[j] então # filho direito
        j ← j + 1
      fim-se
    fim-se
    se T[i] < T[j] então
      troca(T[i],T[j])
      descer(j, n)
    fim-se
  fim-se
fim.
```

[20, 12, 14, 10, 4, 1, 5, 4, 7]

n=9

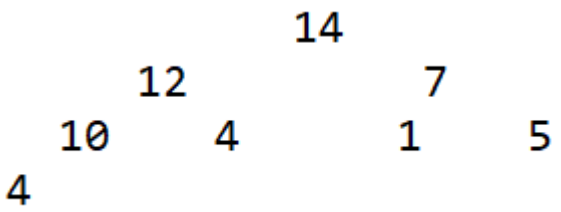
Remove()

n!=0
agir(20)
T[1] = 7
n=n-1 n=8
[7, 12, 14, 10, 4, 1, 5, 4]
Descer(1,8)

i=1
J=2
J<=n
J<n
T[3]>T[2]
J++ J=3
T[1]<T[3]
[14, 12, 7, 10, 4, 1, 5, 4]
Descer(3,8)

i=3
J=6
J<=n
J<n
T[3]>T[2]
J++ J=7
T[3]<T[7]

[14, 12, 7, 10, 4, 1, 5, 4]



Ordenação por Heap de máxima

[14, 12, 7, 10, 4, 1, 5, 4]

n=8

```
Algoritmo Heasort(n)
início
   arranjar(n)
   k ← n
   enquanto k > 1 faça
      troca(T[1],T[k])
      k ← k − 1
      descer(1, k)
    fim-enquanto
fim.
```

```
Algoritmo descer(i, n)
início
  j ← 2 × i
  se j ≤ n então
     se j < n então # tem filho a direita
        se T[j + 1] > T[j] então  # filho direito
           j ← j + 1
        fim-se
     fim-se
     se T[i] < T[j] então
        troca(T[i],T[j])
        descer(j, n)
     fim-se
  fim-se
fim.
```

HeapSort(8)

Arranjar já está feito= [14, 12, 7, 10, 4, 1, 5, 4]
k = 8
8>1
Troca(14, 4) [4, 12, 7, 10, 4, 1, 5, 14]
k=k-1 k=7
Descer(1,7)

[4, 12, 7, 10, 4, 1, 5]
i=1
n=7
J=2
J<= 7
J< 7
T[3]>T[2]
T[1]<T[2]
Troca(4,12) [12, 4, 7, 10, 4, 1, 5]
Descer(2, 7)

[12, 4, 7, 10, 4, 1, 5]
i=2
n=7
J=4
J<= 7
J< 7
T[5]>T[4]
T[2]<T[4]
Troca(4,10) [12, 10, 7, 4, 4, 1, 5]
Descer(4, 7)

[12, 10, 7, 4, 4, 1, 5]
i=4
n=7
J=8
J<= 7

7>1

Troca(12, 5) [5, 10, 7, 4, 4, 1, 12, 14]

k=k-1 k=6

Descer(1,6)

[5, 10, 7, 4, 4, 1]

i=1

j=2

n=6

J<= 6

J< 6

T[3]>T[2]

T[1]<T[2]

Troca(5,10) [10, 5, 7, 4, 4, 1]

Descer(2, 6)

[10, 5, 7, 4, 4, 1]

i=2

j=4

n=6

J<= 6

J< 6

T[5]>T[4]

T[2]<T[4]

6>1

Troca(10, 1) [1, 5, 7, 4, 4, 10, 12, 14]

k=k-1 k=5

Descer(1,5)

[1, 5, 7, 4, 4]

i=1

j=2

n=5

J<= 5

J< 5

T[3]>T[2]

j=j+1 j=3

T[1]<T[3]

Troca(1,7) [7, 5, 1, 4, 4]

Descer(3, 5)

i=3

j=6

n=5

J<= 5

5>1

Troca(7, 4) [4, 5, 1, 4, 7, 10, 12, 14]

k=k-1 k=4

Descer(1,4)

[4, 5, 1, 4]
i=1
j=2
n=4
J<= 4
J< 4
T[3]>T[2]
T[1]<T[2]

Troca(4,5) [5, 4, 1, 4]

Descer(2, 4)

[5, 4, 1, 4]
i=2
j=4
n=4
J<= 4
J< 4
T[2]<T[4] #Estabilidade Verificada

4>1

Troca(5, 4) [4, 4, 1, 5, 7, 10, 12, 14]

k=k-1 k=3

Descer(1,3)

[4, 4, 1]
i=1
j=2
n=3
J<= 3
J< 3
T[3]>T[2]
T[1]<T[2]

3>1

Troca(4, 1) [1, 4, 4, 5, 7, 10, 12, 14]

k=k-1 k=2

Descer(1,2)

[1, 4]
i=1
j=2
n=2
J<= 2
J< 2
T[1]<T[2]
Troca(1,4) [4, 1]
Descer(2, 2)

[4, 1]
i=2
j=4
n=2
J<= 2

2>1

Troca(4, 1) [1, 4, 4, 5, 7, 10, 12, 14]

k=k-1 k=1

Descer(1,1)

[1]
i=1
j=2
n=1
J<= 2

1>1

Return [1, 4, 4, 5, 7, 10, 12, 14]

Lista de prioridades:

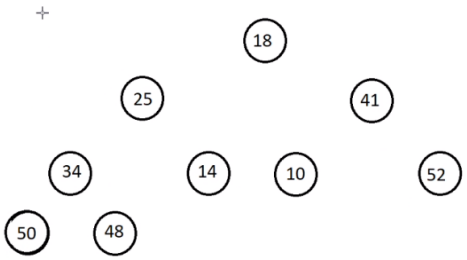
- **Lista não ordenada**
  - **Seleção O(n): Busca o maior valor**
  - **Inserção O(1): Coloca em qualquer lugar**
  - **Remoção O(n): Busca o maior elemento e remove**
  - **Construção O(n): Insere o número de elementos**
- **Lista ordenada**
  - **Seleção O(1): Já sabemos onde está o maior elemento.**
  - **Inserção O(n):Busca a posição adequada para aquele valor**
  - **Remoção O(1): Já sabemos a localização do maior elemento**
  - **Construção O(n lg n): MergeSort para deixar ela ordenada**
- **Heap.**
  - **Seleção O(1): Selecionamos a raiz, que é o maior elemento**
  - **Inserção O(lg n): Inserimos ele como último elemento e então usamos a função SUBIR()**
  - **Remoção O(lg n): Troca de lugar com a raiz e usa a função Descer()**
  - **Construção O(n): Quando é realizada a função descer dentro de arranjar, todos os elementos do vetor são percorridos.**

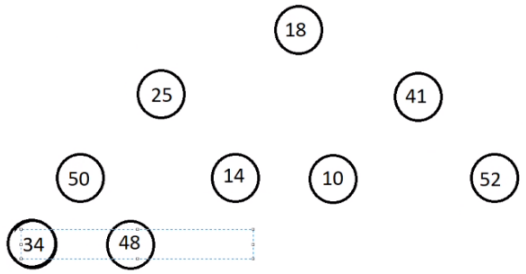
Praticando:

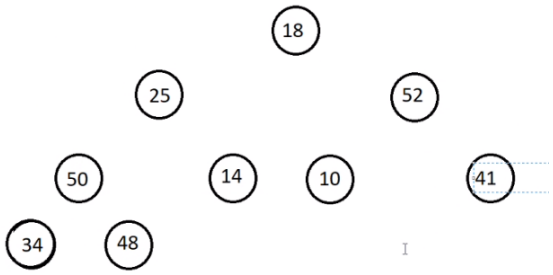
4) Seja uma lista dada pelas prioridades a seguir:

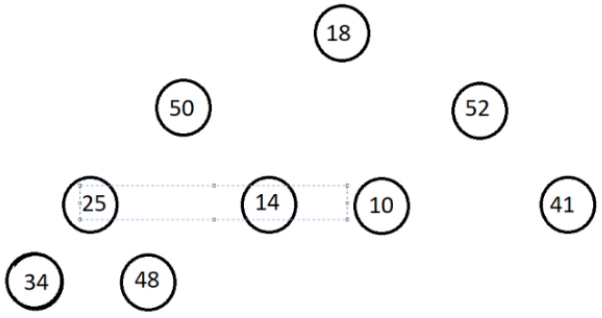
`18 25 41 34 14 10 52 50 48`

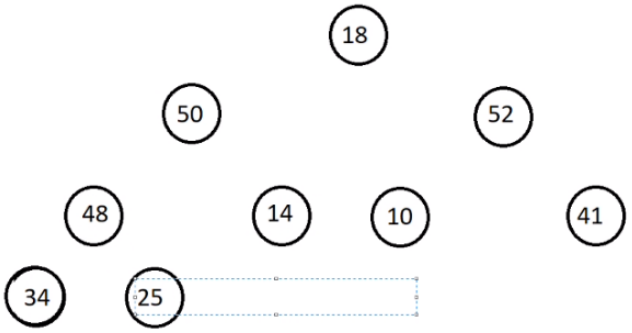
Determinar o heap obtido pela aplicação do algoritmo de construção $\text{arranjar}(n)$.

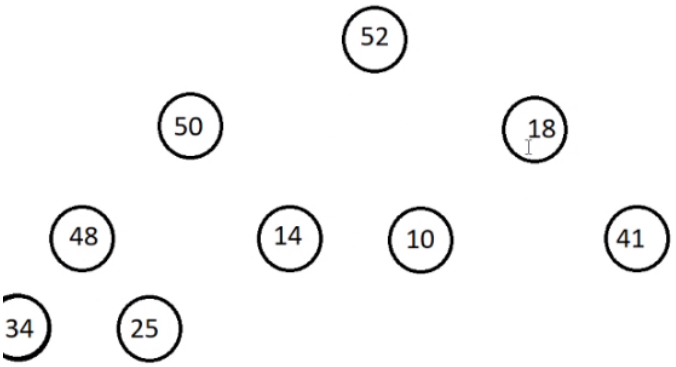


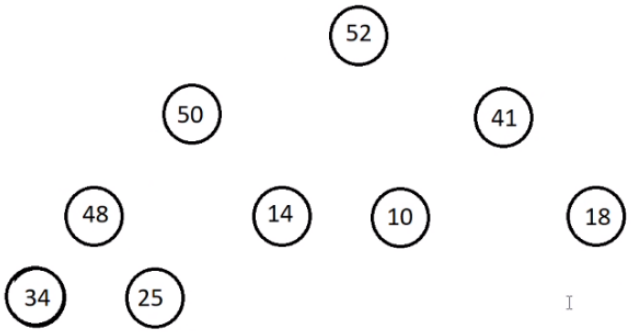












Escreva uma função na linguagem C que verifica se um vetor T[1.. n] é ou não um heap.

Um heap é uma **lista linear** composta de elementos com chaves s_1, ..., s_n, considere que o lista linear possui M posições disponíveis e satisfaz a **seguinte propriedade**:

$$\text{propriedade(i)} = s_i \leq s_{\lfloor i/2 \rfloor} \text{ para } 1 < i \leq n.$$

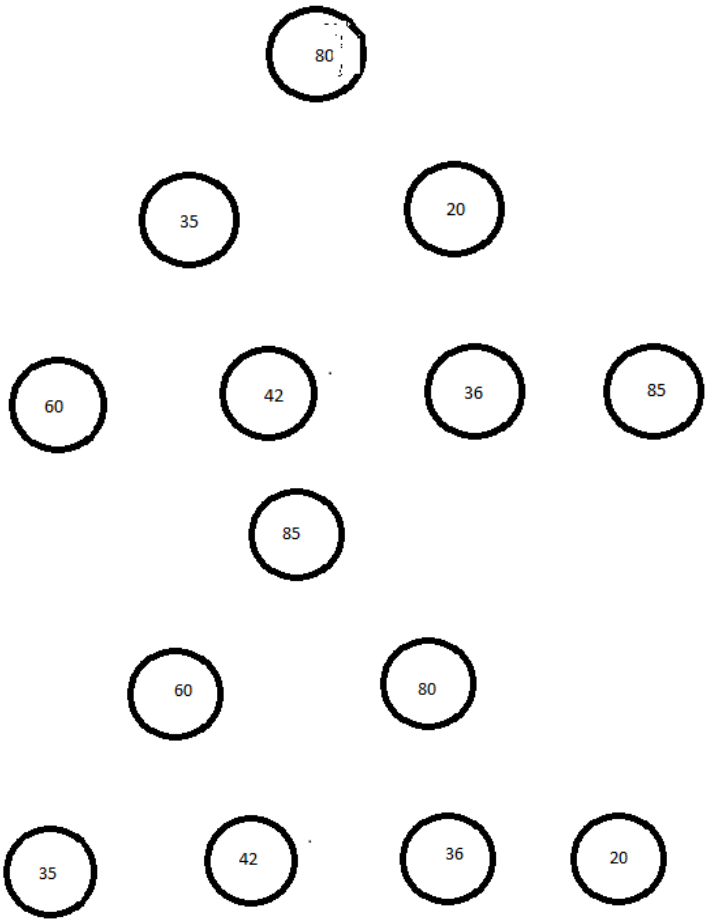
[52, 50, 41, 48, 14, 10, 18, 34, 25]

```
int verifica(int A[], int n){
        For(int i=1; i<=n/2; i++){
                if (A[i] >= A[i*2] ){
                        continue;
                }
                else{
                        return 0;
                }
                if( (i*2) < n ){ //verifica o filho a direita
                        if (A[i] >= A[i*2+1] ){
                                continue;
                        }
                        else{
                                return 0;
                        }
                }
        }
        return 1;
}
```

6) Mostrar o comportamento da ordenação em heap para a lista

80, 35, 20, 60, 42, 36, 85.

80, 35, 85, 60, 42, 36, 20
80, 60, 85, 35, 42, 36, 20
85, 60, 80, 35, 42, 36, 20



7) O que acontece se o algoritmo Heapsort for utilizado em um vetor já ordenado? E se vetor estiver ordenado decrescente ?

Quando o vetor já está ordenado é o pior caso para o algoritmo HeapSort enquanto a ordem decrescente é o melhor caso, pois são necessárias menos operações de trocas entre os elementos do vetor. No entanto, ambos rodam em n lg n, sendo assim assintoticamente iguais.