

**Estrutura de Dados e Algoritmos I**

**Universidade de Évora**

**Curso de Engenharia Informática**

****

**Projeto: HeapSort e MergeSort**

**Trabalho realizado no âmbito da**

**disciplina de Estrutura de Dados e Algoritmos I por:**

**Mafalda Rosa nº40021**

**Miguel Carvalho nº41136**

**Introdução**

Este trabalho foi realizado a pedido da professora Lígia Ferreira de Estrutura de Dados e Algoritmos 1, e tem como objetivo a criação por parte dos alunos uma forma de ordenar um array, com os algoritmos de ordenação lecionados nas aulas.

Sendo que para a realização deste trabalho era dado a opção de escolher dois dos algoritmos de ordenação dos que foram lecionados, sendo que escolhemos o HeapSort e o MergeSort.

**Descrição do Trabalho**

Existem vários algoritmos que ajudam a ordenar um array, como é exemplo o HeapSort e o MergeSort, que escolhemos para este trabalho.

No caso do HeapSort, este é um algoritmo de ordenação que pertence à classe dos algoritmos de seleção, como o SelectionSort. O HeapSort, funciona com construções de heaps, sendo que estas são representadas por árvores binárias balanceadas com algumas particularidades especiais, ou por um array. Visto que o HeapSort apresenta este tipo de ordenação os elementos à medida que vão sendo inseridos na estrutura, as heaps terão sempre o maior elemento na raiz (no caso das max heaps - onde os pais são sempre maiores que os filhos) ou o menor elemento na raiz (no caso das min heaps - onde os filhos são maiores que os pais), caso se queira uma ordenação crescente ou decrescente, respetivamente. Assim, aquando do fim das inserções do algoritmo, os elementos podem ser removidos de uma forma sucessiva da raiz, na ordem que se desejar, desde que se consiga manter a propriedade de max heap.

Considerando que as remoções na heap levam O(log(n)) e que para ordenar uma quantidade infinita de elementos seria necessário fazer infinitas remoções. Por conseguinte, temos que o tempo médio do HeapSort é O(n log(n)), onde n é quantidade de elementos. Assim, a sua complexidade quer no melhor dos casos quer no pior dos casos é idêntica, sendo esta O(n log (n)).

No caso do MergeSort, este é um algoritmo que se baseia na ordenação por divisão e conquista, como o QuickSort. No caso da divisão, divide-se o problema em vários problemas mais pequenos e resolvem-se estes subproblemas usando a recursividade. No que diz respeito à conquista, após a resolução da subdivisão do problema, unem-se os subproblemas que resulta na resolução do problema maior. Este algoritmo, dado que utiliza a recursividade, apresenta um alto consumo de memória e tempo de execução.

Existem 3 passos que são aplicáveis a este algoritmo, sendo que estes são:

1. Calcula-se o ponto médio do array, o que leva um tempo constante, isto é, O(1);
2. Utilizando a recursividade, resolvem-se os dois subproblemas que foram criados na Divisão, em que cada um apresenta metade do tamanho original;
3. Unem-se os dois subproblemas num único conjunto ordenado, o que leva a um tempo O(n).

Comparando com outros algoritmos de ordenação, o MergeSort apresenta uma complexidade idêntica à do QuickSort. Este algoritmo é mais rápido e eficiente no que diz respeito a uma grande quantidade de dados do que outros algoritmos de ordenação. A sua complexidade, no melhor dos casos é O(n log (n)), tal como é nos piores dos casos.