# Algoritmos e Complexidade
# LEI/LCC (2º ano)

6ª Ficha Prática

Ano Lectivo de 2010/11

O objectivo desta ficha é a análise dos algoritmos de *árvores binárias de pesquisa* e de *heaps*.

1. Uma *árvore binária de pesquisa* é uma árvore binária que verifica a seguinte propriedade: ou a árvore é vazia ou, não sendo vazia, todos os valores da sub-árvore esquerda são menores (ou iguais) do que o da raiz, e todos os valores da sub-árvore direita são maiores do que o que está na raiz. Além disso, as sub-árvores são também árvores binárias de pesquisa.

   Considere as seguintes declarações com que se pretende representar uma árvore binária de pesquisa com valores inteiros.

   ```
   typedef struct node {
    int elem;
    struct node *esq;
    struct node *dir;
   } Node, *Tree;

   Tree insert(Tree t, int x);      // insere um elemento na árvore
   Tree extractMin(Tree t, int *x); // extrai o menor elemento da árvore
   int exists(Tree t, int x);       // verifica se um elemento está na árvore
   int sum(Tree t, int *x);        // calcula a soma de todos os elementos da árvore
   ```

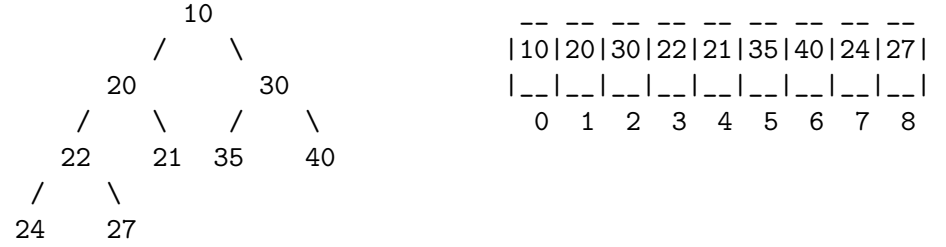
   - Implemente as funções indicadas de forma recursiva e analise o seu tempo de execução.
   - Re-implemente as funções indicadas de forma não recursiva e analise o seu tempo de execução.

2. Uma *(min)-heap* é uma árvore binária que verifica duas propriedades:

   - *shape property*: a árvore é completa, ou quasi-completa.
   - *(min)-heap property*: o conteúdo de cada nó é menor ou igual que o conteúdo dos seus descendentes (não havendo, no entanto, qualquer relação de ordem entre os conteúdos das duas sub-árvores de um mesmo nó).

   As heaps têm assim uma implementação muito vantajosa em array, em que a árvore vai sendo disposta por níveis ao longo do array (da esquerda para a direita). O acesso ao nó pai e aos nós filhos é feito de forma directa por aritmética de índices.

Exemplo de uma *min-heap* e sua representação em array:




Considere o seguinte *header file* para **min-heaps** de inteiros.

```
#define MAX        1000
#define PARENT(i)  (i-1)/2     // o indice do array começa em 0
#define LEFT(i)    2*i + 1
#define RIGHT(i)   2*i + 2

typedef int Elem;  // elementos da heap.

typedef struct {
 int   size;
 Elem  heap[MAX];
} Heap;

int insertHeap(Heap *h, Elem x);   // Insere um elemento na heap
void bubbleUp(Heap *h, int i);     // Função auxiliar de inserção: dada uma posição i
                                   // da heap com um novo valor que possivelmente viola
                                   // a propriedade da heap, propaga esse gradualmente
                                   // para cima.
int extractMin(Heap *h, Elem *x);  // Retira o mínimo da heap
void bubbleDown(Heap *h, int i);   // Função auxiliar de remoção: dada uma posição i
                                   // da heap com um valor que possivelmente viola
                                   // a propriedade da heap, propaga esse gradualmente
                                   // para baixo.
```

Apresente uma implementação das operações indicadas e analise o seu tempo de execução.

3. Uma *fila de prioridades* (FP) é uma estrutura de dados abstracta com duas operações:

   - *insert*, que insere um novo elemento numa FP;
   - *extractMin*, que extrai o menor elemento de uma FP, isto é, devolve o menor elemento e remove-o da estrutura.

   (a) Explique o funcionameno e efectue a análise assimptótica de pior caso do tempo de execução das duas operações sobre esta estrutura de dados no caso de a implementação se basear:

   i. Numa lista ligada;
   ii. Numa lista ligada ordenada;
   iii. Numa árvore binária de pesquisa;
   iv. Numa min-heap.

   (b) Efectue a mesma análise, agora relativamente a uma *sequência* de $N$ operações efectuadas a partir de uma fila de prioridades inicialmente vazia, correspondendo metade destas a cada uma das operações *insert* e *extractMin*. Baseado neste resultado, compare as 4 soluções acima quanto à sua eficiência global.