**PROGRAMAÇÃO ESTRUTURADA T2**

Grupo: Amarildo Junior, Leonardo, Elias Júnior.

Parte 1: Tempo de ordenação dos vetores.

Máquina utilizada nos testes:

Desktop

Processador: R7 2700 3.2GHz 8 cores 16 threads

Ram: 8gb | SSD: 240GB SATA |HD: 2Tb | Placa de Vídeo: RX570 4GB

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 |  | 10 | 1 | 9 |  | 4 | 2 |  | 5 | 7 | 3 |

SO: Windows 10 Pro 64 bit’s

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Bubble | Selection | Insertion | Quick | Merge | Heap |
| Vetor A | 0.087 | 0.032 | 0.017 | 0.000 | 0.000 | 0.001 |
| Vetor B | 0.359 | 0.122 | 0.068 | 0.002 | 0.001 | 0.002 |
| Vetor C | 1.512 | 0.490 | 0.273 | 0.002 | 0.003 | 0.004 |
| Vetor D | 4.277 | 1.096 | 1.162 | 0.003 | 0.006 | 0.004 |

Parte 2: Explicação da ordenação utilizando Mergesort e Heapsort de maneira **não-crescente**.

Vetor a ser ordenado:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 | 10 | 1 | 9 | 4 | 2 | 5 | 7 | 3 |

**Mergesort:**

O ***mergesort***, ou ordenação por mistura, é um exemplo de algoritmo de ordenação por comparação do tipo dividir para conquistar.

Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas).

O vetor será dividido pela metade até sobrarem apenas itens individuais:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 | 10 | 1 | 9 | 4 | 2 | 5 | 7 | 3 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 | 10 | 1 | 9 |  | 4 | 2 | 5 | 7 | 3 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 |  | 2 |  | 10 |  | 1 | 9 |  | 4 |  | 2 |  | 5 |  | 7 | 3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 |  | 9 |  | 7 |  | 3 |

Com todos os itens do vetor já divididos em partes individuais eles serão colocados em vetores temporários de maneira já ordenada (ordenação feita no procedimento recursivo):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 |  | 10 |  | 9 | 1 |  | 4 | 2 |  | 5 |  | 7 | 3 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 1 |  | 7 | 5 | 3 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 6 | 2 | 1 |  | 7 | 5 | 4 | 3 | 2 |

Quando restam apenas dois vetores temporários já ordenados eles são colocados no vetor principal com a ordenação completa.

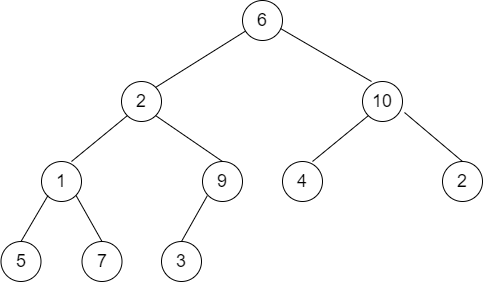
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |

**Heapsort:**

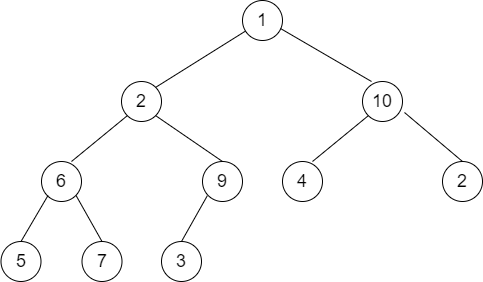
O heapsort utiliza uma estrutura de dados chamada heap, para ordenar os elementos à medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de max-heap(pai>filho).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | 2 | 10 | 1 | 9 | 4 | 2 | 5 | 7 | 3 |

Ao executar o algoritmo ele percorrerá o vetor e criará uma árvore com os elementos. Da esquerda para a direita é criada uma árvore com os valores encontrados no vetor. A árvore ficará com a seguinte organização:



Após a estruturação da árvore será encontrado o maior valor do vetor, este valor será o max\_heap, nesse caso(ordenação decrescente, o valor a ser posto no topo é o menor) o valor 1 será trocado com o valor 6.

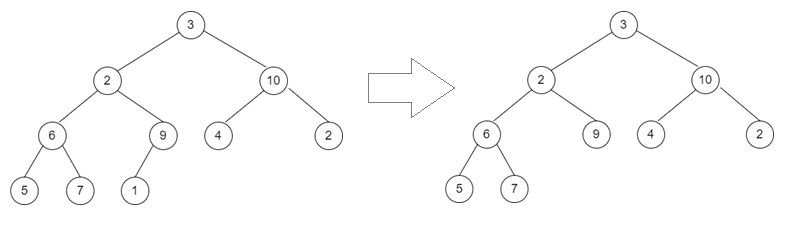
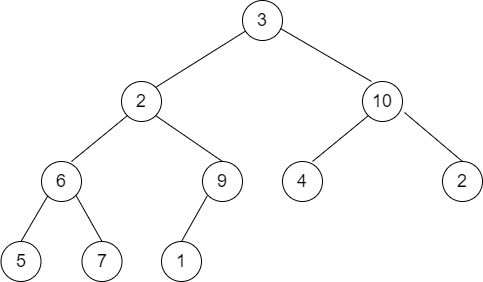


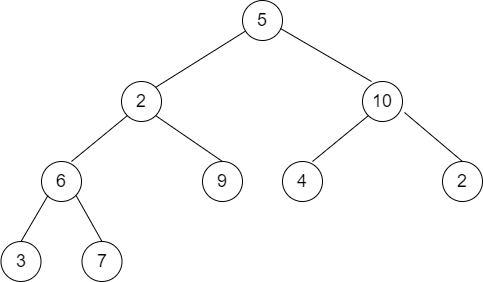
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 10 | 6 | 9 | 4 | 2 | 5 | 7 | 3 |

Encontrado o max\_heap é trocado o valor da primeira posição com o valor da última:

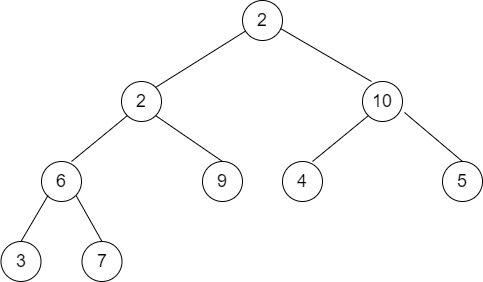
O número 1 já é considerado ordenado e está em sua posição final.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 10 | 6 | 9 | 4 | 2 | 3 | 7 | 1 |



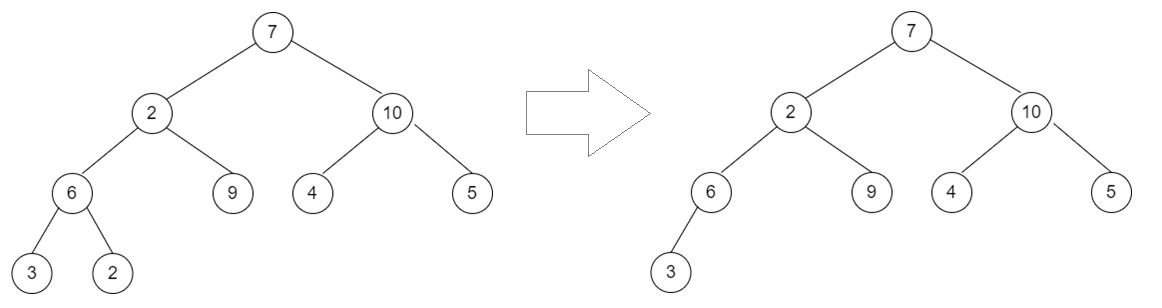
O número 3 vai para a última posição do nó da esquerda e é substituído pelo menor valor encontrado(5 nesse caso). 

O número 2 mais abaixo vai para a posição max\_heap e é substituído pelo valor max\_heap encontrado(5 nesse caso).



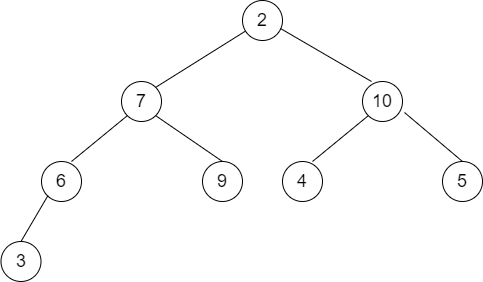
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 2 | 10 | 6 | 9 | 4 | 5 | 3 | 7 | 1 |

O max\_heap (numero 2) é substituído pela ultima posição (numero 7) e é excluído da árvore.



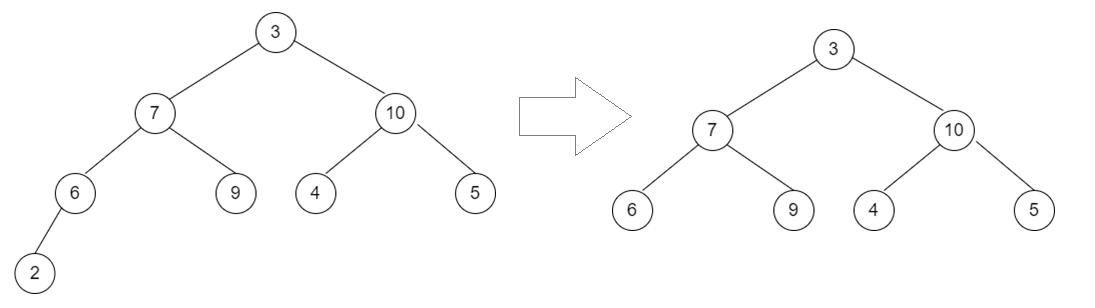
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 10 | 6 | 9 | 4 | 5 | 3 | 2 | 1 |

É verificado que o menor número não está no max\_heap, então troca o menor número (número 2) com o max\_heap (numero 7).



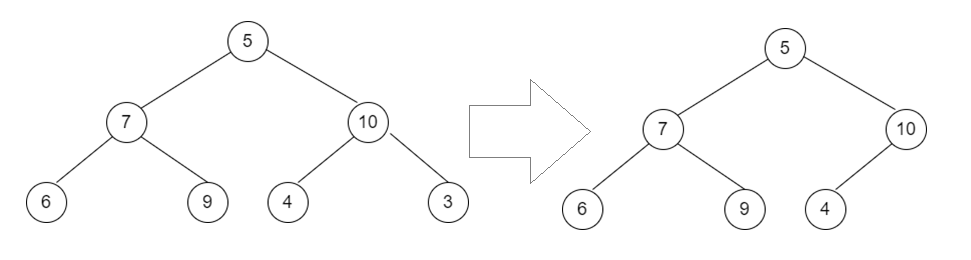
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 7 | 10 | 6 | 9 | 4 | 5 | 3 | 2 | 1 |

Verificado que o max\_heap é o menor número (número 2), então é trocado com o ultimo nó da árvore (número 3), após a troca está ordenado então é excluído da árvore.



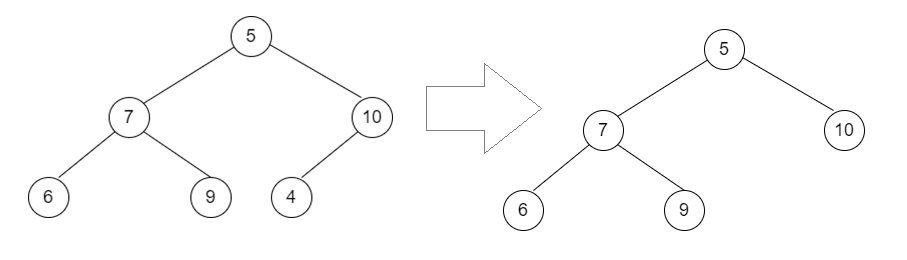
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3 | 7 | 10 | 6 | 9 | 4 | 5 | 2 | 2 | 1 |

Verificado que o max\_heap novamente é o menor número da arvore, trocasse então o 3 com 5, nó mais a direita da árvore, verificado então que o 3 está na posição ordenada é excluído.



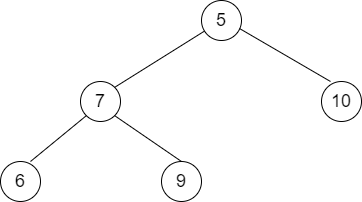
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 7 | 10 | 6 | 9 | 4 | 3 | 2 | 2 | 1 |

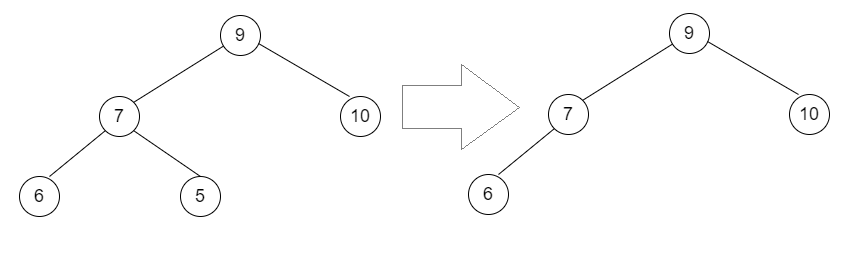
Verificado que a última posição (número 4) é a menor posição da arvore, então apenas exclui a menor posição da árvore, pois já está ordenada.



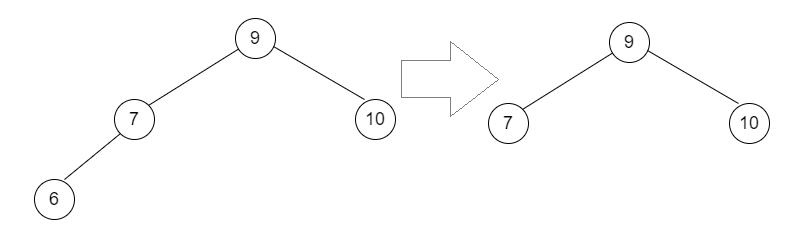
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 7 | 10 | 6 | 9 | 4 | 3 | 2 | 2 | 1 |

Verificado que o max\_heap é a menor valor da árvore, então troco o 5 com o 9 e excluo o 5, pois está ordenado.



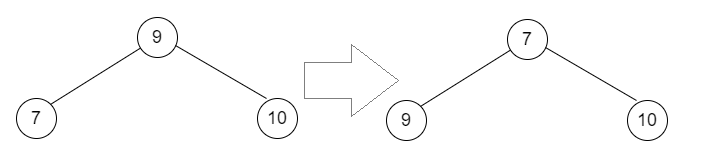


|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 7 | 10 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |

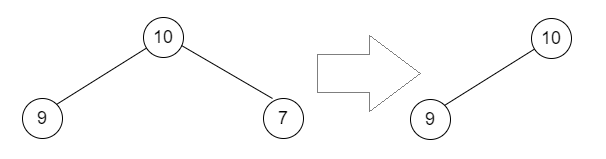
Verificado o menor número da árvore já está na extremidade da árvore, então apenas exclui o número 6 da árvore.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9 | 7 | 10 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |

Verificado que o nó da árvore mais a esquerda é o menor valor da árvore, então trocasse o menor valor com o max\_heap(número 9).

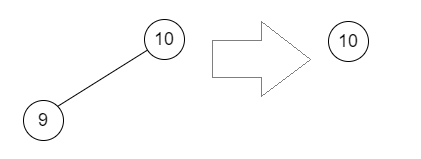


|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 9 | 10 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |

Verificado que o menor valor da árvore é o max\_heap e o nó mais a direita é o maior valor da árvore, então trocasse o número 7 com o 10 e exclui o nó mais a direita da árvore (número 7).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |

Verificado que o nó mais a esquerda é o menor valor restante na árvore, então excluísse o nó mais a esquerda da árvore e o vetor está ordenado de forma não-crescente.



|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10 | 9 | 7 | 6 | 5 | 4 | 3 | 2 | 2 | 1 |