Heapsort

Heapsort é um método de ordenação cujo princípio de funcionamento é o mesmo utilizado para a ordenação por seleção.

Selecione o maior (ou menor) item do vetor e a seguir troque-o com o item que está na última (ou primeira) posição do vetor; repita estas duas operações com os **n - 1** itens restantes; depois com os **n - 2** itens; e assim sucessivamente.

UNIVASF

Heapsort

O custo para encontrar o maior (ou o menor) item entre **n** itens e de **n - 1** comparações.

Este custo pode ser reduzido?

SIM.

Este custo pode ser reduzido através da utilização de uma estrutura de dados chamada de fila de prioridades.

UNIVASF

Fila de Prioridades

Fila:
Sugere espera por algum serviço.

Prioridade:
Sugere que o serviço será fornecido com base em um critério.

Fila de Prioridade:
Conjunto de elementos com o comportamento elemento-de-maior-valor (menor-valor) é o primeiro a abandonar o conjunto de elementos.

Aplicações de Filas de Prioridades

- Sistemas operacionais usam filas de prioridades, onde as chaves representam o tempo em que eventos devem ocorrer.

- Alguns métodos numéricos iterativos são baseados na seleção repetida de um item com maior (menor) valor.

- Sistemas de gerência de memória usam a técnica de substituir a página menos utilizada na memória principal do computador por uma nova página.

UNIVASF

Fila de Prioridades

Operações Principais:

1. Construir uma fila de prioridades a partir de um conjunto com **n** itens;
2. Retirar o item com maior prioridade;
3. Restaurar a fila de prioridades.

Forma de implementação:

Árvore binária

UNIVASF

Árvore Binária

Considerando as características de uma árvore binária de busca, um aluno atento chegará a seguinte reflexão:

As chaves poderiam ser inseridas, uma a uma, em uma árvore binária de busca;

Após a inserção de todas as chaves a árvore poderia ser percorrida, por exemplo, em in-ordem e as chaves seriam obtidas em ordem crescente.

UNIVASF

Árvore Binária

Desvantagens da utilização de uma árvore binária de busca:

- Necessidade de área de memória adicional para o armazenamento da árvore;
- O que aconteceria se as chaves já se encontrarem em ordem ou em ordem inversa?
  - Seria gerada um árvore degenerada. O que significa que para a inserção do ***i***-ésimo elemento seriam requeridas ***i-1*** comparações, o que, praticamente, elimina a vantagem de se utilizar uma árvore no processo.

UNIVASF

Heap Sort

As deficiências da classificação utilizando árvore binária ordenada são eliminadas no método denominado **heap sort**.

O heap sort é um método *in situ* de complexidade constante, independente da ordem da entrada.

Um heap decrescente de tamanho **n** é implementado utilizando-se uma árvore binária quase completa representada sequencialmente, com a característica de que todo nó possui um valor maior ou igual aos valores armazenados em seus filhos, caso estes existam.

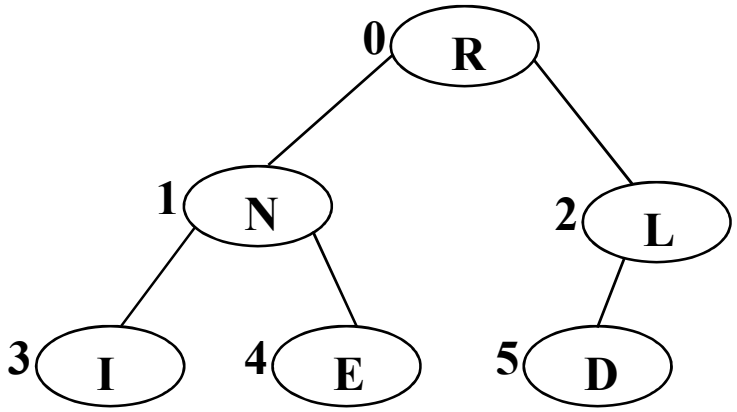
UNIVASF

Heap Sort

Árvore Binária quase completa é uma árvore binária onde (revisão):

1. Cada folha na árvore está no nível **d** ou no nível **d-1**;
2. Para todo nó **nd** que possui um descendente direito no nível **d**, todo descendente esquerdo de **nd** é folha no nível **d** ou tem 2 filhos.

Exemplo: Árvore binária quase completa



Índices	0	1	2	3	4	5
Valores	R	N	L	I	E	D

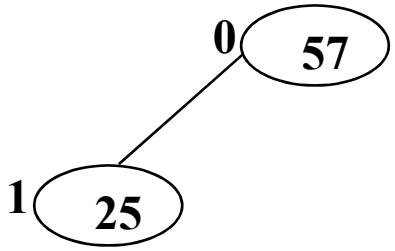
Relação: info[j] ≤ info[(j-1)/2]
para 0 ≤ ((j-1)/2) < j ≤ n-1

Heap Sort

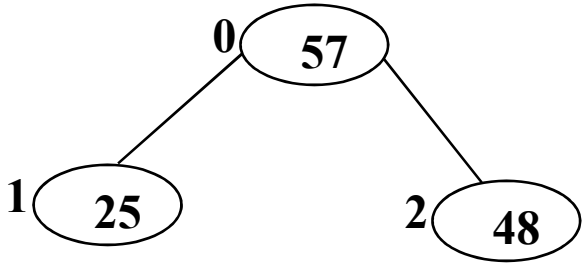
Para uma melhor compreensão vamos analisar o processo de construção de um heap partindo de um conjunto de **n** elementos (chaves) sobre o exemplo onde o vetor de chaves originalmente é:

25 57 48 37 12 92 86 33

57 25 48 37 12 92 86 33

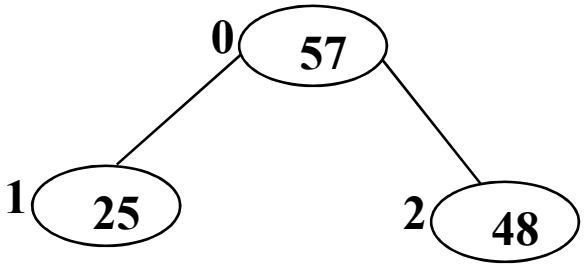


57 25 48 37 12 92 86 33

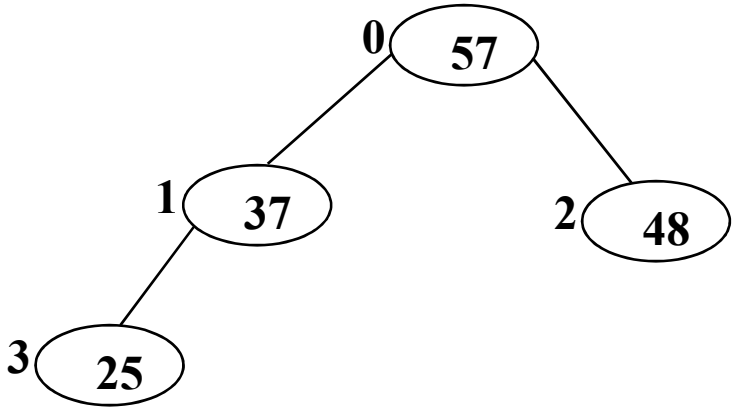


Heap Sort

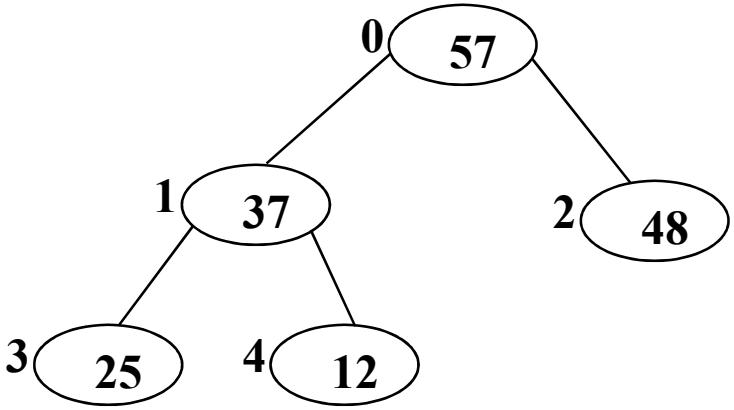
57 25 48 37 12 92 86 33



57 37 48 25 12 92 86 33



57 37 48 25 12 92 86 33

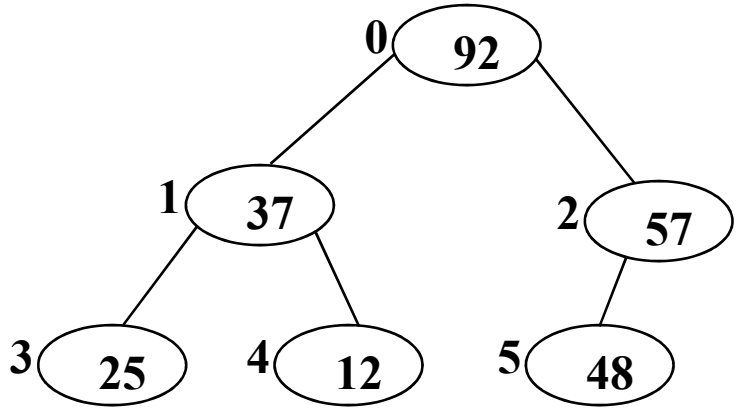


Heap Sort

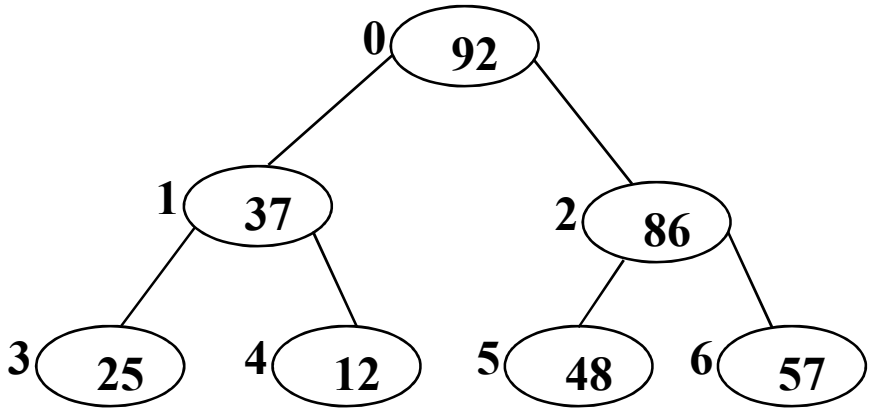
57 37 48 25 12 92 86 33

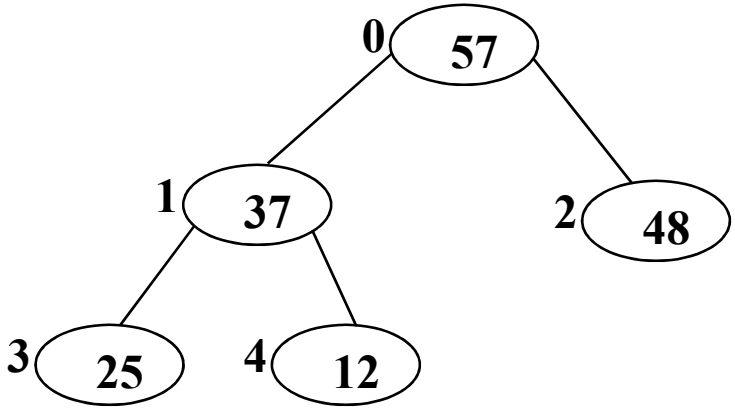
57 37 92 25 12 48 86 33

92 37 57 25 12 48 86 33



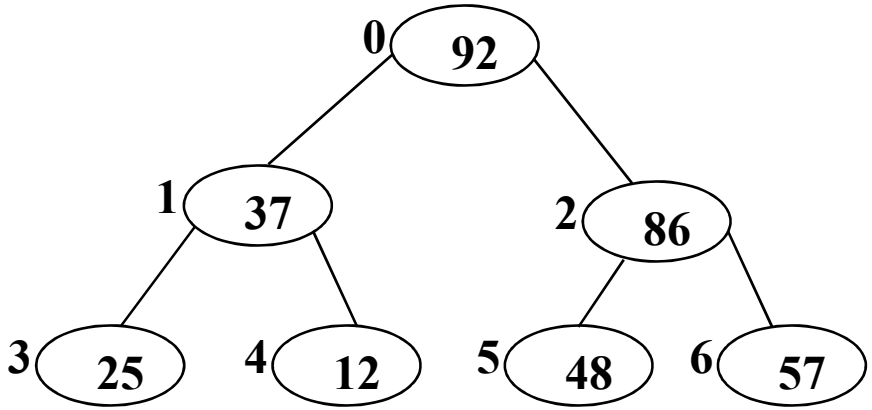
92 37 86 25 12 48 57 33



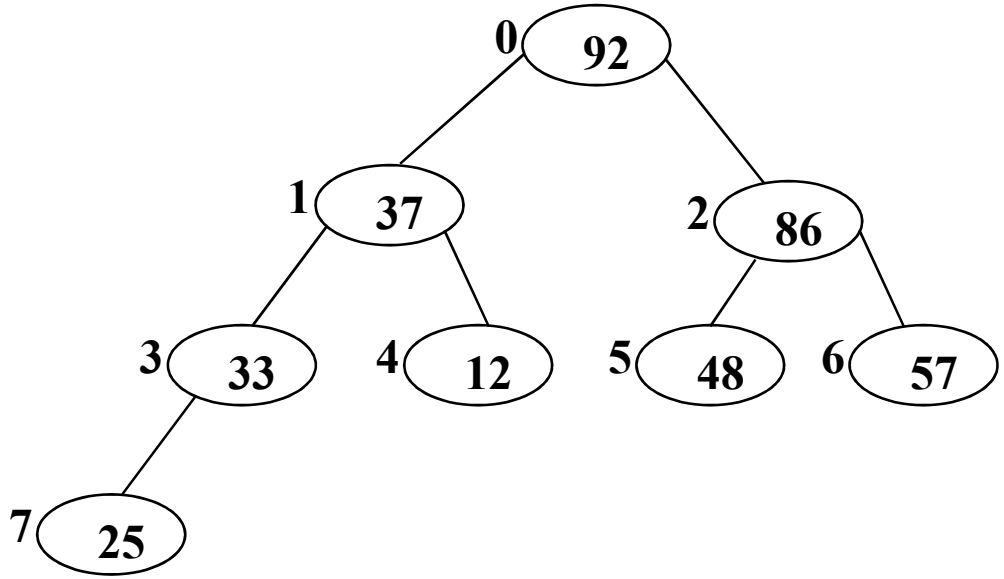


Heap Sort

92 37 86 25 12 48 57 33



92 37 86 33 12 48 57 25



Heap Sort

Exercício:

Com base no que foi discutido implemente uma função, em C, que receba, como parâmetros, um vetor de inteiros e a quantidade de elementos no mesmo e retorne um vetor que represente um heap decrescente com os valores contidos inicialmente no vetor.

UNIVASF

```
heap (int *x, int n)
{
  int i, e, s, f;
  for (i=1; i<n; i++)
  {
    e = x[i];
    s = i;
    f = (s-1)/2;
    while (s>0 && x[f]<e)
    {
      x[s] = x[f];
      s = f;
      f = (s-1)/2;
    }
    x[s] = e;
  }
}
```

Heap Sort

Agora que já definimos como receber um conjunto de chaves e transformá-lo em um heap devemos determinar como iremos utilizá-lo.

Se observarmos a característica básica da árvore que representa o heap, perceberemos que a raiz contém o elemento de maior valor do conjunto de chaves.

Sendo assim, podemos removê-lo e posicioná-lo no final do vetor que armazena o heap. Contudo, para isso temos que reorganizar o heap, mantendo sua propriedade e liberando espaço no final do vetor para colocar o elemento mencionado.

UNIVASF

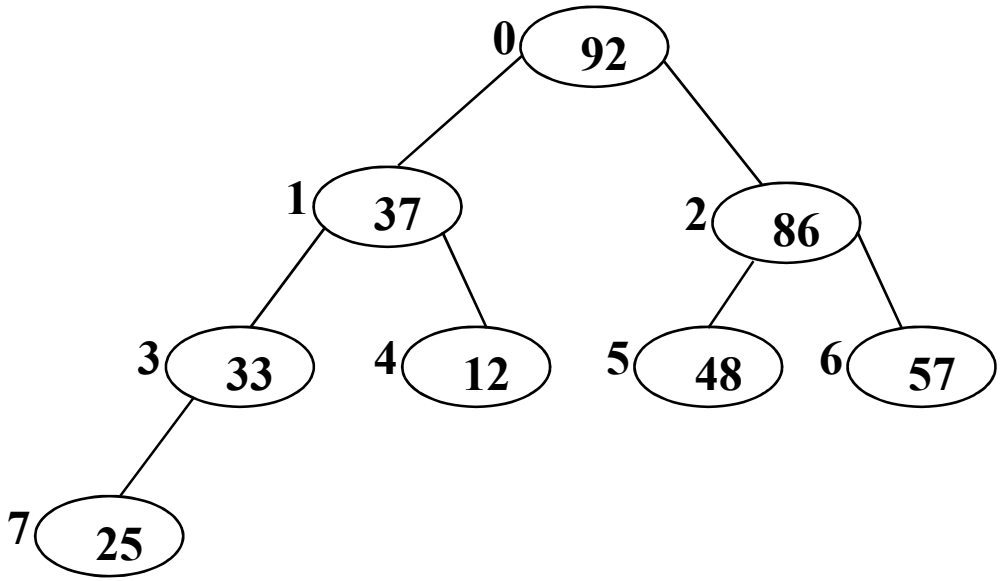
Heap Sort

Vamos analisar este processo retomando o exemplo anterior.

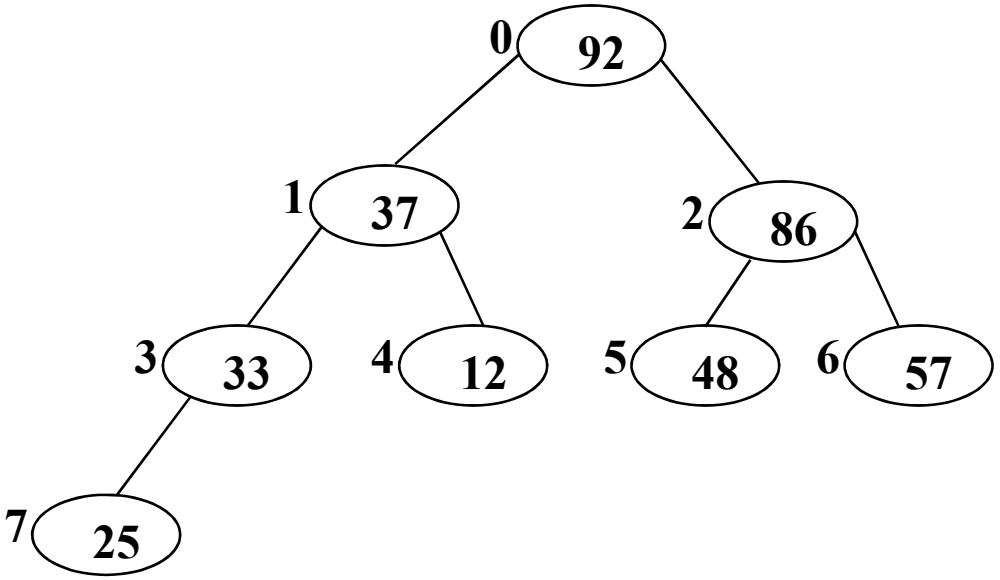
Considerando o vetor de chaves:

92 37 86 33 12 48 57 25

Que representa o heap:

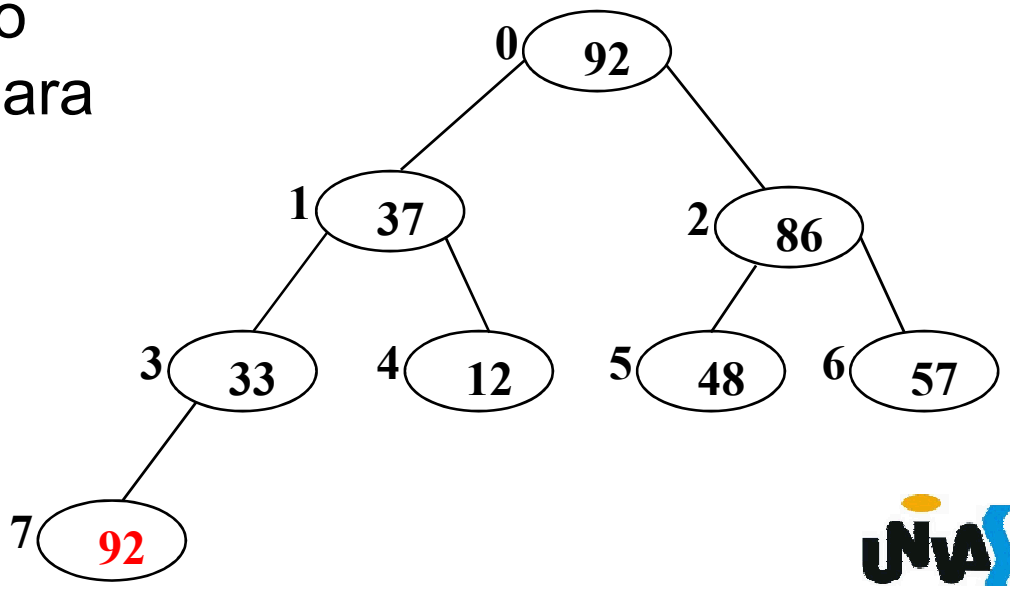


Heap Sort

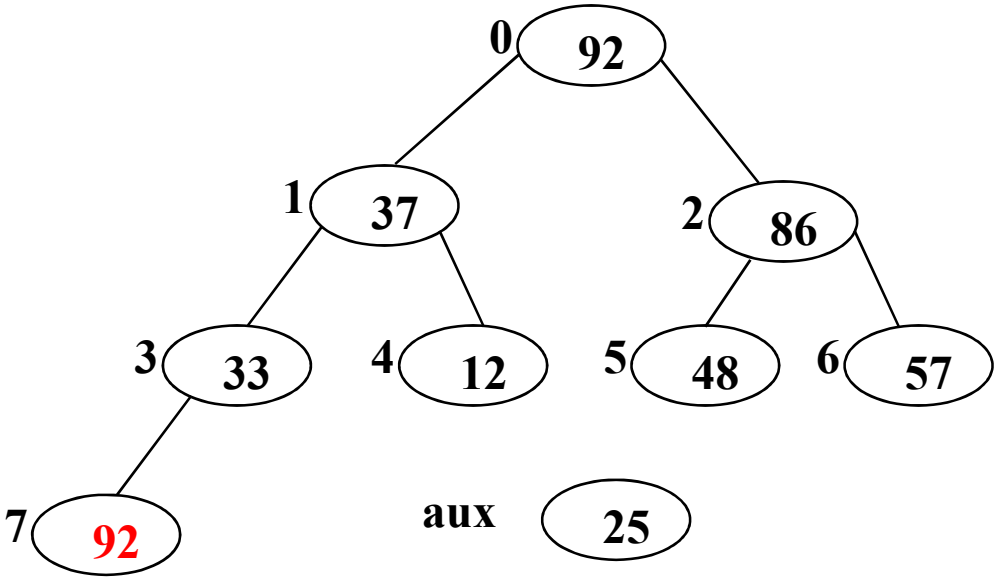


Armazenaremos o último elemento do nosso vetor em uma variável auxiliar.

Agora podemos copiar o elemento de maior valor para aposição final do vetor.



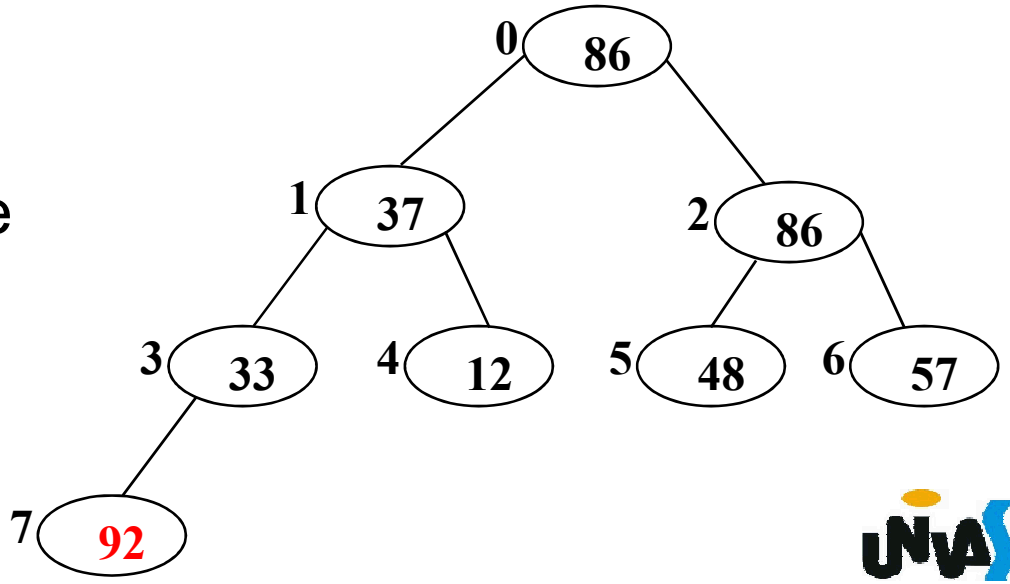
Heap Sort



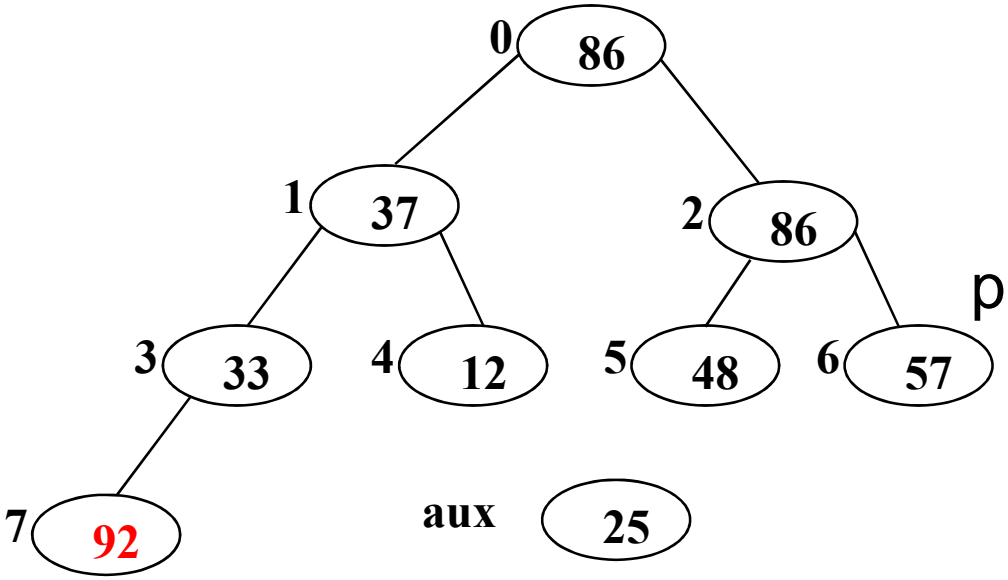
Podemos considerar a posição com índice zero como livre no vetor, reorganizar o heap e posicionar o valor armazenado em aux na posição correta.

Como fazer isto?

Escolhendo o maior dentre os filhos da raiz e deslocá-lo para tal posição.

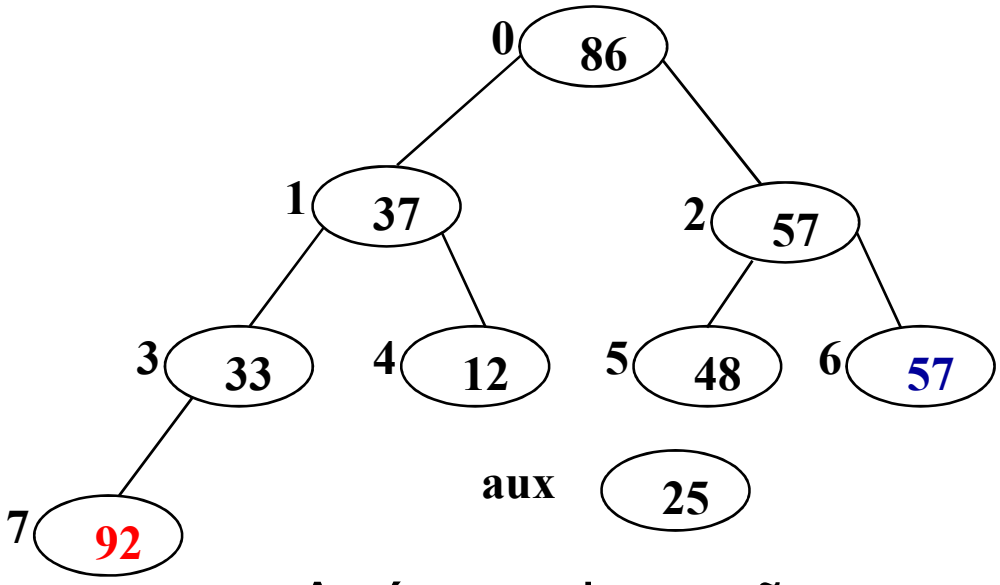


Heap Sort



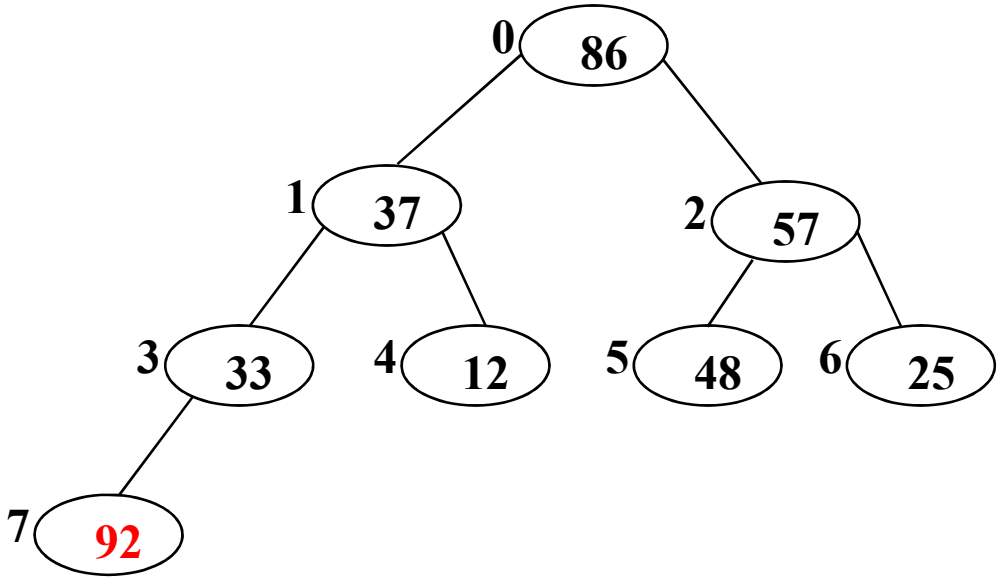
Mantendo este raciocínio, teremos a posição com índice 2 livre e a preencheremos com seu filho de maior valor.

Quando chegarmos ao último filho que foi movido teremos a posição de inserção do valor em aux.

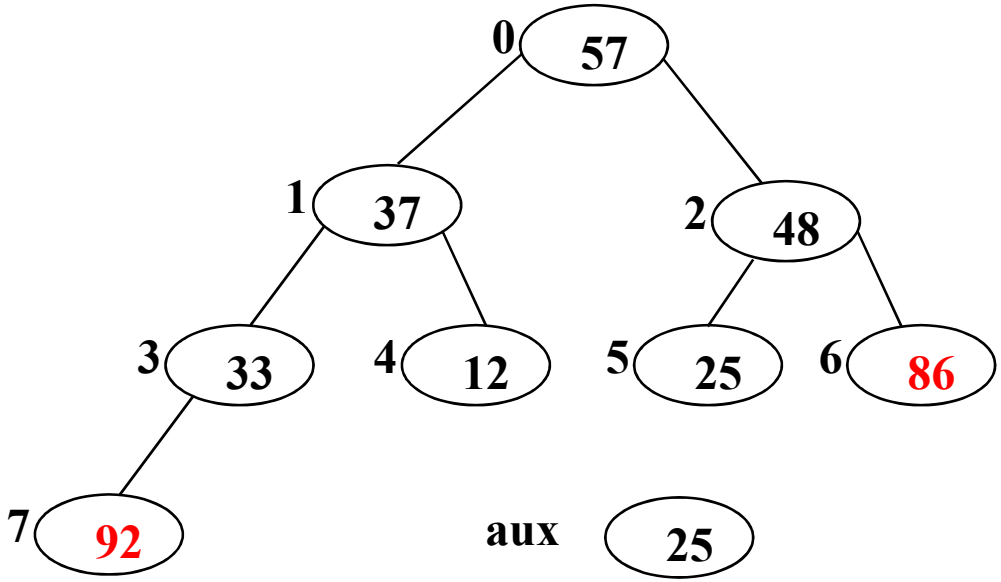


Após sua inserção teremos o heap do próximo slide.

Heap Sort

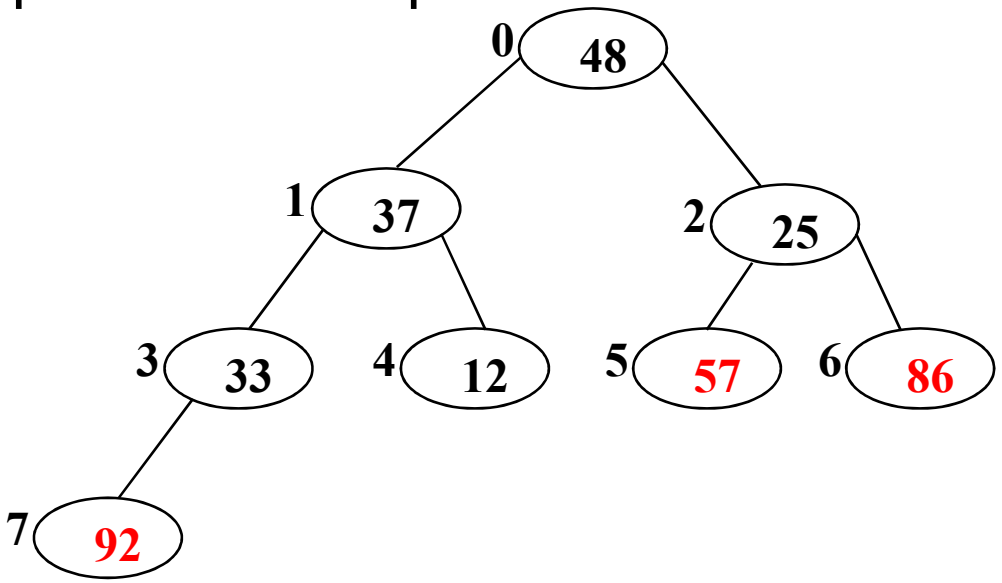


Aplicando este procedimento ao subvetor de n-1 teremos:



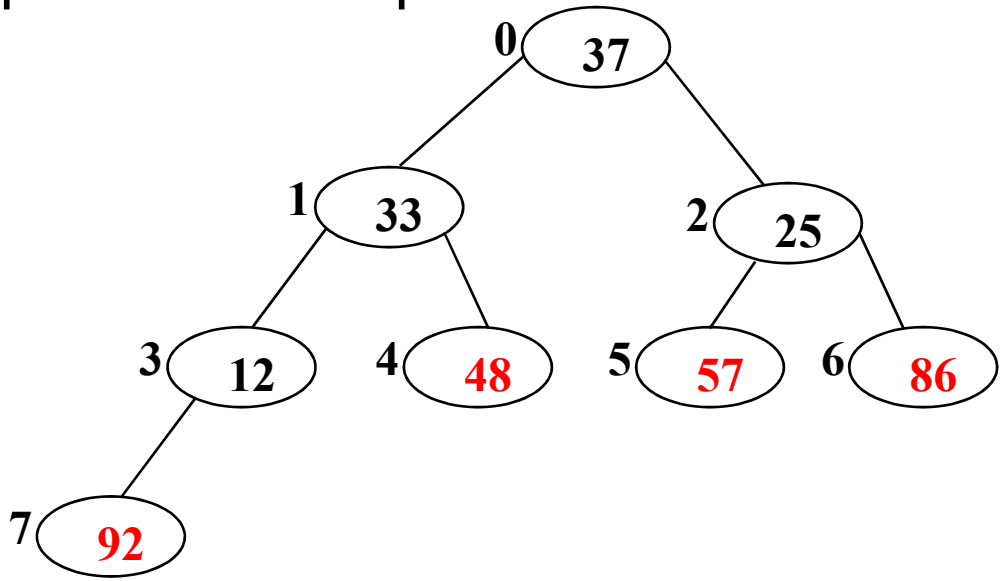
Heap Sort

Aplicando este procedimento ao subvetor de n-2 teremos:



aux 25

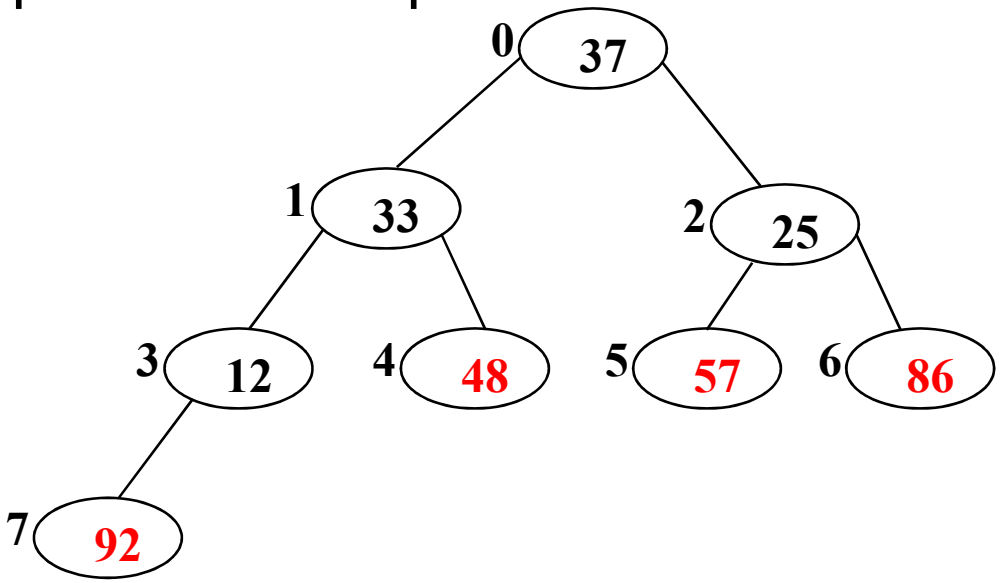
Aplicando este procedimento ao subvetor de n-3 teremos:



aux 12

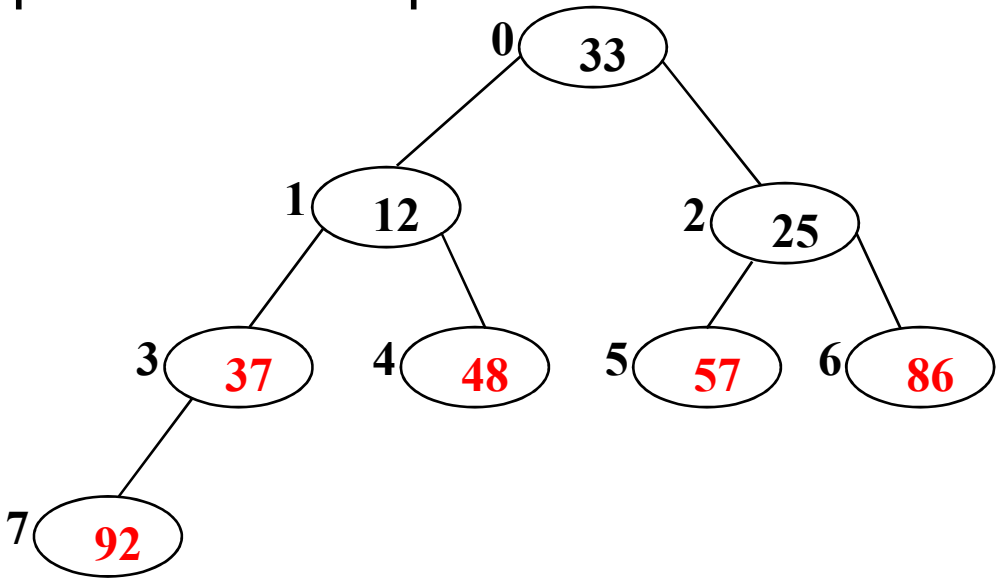
Heap Sort

Aplicando este procedimento ao subvetor de n-3 teremos:



aux 12

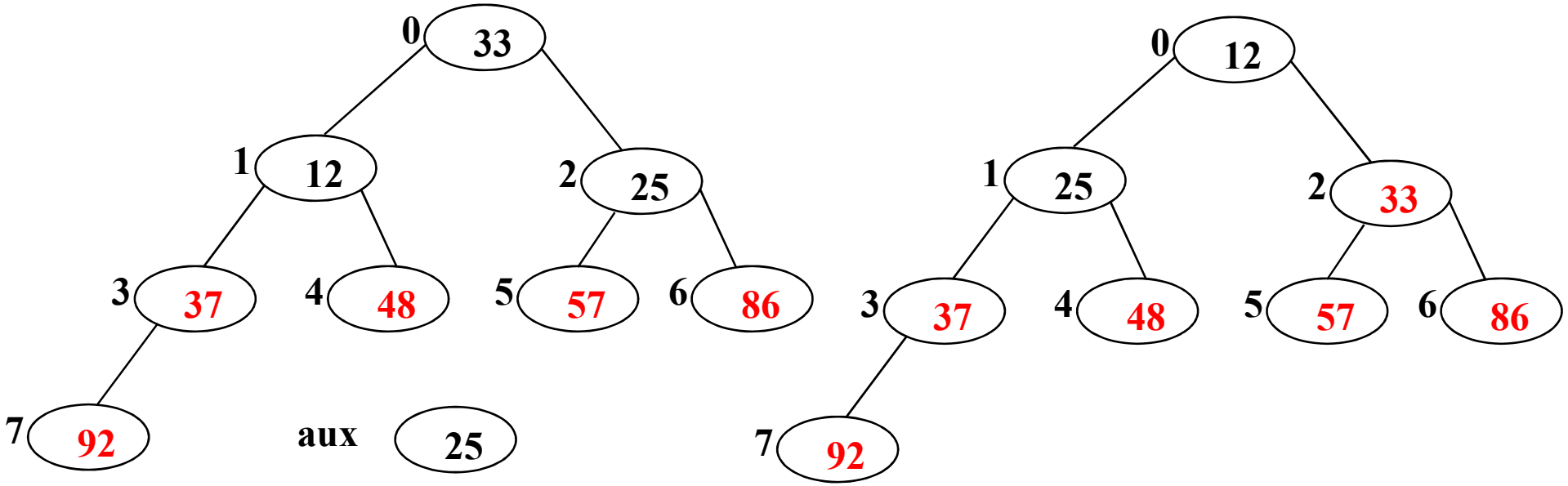
Aplicando este procedimento ao subvetor de n-4 teremos:



aux 12

Heap Sort

Aplicando este procedimento ao subvetor de n-5 teremos:



Na última fase, n-6, não movemos diretamente o filho, analisamos seu valor e apenas se este for menor que aux o movemos.

Com base no que foi discutido, codifique uma função, na linguagem C, que receba um vetor (de inteiros) e o número de elementos no mesmo e através do método *heap sort* ordene de forma crescente os elementos do vetor.