Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática
Prof. Flávio Rogério Uber

HEAPSORT

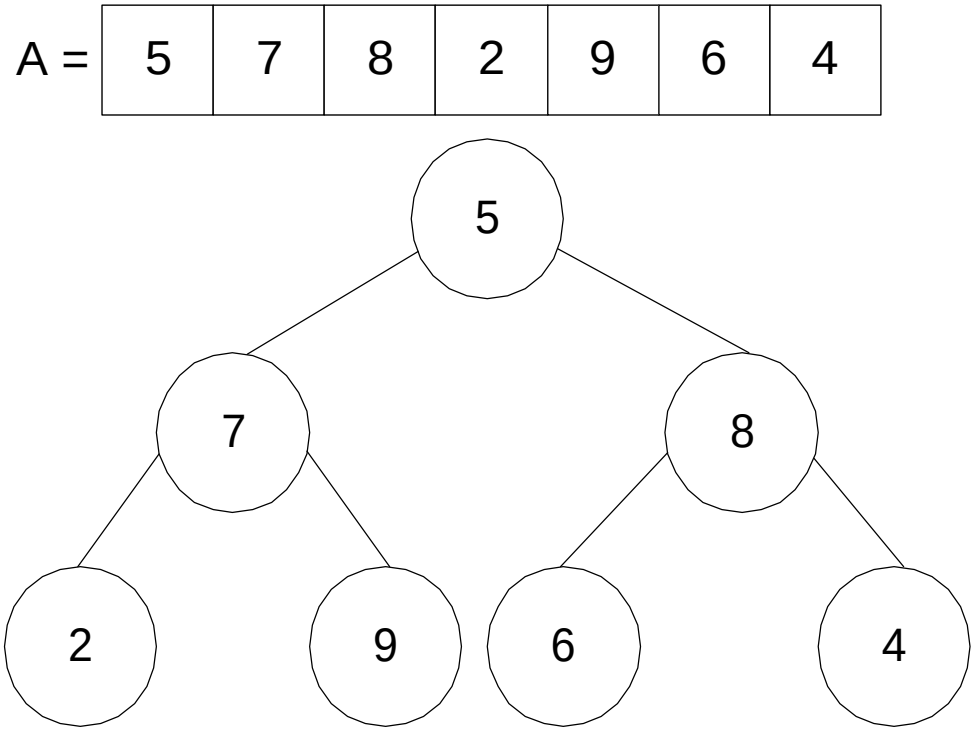
Heapsort

- Proposto por Floyd e Williams em 1964;
- O desempenho, no pior caso, é semelhante ao caso médio;
- É considerado uma variação do método de ordenação por seleção;

HeapSort

- Heap: árvore binária, organizada em um vetor, obedecendo a seguinte regra:
  - Os filhos de um nó i estão armazenados nas posições 2i e 2i + 1 do vetor
  - Todo nó tem valor maior ou igual ao de seus filhos
  - A diferença máxima de altura entre os nós folha é de um nível
- A raiz do heap é maior chave

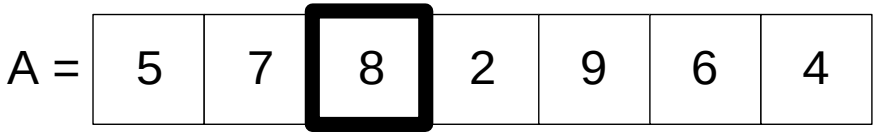
Heapsort

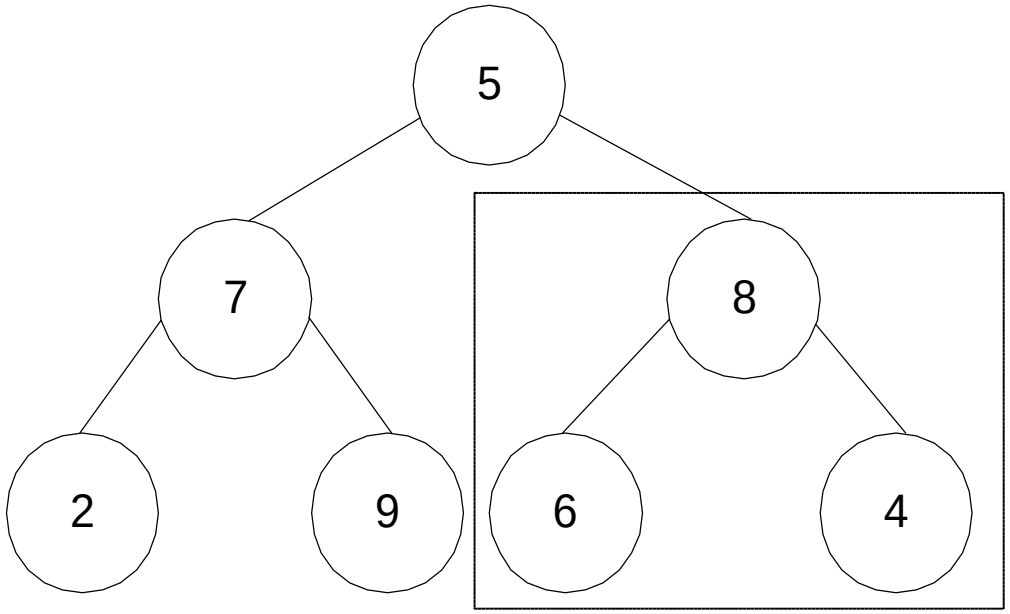


É preciso que a estrutura seja transformada em um heap, já que a restrição com relação aos valores das chaves não é atendida.

Heapsort

- A transformação começa a partir do nó que ocupa a posição n div 2 do vetor.

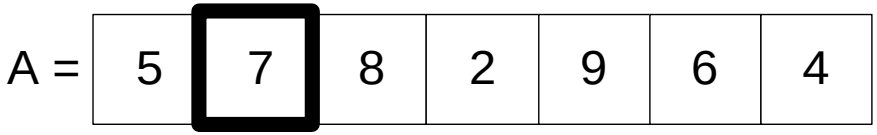




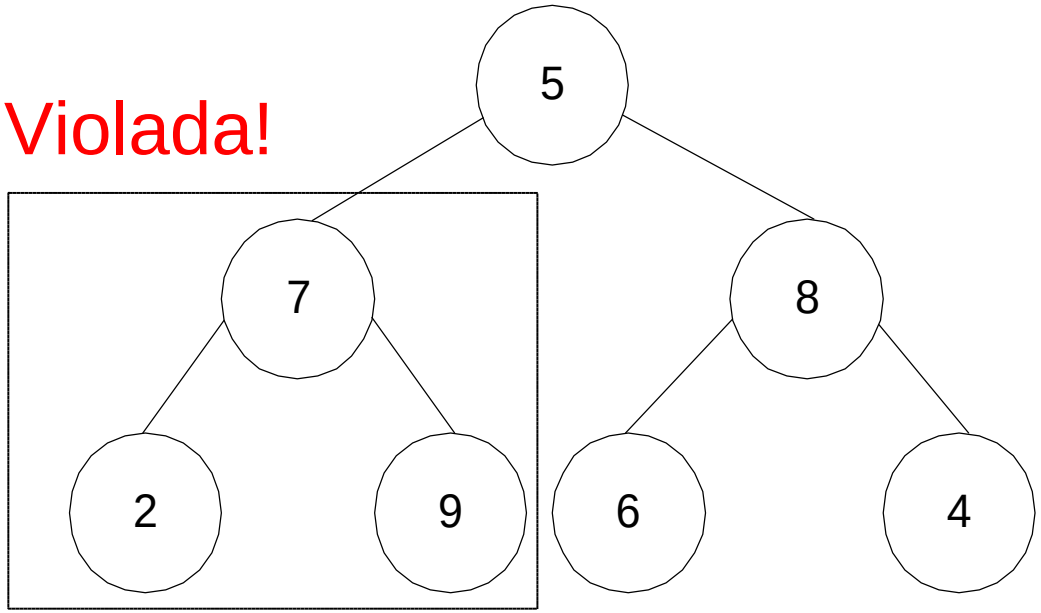
--- OK!

Heapsort

- Os próximos nós são obtidos decrementando-se os índices no vetor até este atingir o valor 1

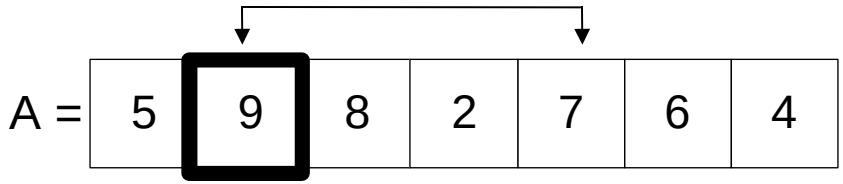


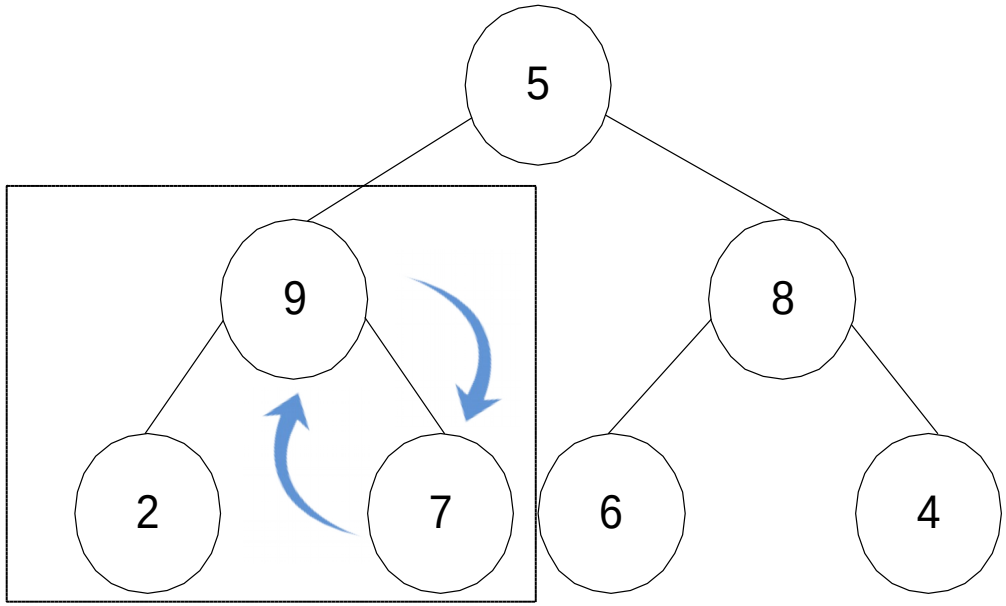
Condição Violada!



Heapsort

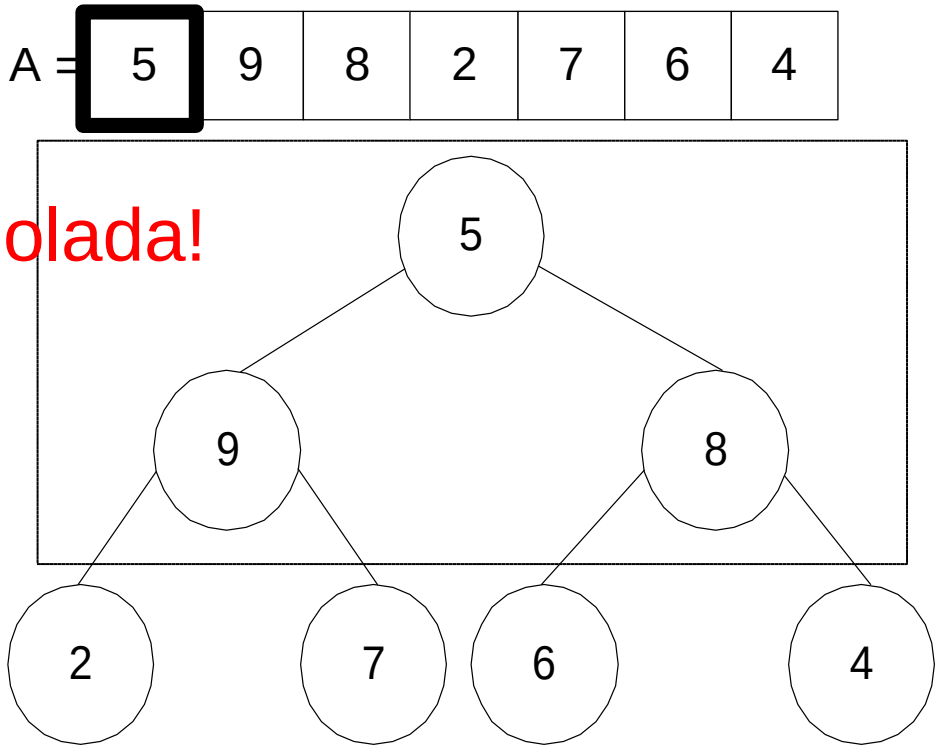
- Com a condição violada, troca-se o nó de valor 7 com o nó de maior valor entre seus filhos





Heapsort

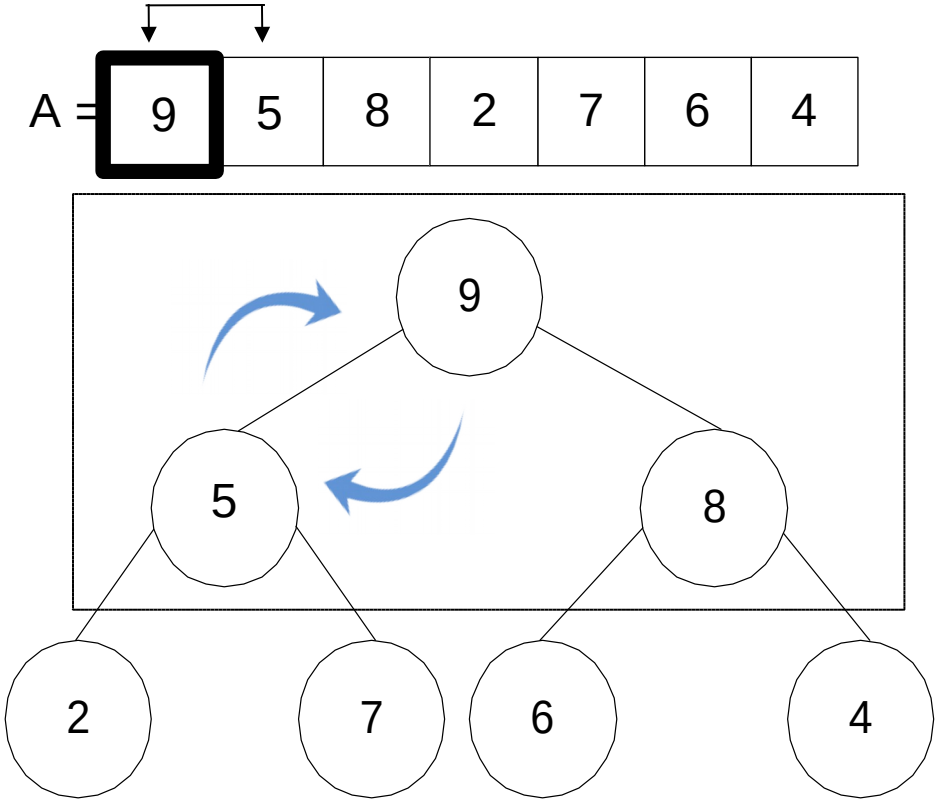
- No próximo passo, o índice 1 foi atingido



Condição Violada!

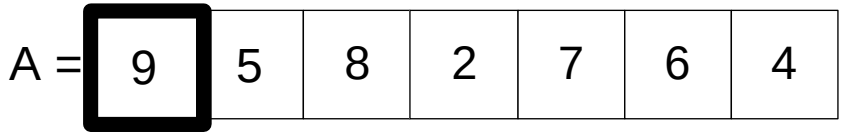
Heapsort

- Com a condição violada, troca-se o nó de valor 7 com o nó de maior valor entre seus filhos

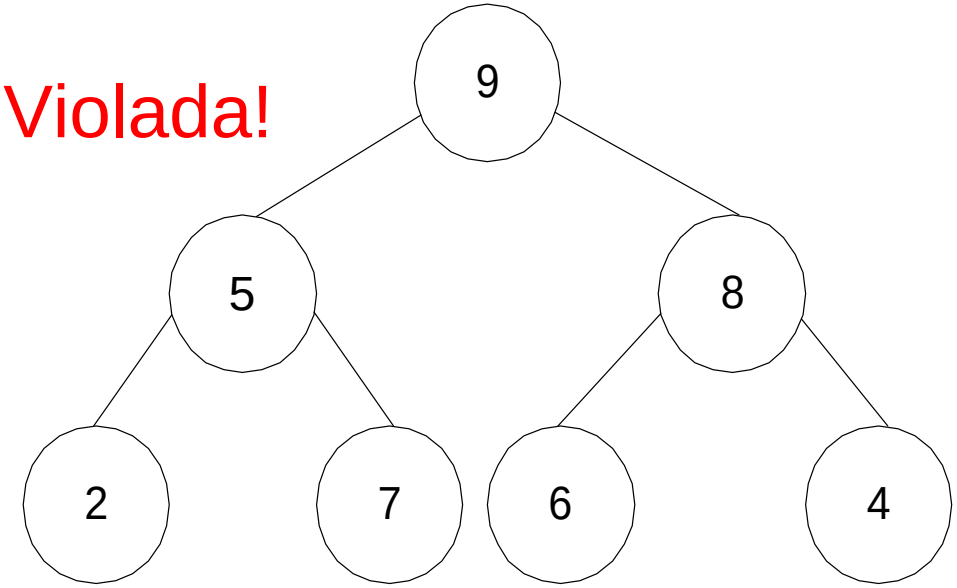


Heapsort

- No entanto, esta última troca, causou problema para o restante do heap.

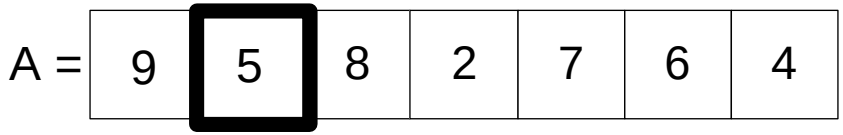


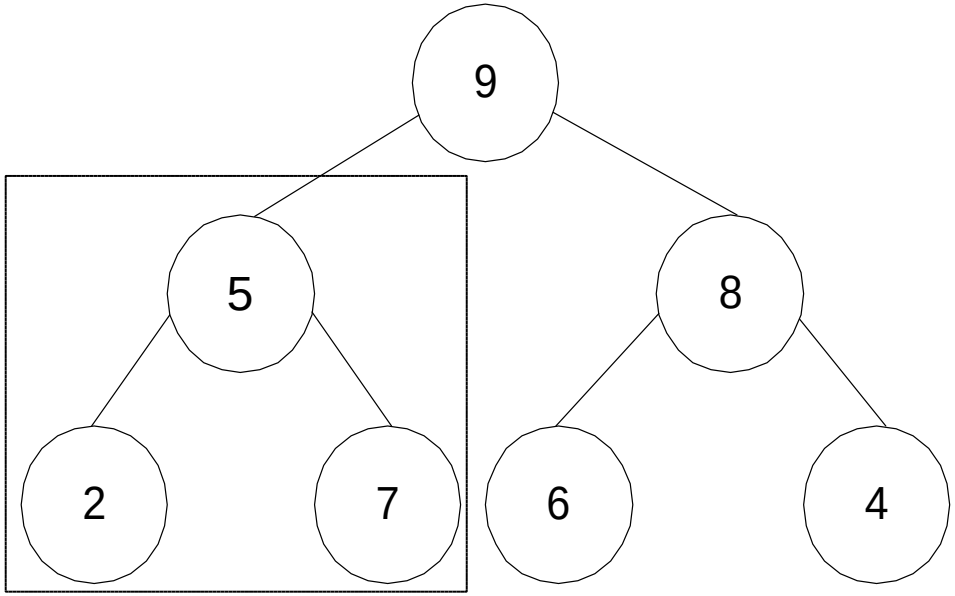
Condição Violada!



Heapsort

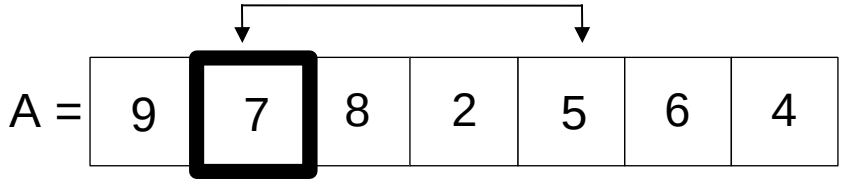
- Deve-se então percorrer o heap na ordem inversa, de 1 até n div 2

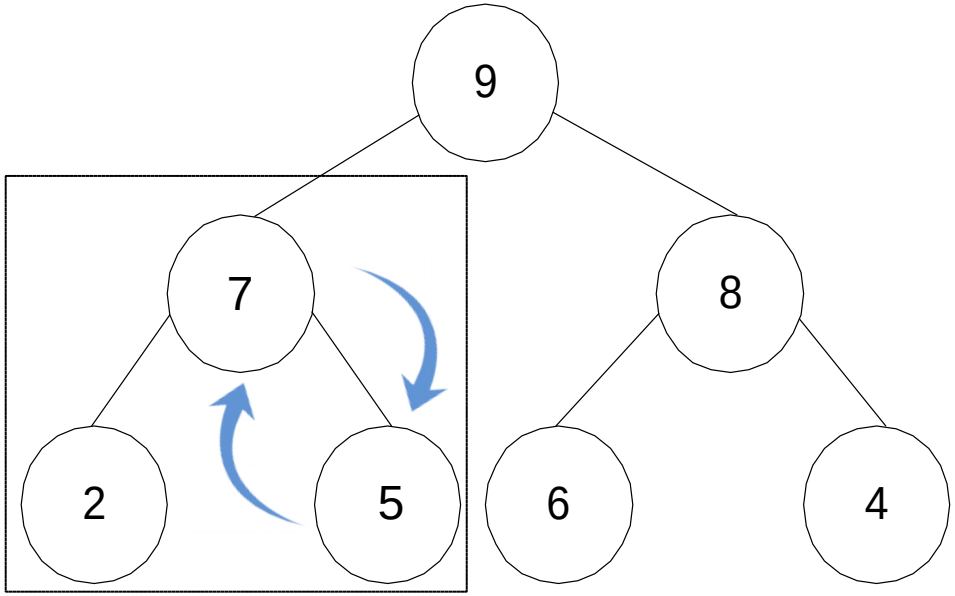




Heapsort

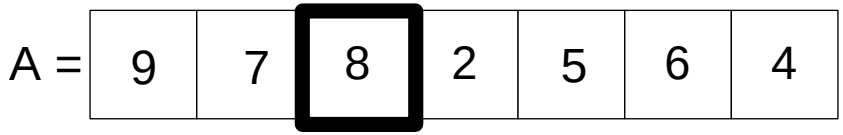
- É feita a troca dos elementos para corrigir a violação

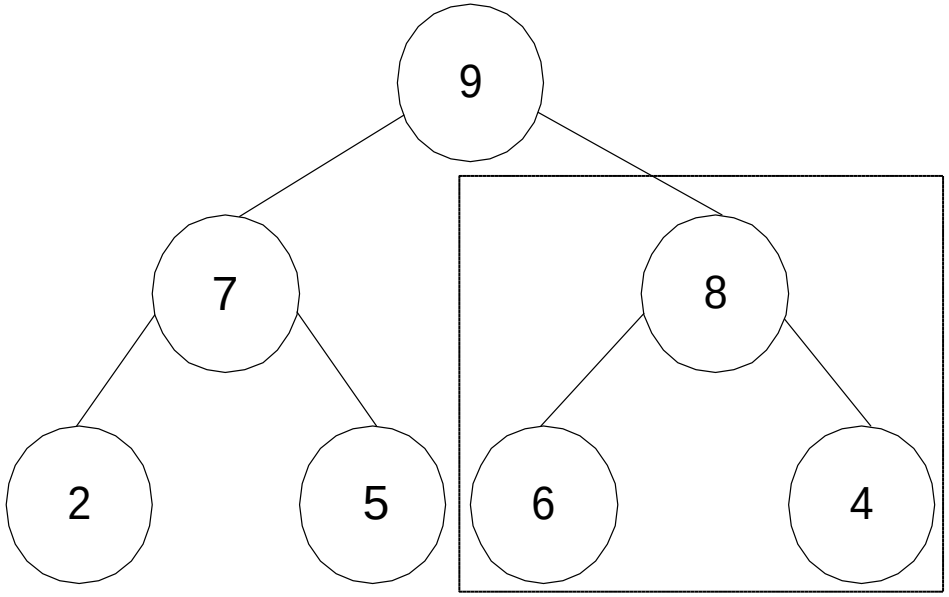




Heapsort

- Prossegue-se até atingir o índice n div 2

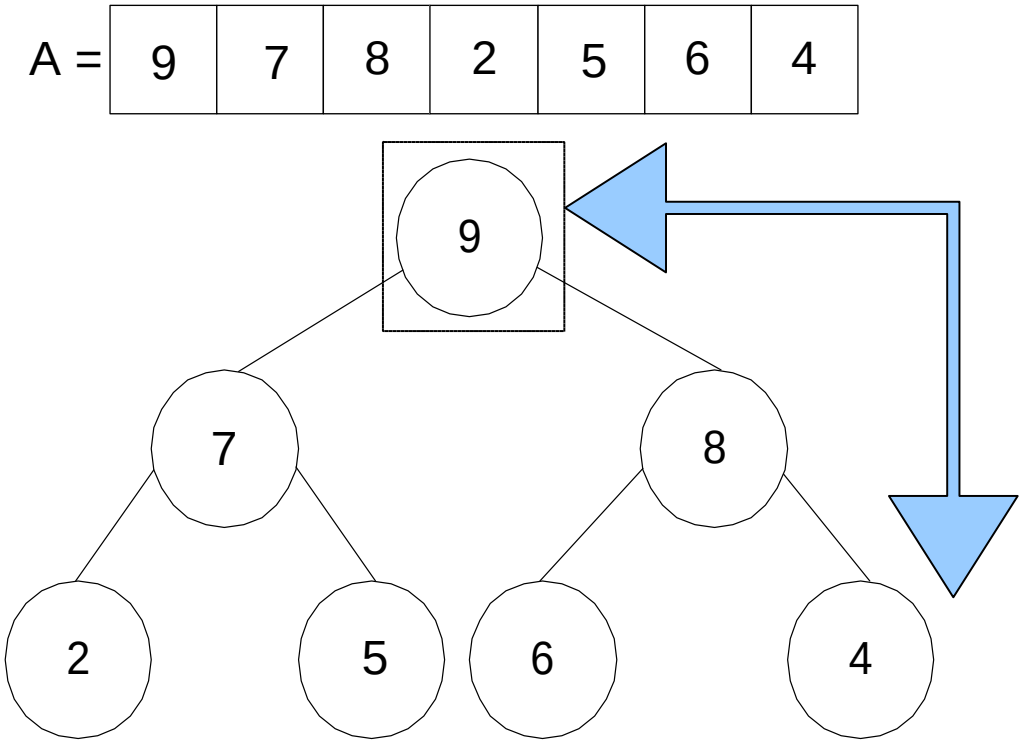




--- OK!

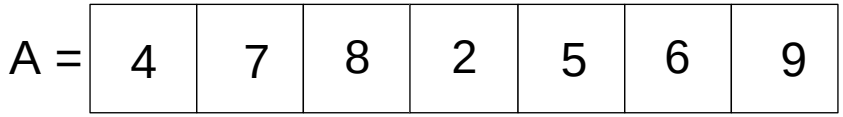
Heapsort

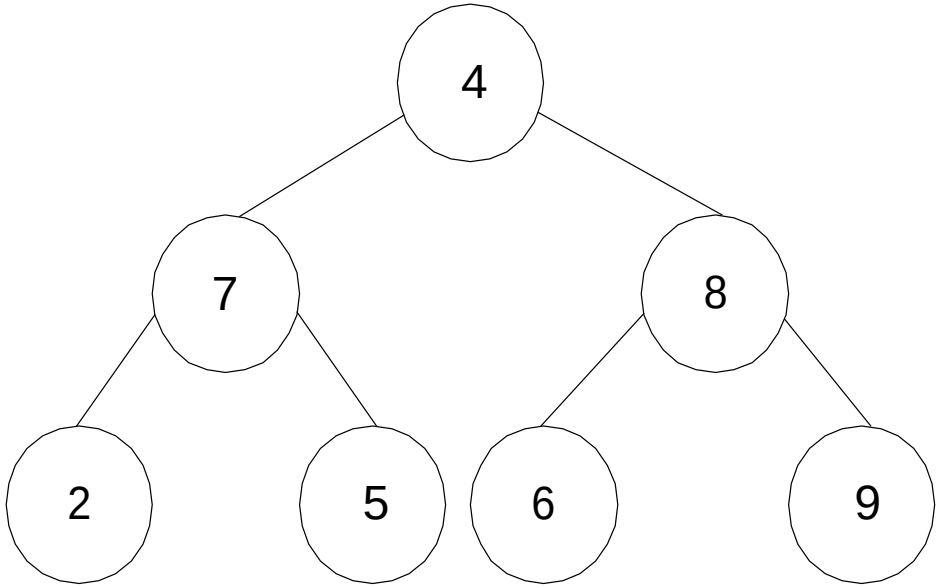
- Formado o heap, sua raiz corresponde à maior chave e portanto está posicionada na primeira posição do vetor



Heapsort

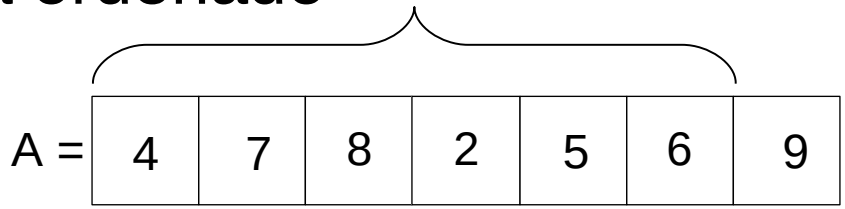
- Com isto, troca-se este valor da primeira posição com a última do vetor

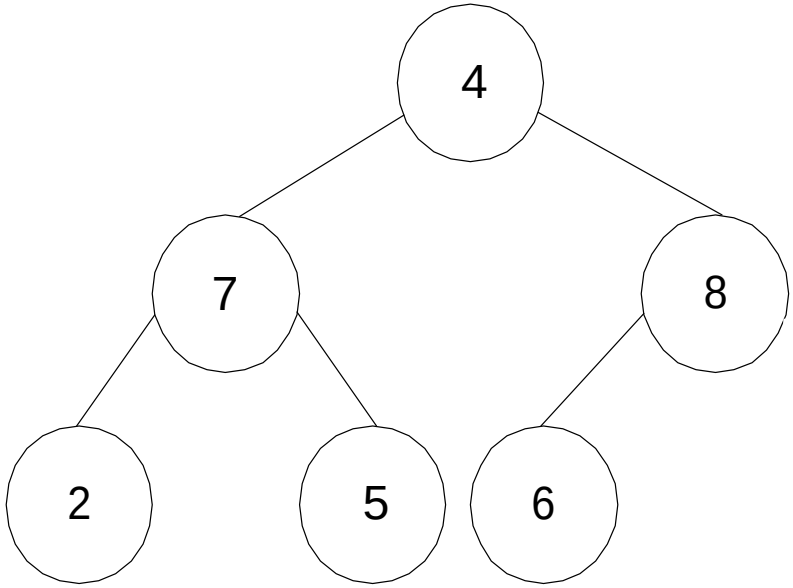




Heapsort

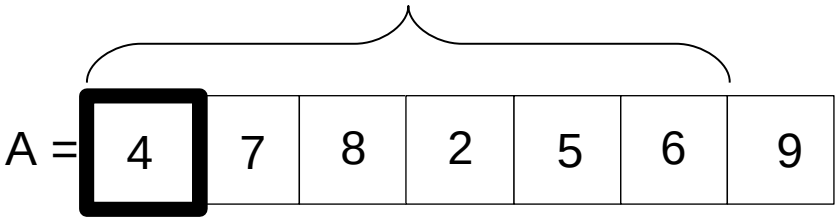
- O processo é repetido, excluindo-se do heap o elemento já ordenado

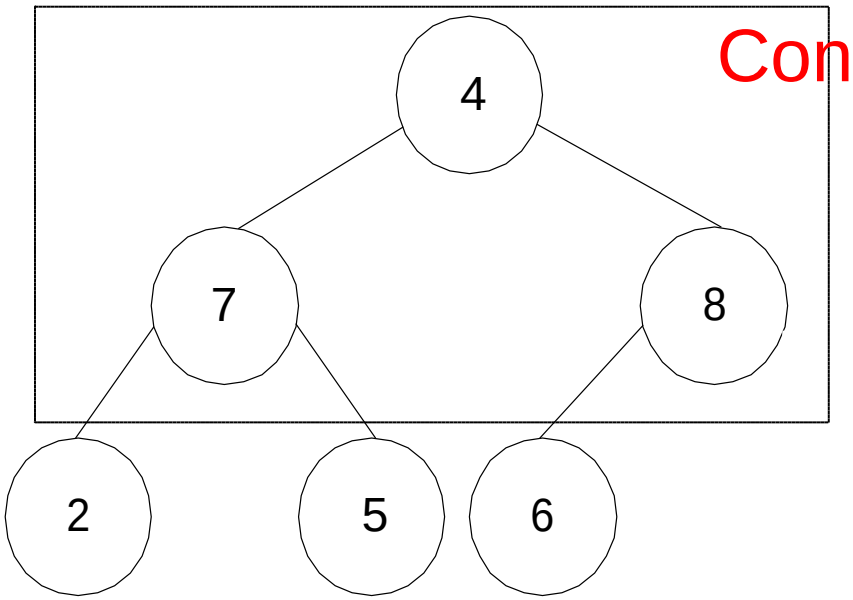




Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

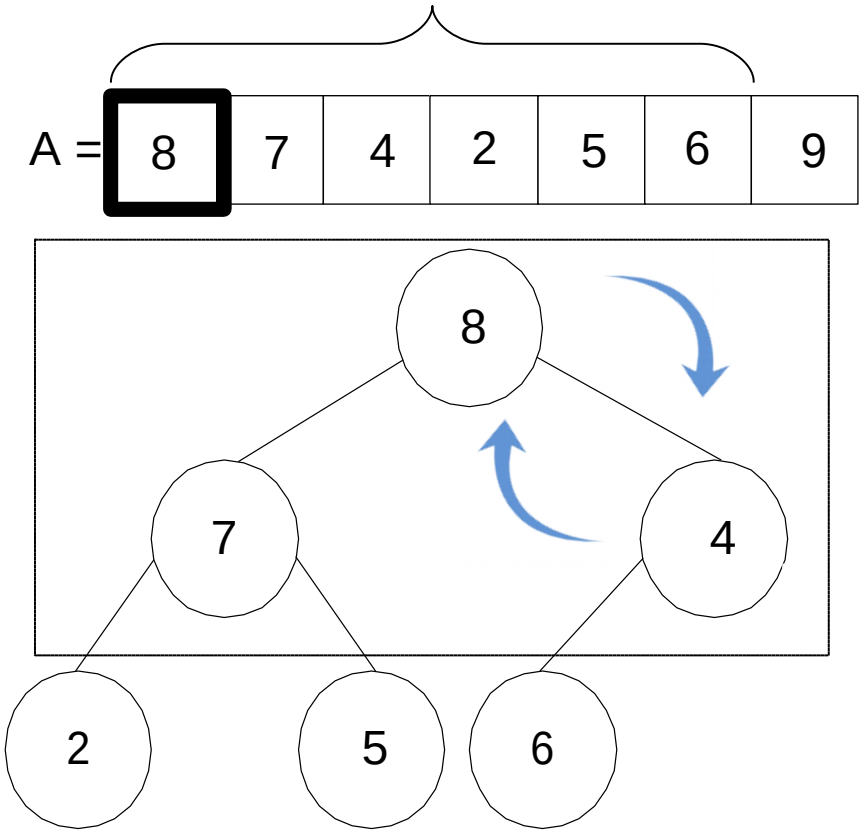




Condição Violada!

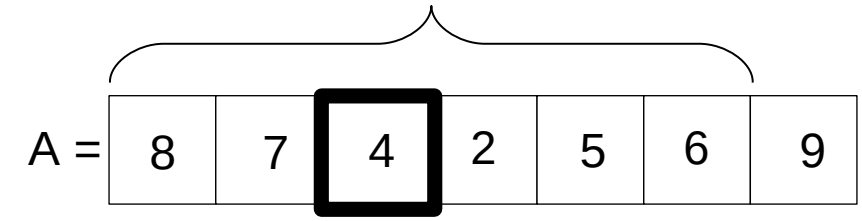
Heapsort

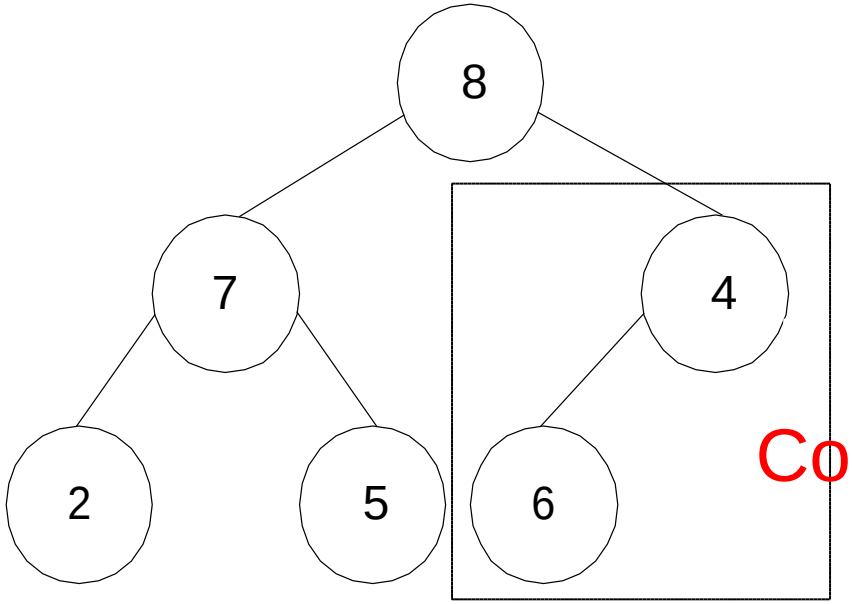
- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

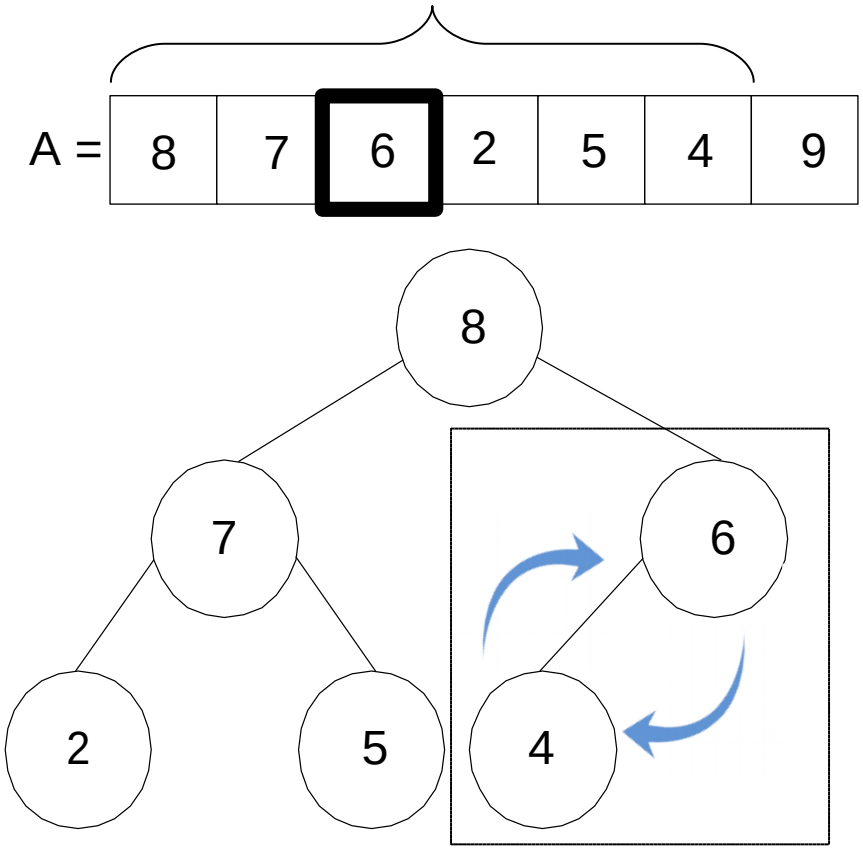




Condição Violada!

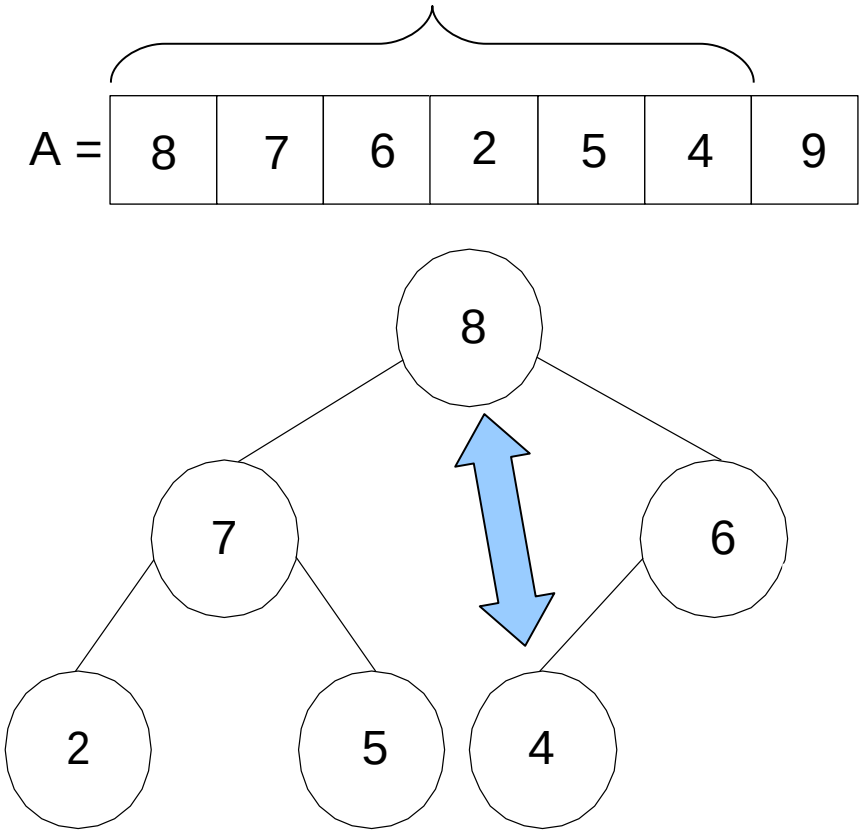
Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



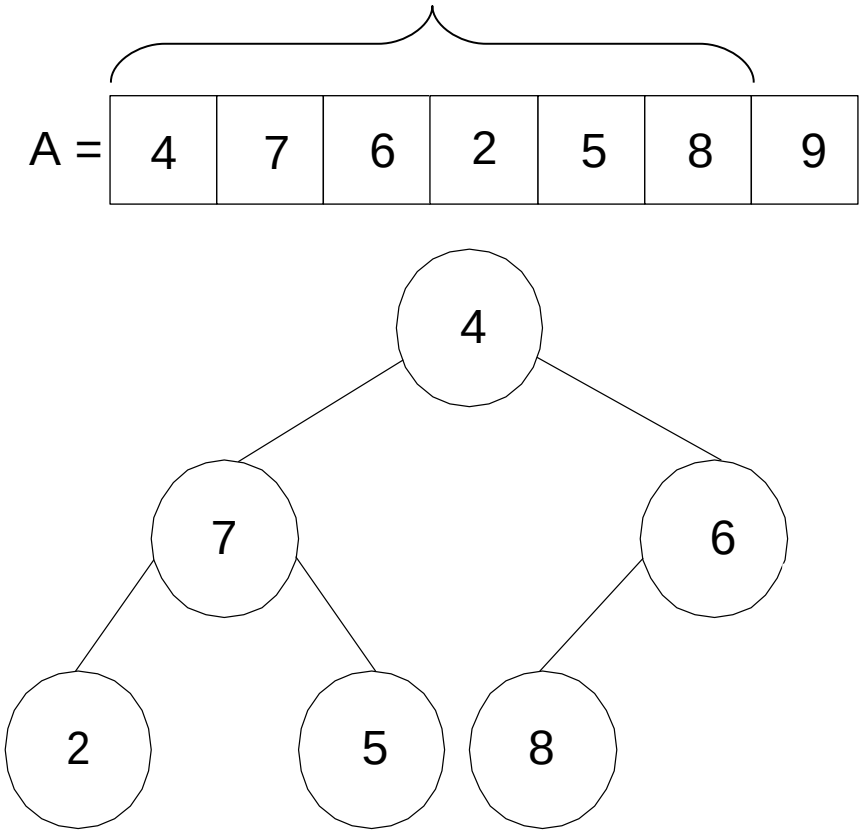
Heapsort

- Restabelecido o heap, novamente na primeira posição temos o maior elemento, que deve ser trocado com o elemento da última posição



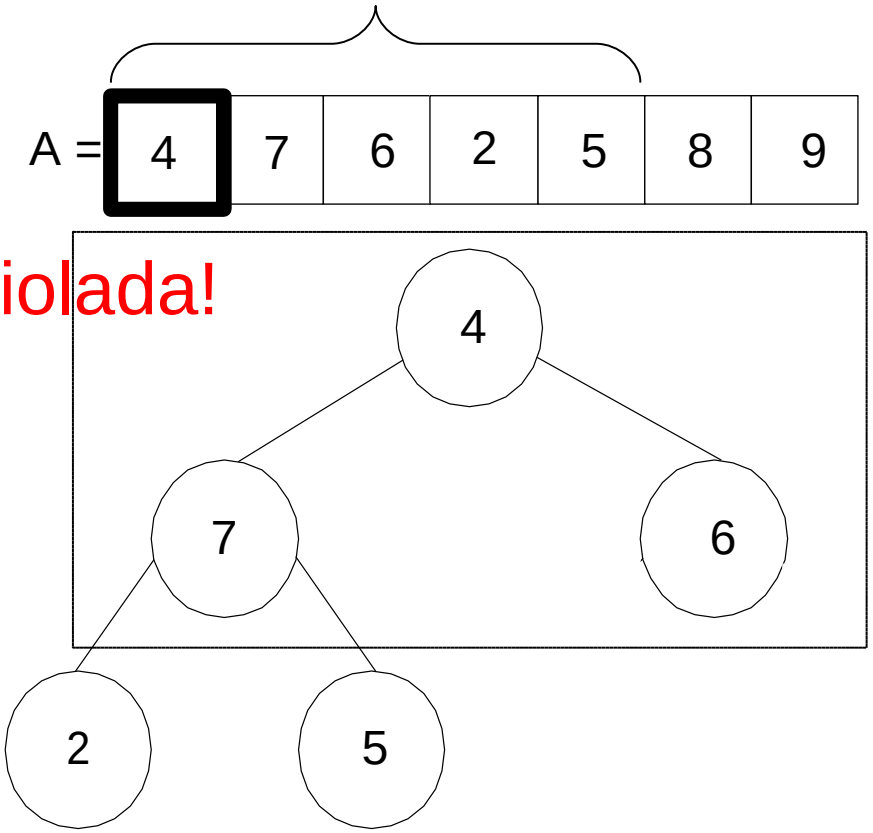
Heapsort

- Restabelecido o heap, novamente na primeira posição temos o maior elemento, que deve ser trocado com o elemento da última posição



Heapsort

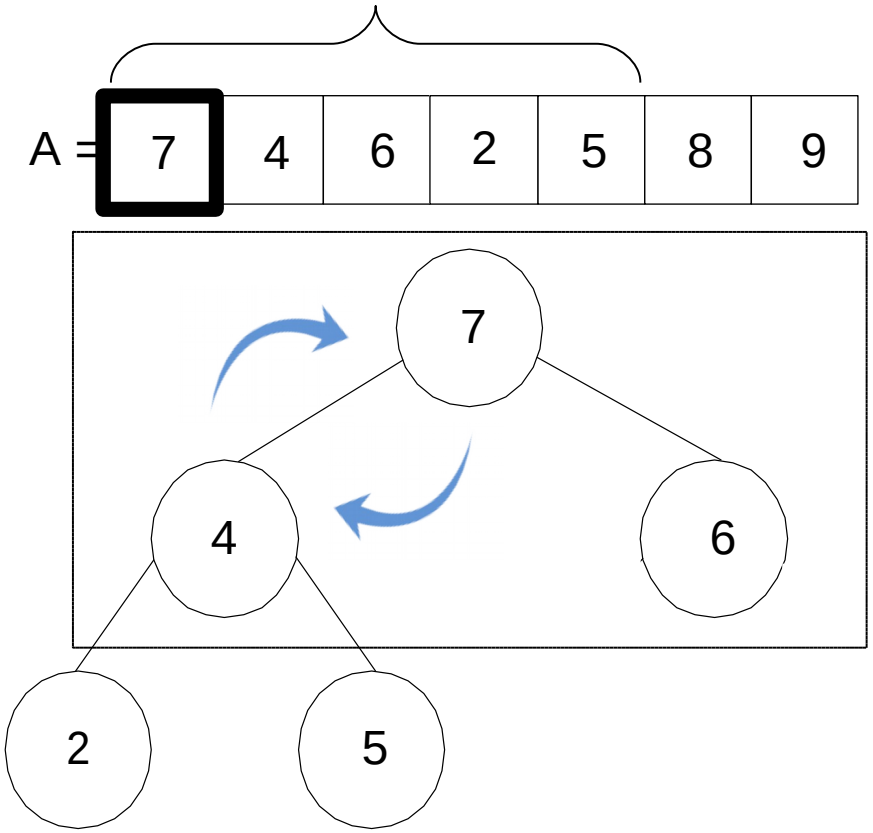
- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



Condição Violada!

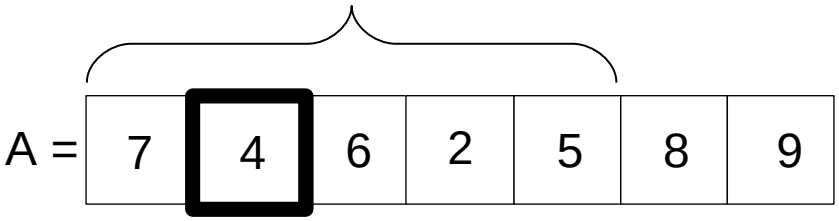
Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

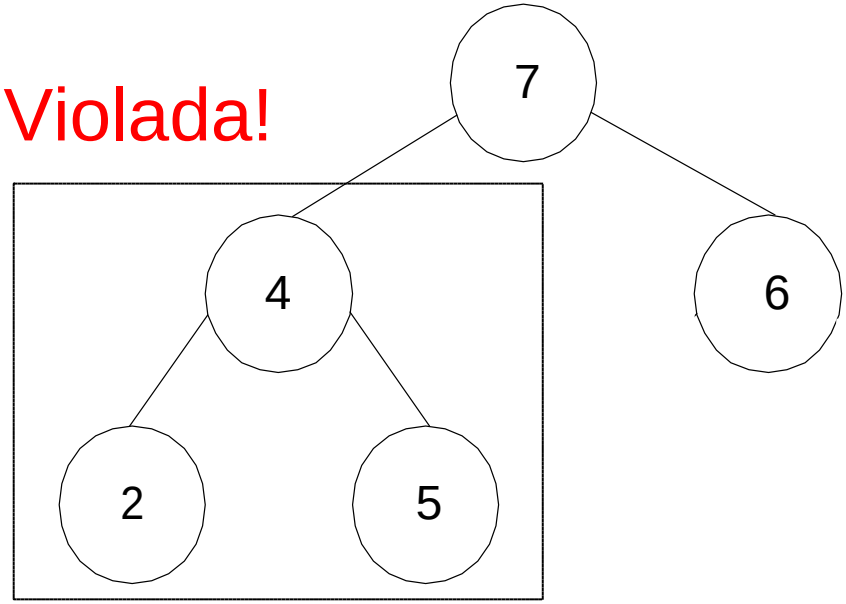


Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

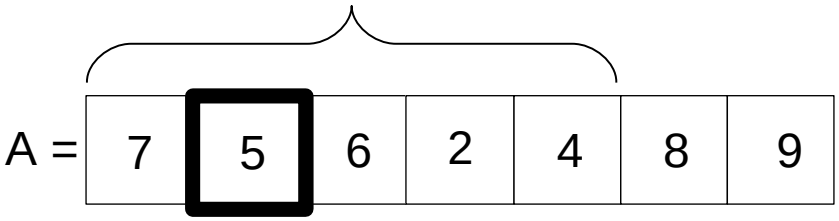


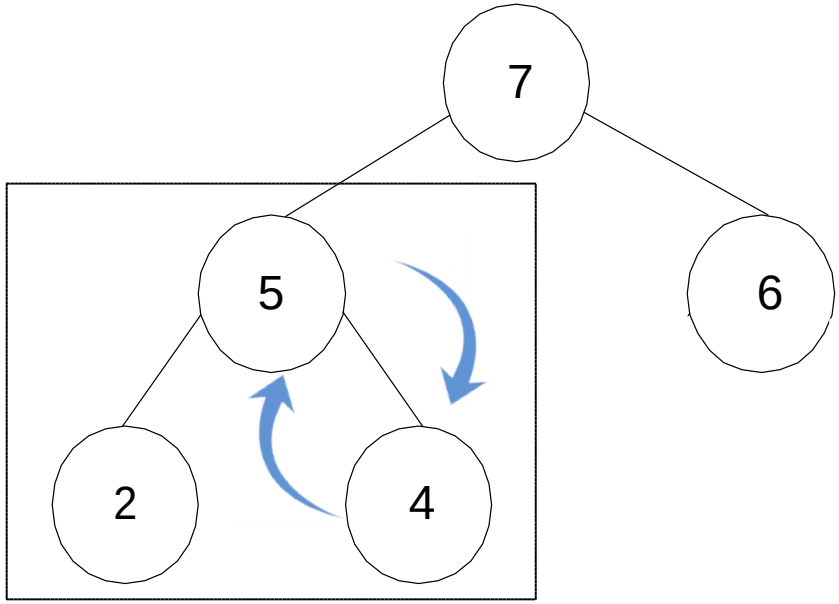
Condição Violada!



Heapsort

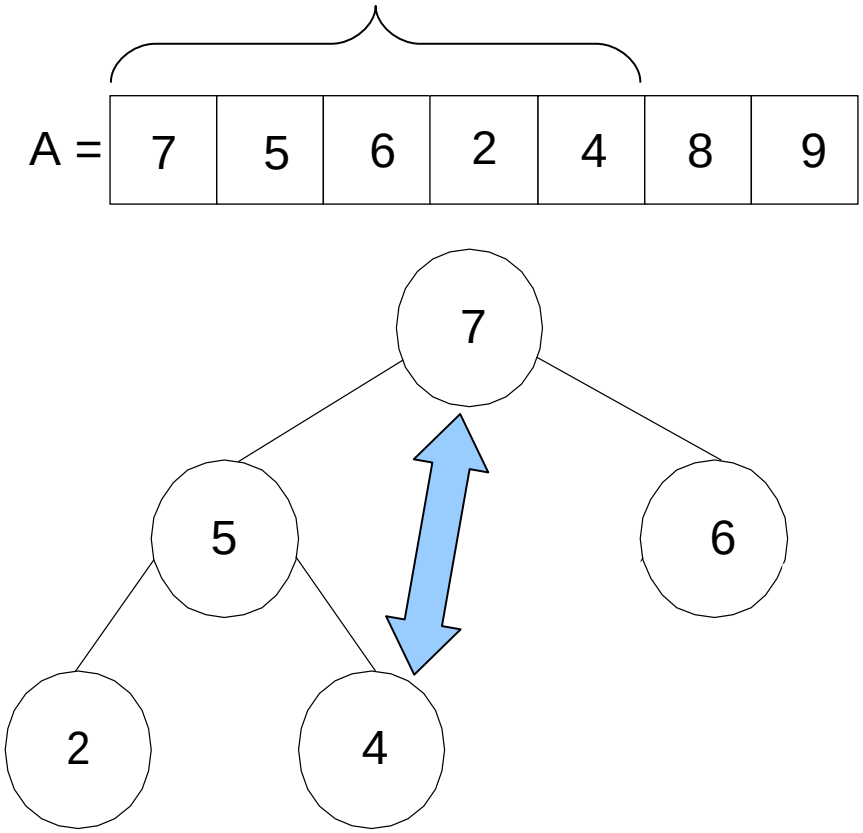
- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados





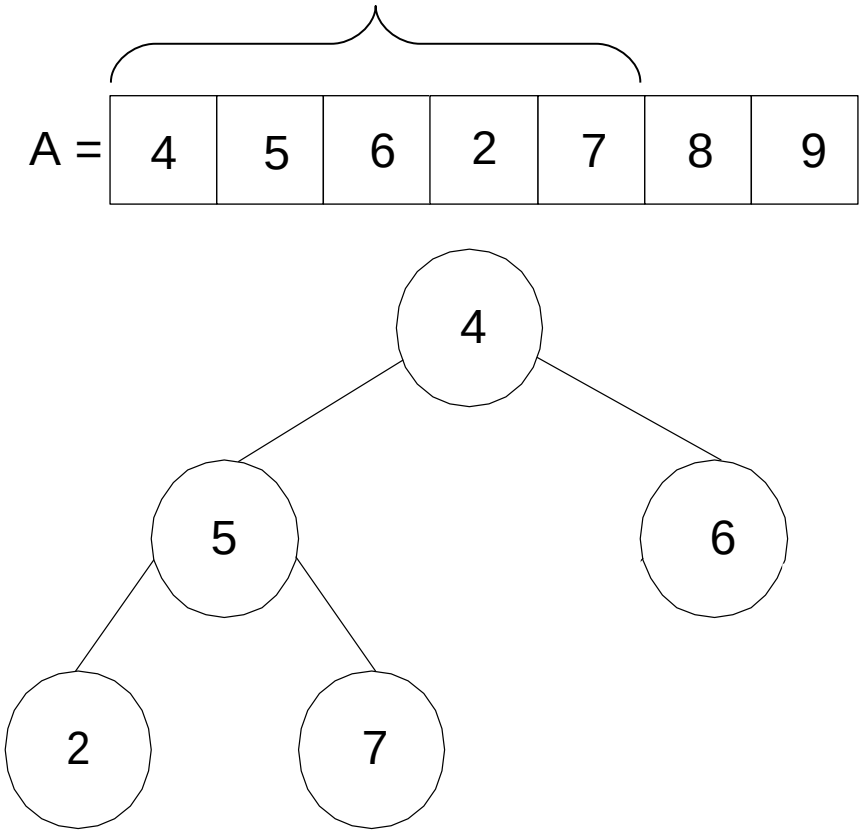
Heapsort

- Ao final de mais uma etapa, novamente o elemento da primeira posição é o maior, e deve ser trocado com o último.



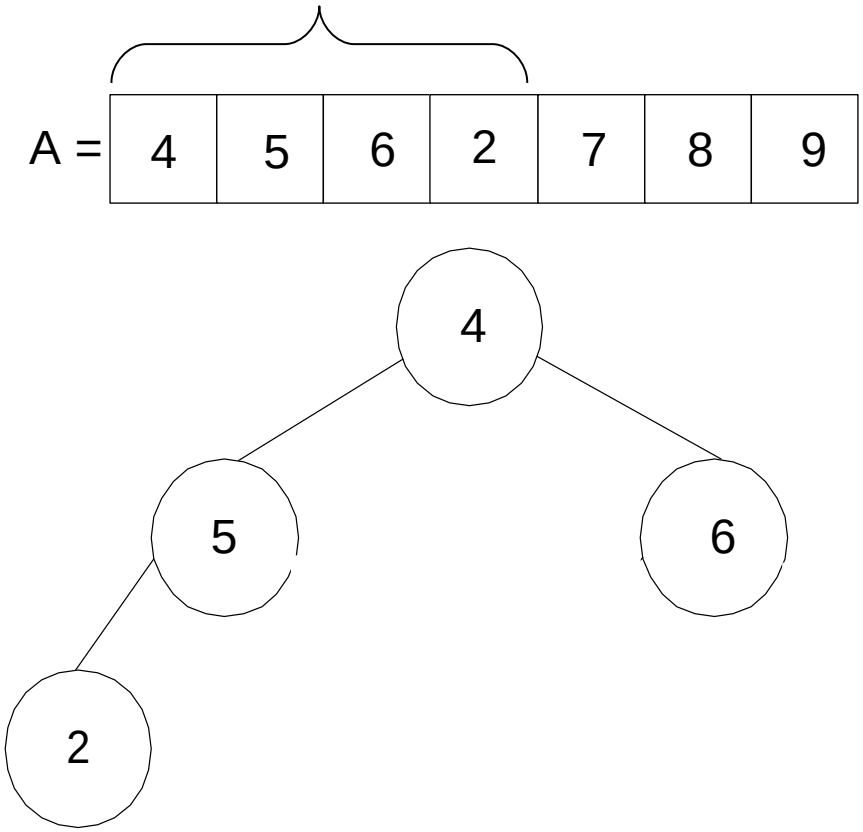
Heapsort

- Ao final de mais uma etapa, novamente o elemento da primeira posição é o maior, e deve ser trocado com o último.



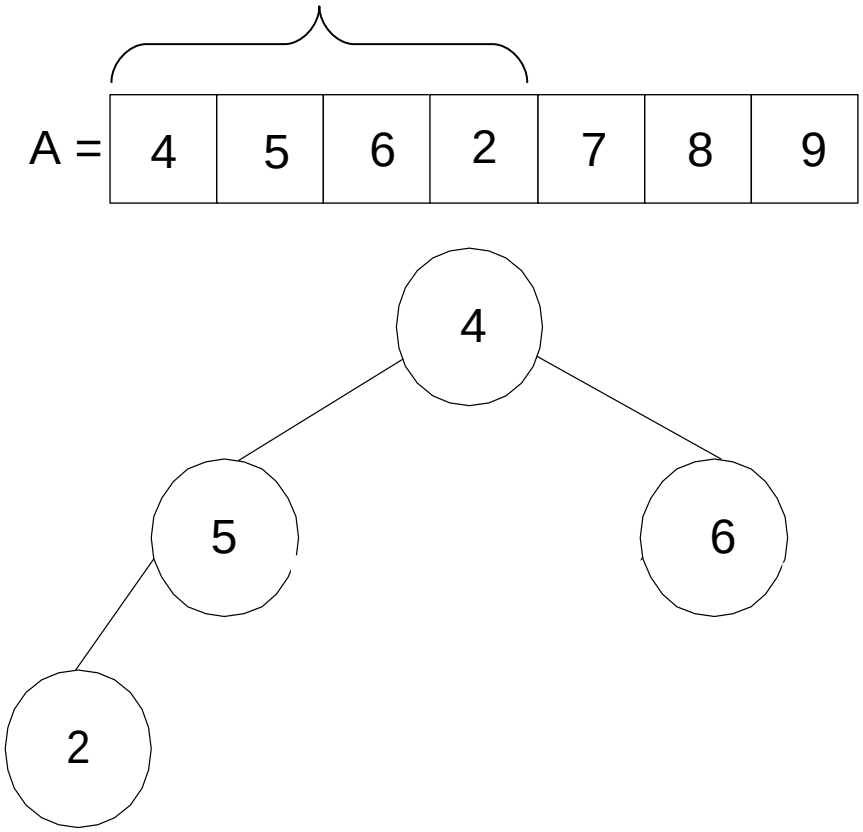
Heapsort

- O heap agora tem apenas 4 elementos.



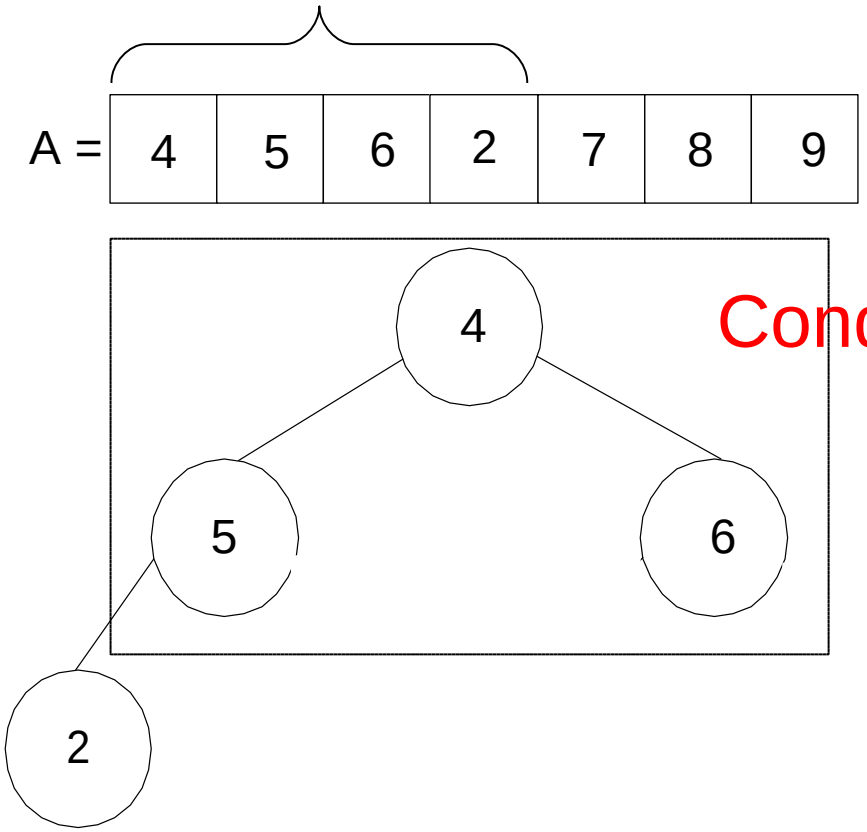
Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



Heapsort

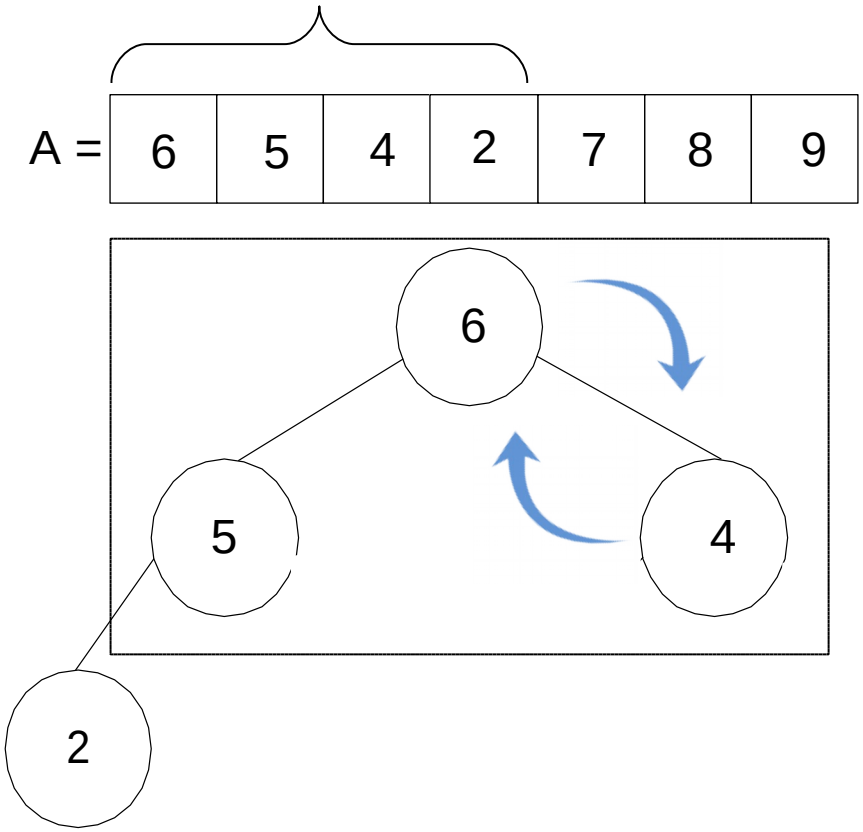
- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



Condição Violada!

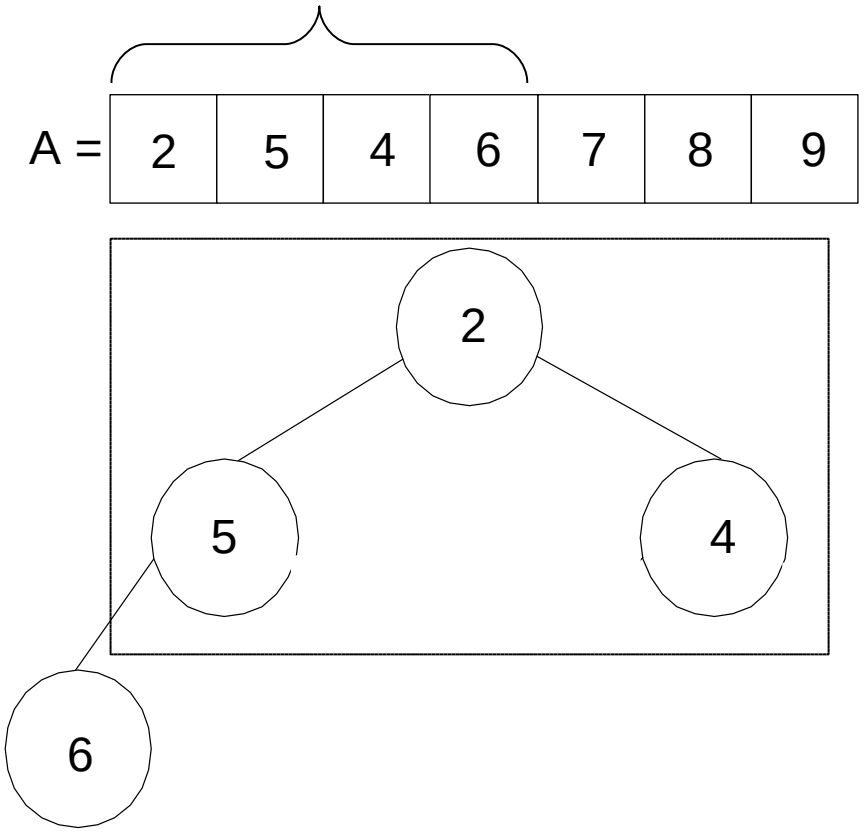
Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados



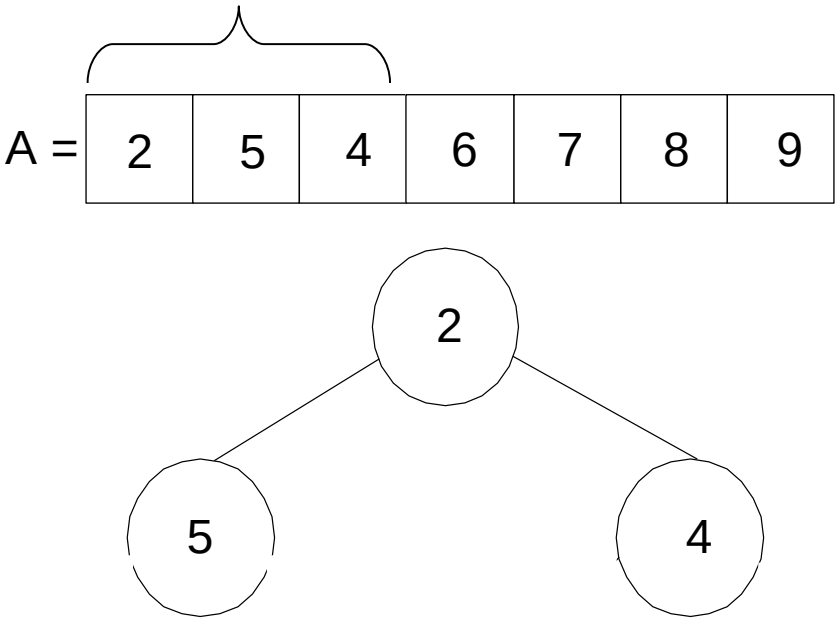
Heapsort

- Mais uma etapa finalizada, e novamente o primeiro elemento é trocado de posição com o último.



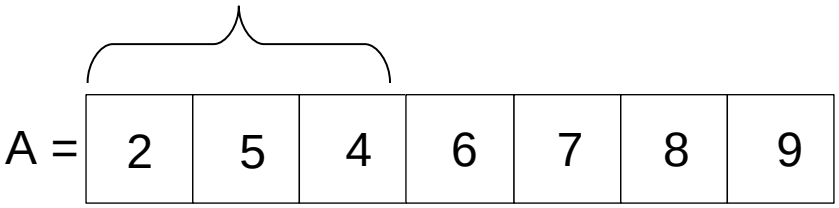
Heapsort

- O heap agora possui três elementos.

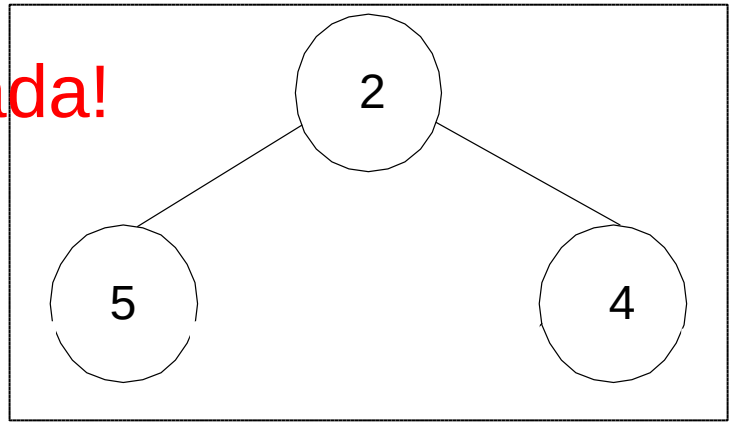


Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

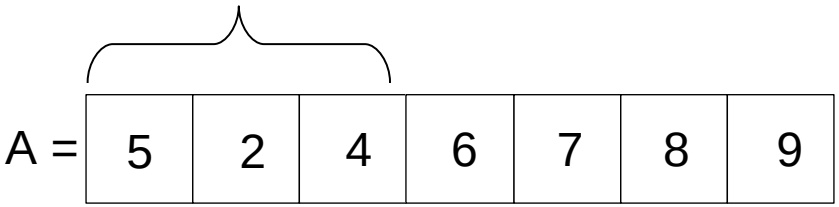


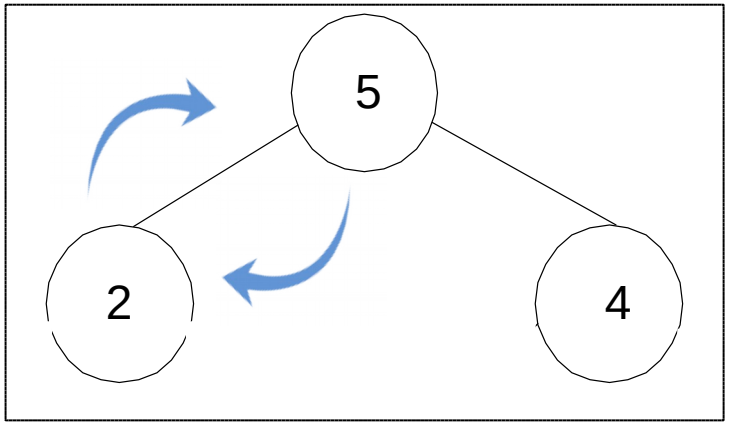
Condição Violada!



Heapsort

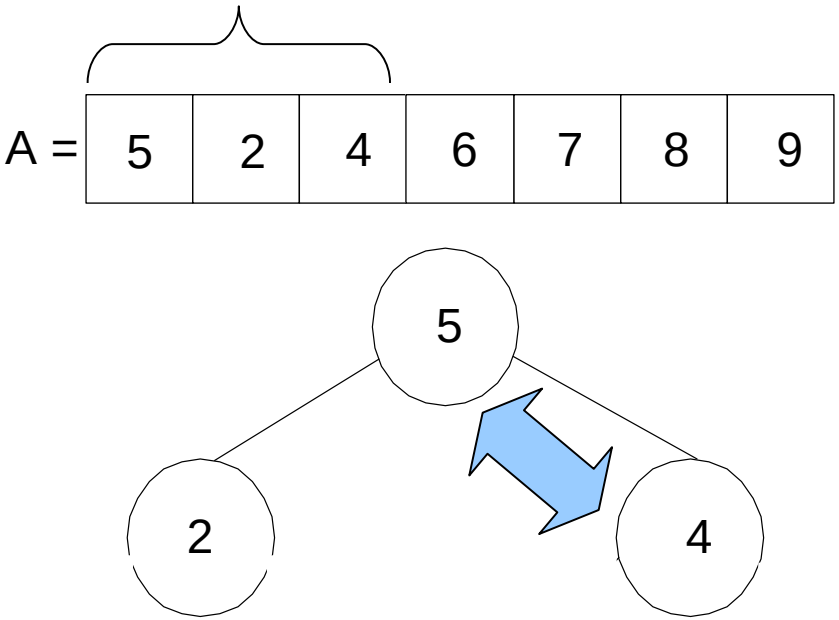
- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados





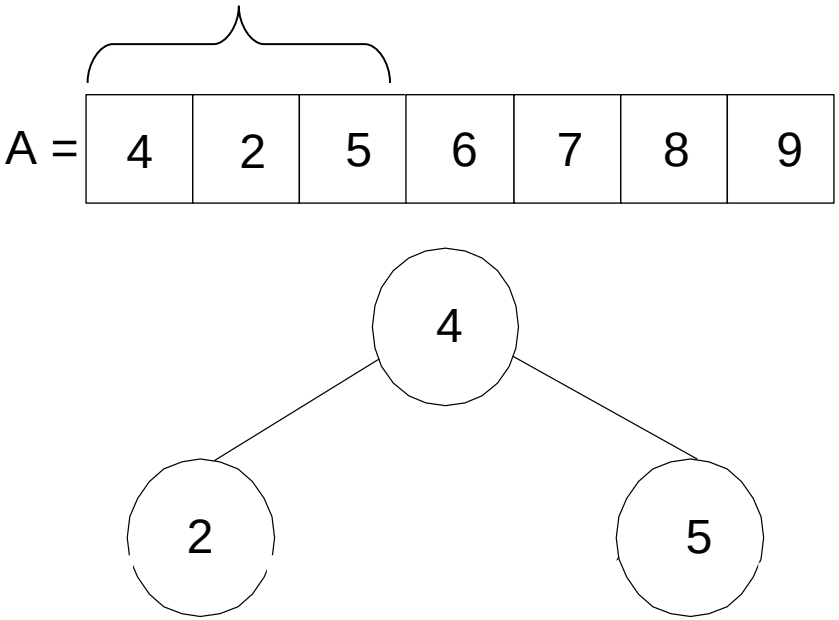
Heapsort

- Novamente, na primeira posição está o maior elemento, que deve ser trocado de posição com o último



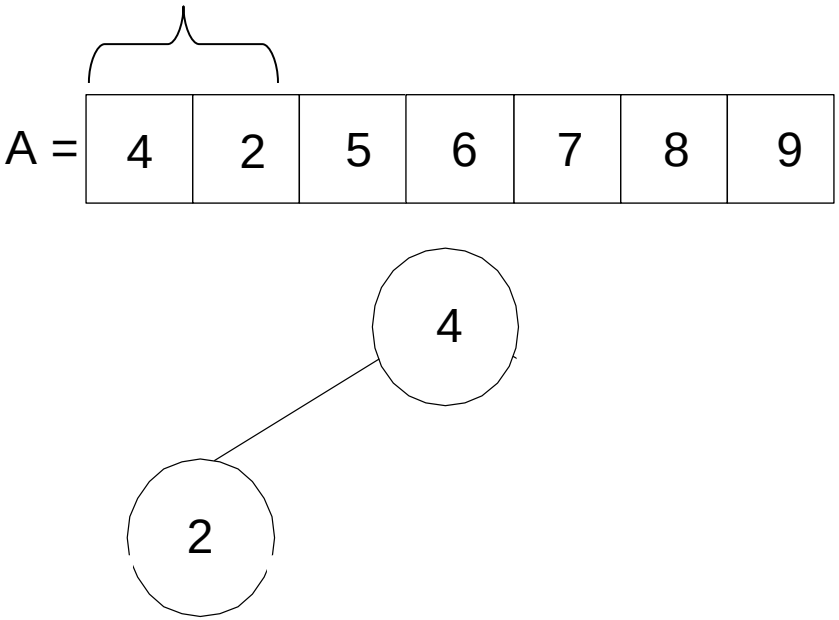
Heapsort

- Novamente, na primeira posição está o maior elemento, que deve ser trocado de posição com o último



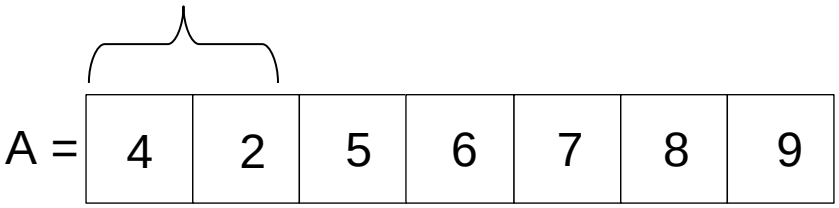
Heapsort

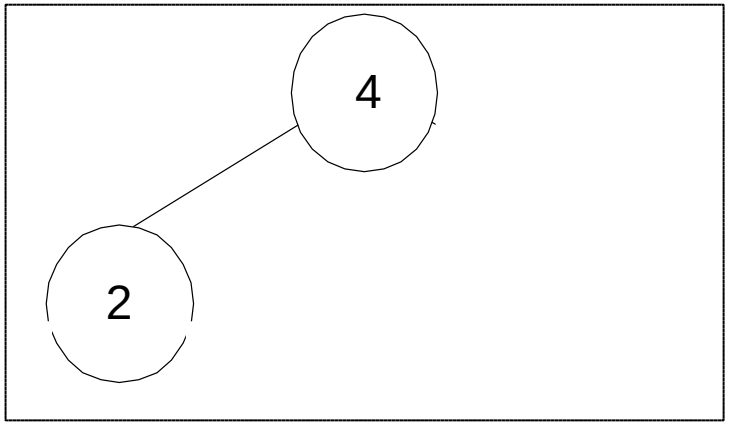
- O heap agora tem apenas dois elementos



Heapsort

- Para restabelecer o heap, percorre-se as chaves no vetor a partir da posição 1 até a posição n div 2, mas considerando apenas os elementos eventualmente trocados

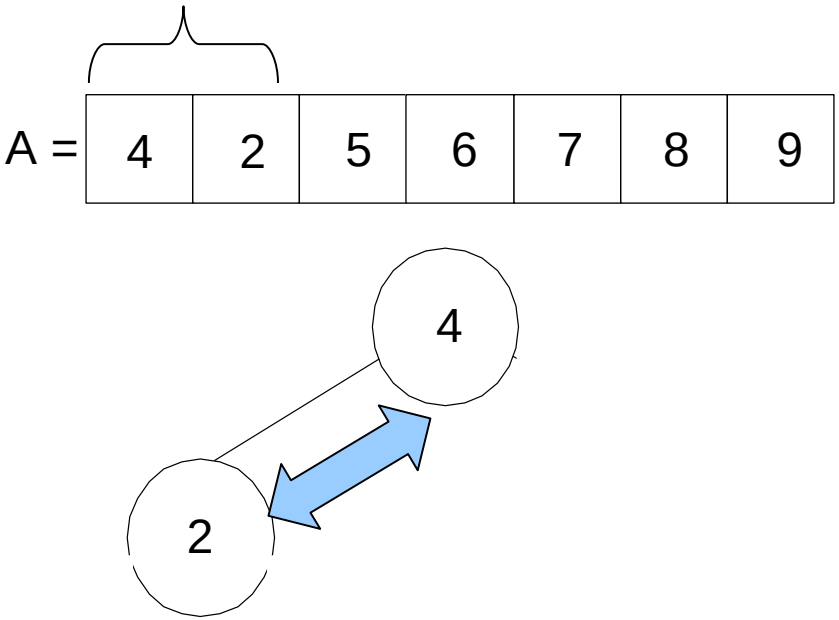




--- OK!

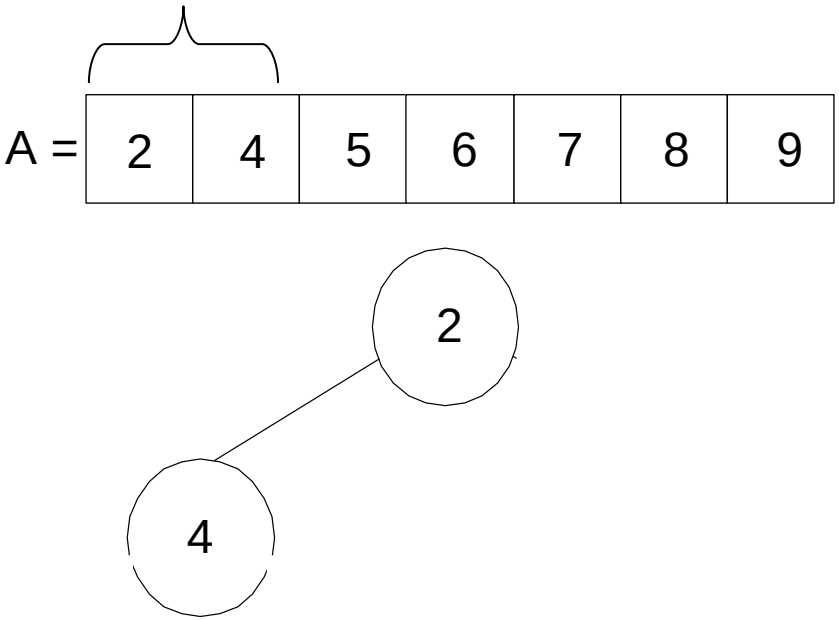
Heapsort

- Novamente, na primeira posição está o maior elemento, que deve ser trocado de posição com o último



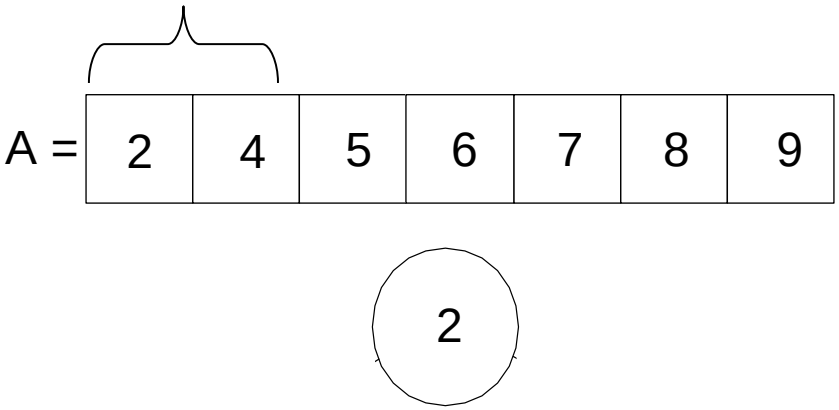
Heapsort

- Novamente, na primeira posição está o maior elemento, que deve ser trocado de posição com o último



Heapsort

- Novamente, na primeira posição está o maior elemento, que deve ser trocado de posição com o último



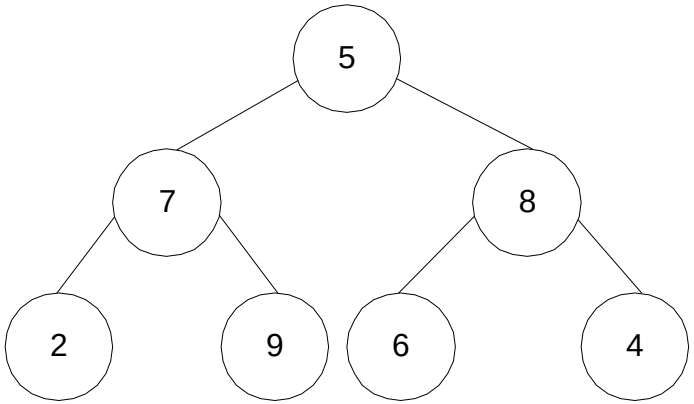
Heapsort

Sequência Ordenada!

A =	2	4	5	6	7	8	9

```
void Constroi (int *A,  int n)
{
        int esq;

         esq = n / 2 + 1;
         while (esq > 1)
        {
                esq --;
                Refaz (esq, n, A);
         }
}
```



```
void Refaz (int esq, int dir, int *A)
{
        int i = esq;
        int j;
        int x;
        j  = i * 2;
        x = A[i];
         while (j<= dir)
        {
                if (j < dir)
                     if (A[j] < A [j+1])
                                j ++;
                 if (x >= A[j]) break;
                 A[i] := A[j];
                 i := j;
                 j := 2 * i;
        }
        A[i] := x;
}
```

A =

5	7	8	2	9	6	4

Heapsort

- Uma vez que além da seleção do maior termo, a cada etapa é preciso a reconstrução do heap, a utilização do heapsort não é recomendada para uma quantidade pequena de termos.
- Seu comportamento é O(nlogn) qualquer que seja a entrada, o que o torna apropriado quando não se pode tolerar um desempenho desfavorável

Heapsort

- Exercício:
  - Ordene a sequência 19 5 30 11 9 12 20 usando heapsort e mostre o comportamento passo a passo da execução do método.

Bibliografia

- ZIVIANI, Nivio. **Projeto de Algoritmos com implementações em Pascal e C.** São Paulo: Pioneira Thomson Learning. 2a. Edição 2004.
- WIRTH, Niklaus. **Algoritmos e Estruturas de Dados.** LTC. 1999.
- Azeredo, P. A. **Métodos de Classificação de Dados e Análise de suas Complexidades**. Riode Janeiro: Editora Campus, 1996;
- Moraes, Celso Roberto. **Estruturas de Dados e Algoritmos.** São Paulo: Berkeley Brasil, 2001;