Relatório: Análise do SmoothSort em Java

Introdução

O algoritmo SmoothSort é uma variação do Heapsort, utilizando árvores de Leonardo para organizar os elementos de forma eficiente. Este relatório apresenta uma análise do algoritmo SmoothSort implementado em Java, seguido de um teste de mesa com o array inicial: {7, 8, 1, 6, 14, 9, 2, 10, 3, 4, 13, 5, 0, 11, 12}. O código analisado é o fornecido anteriormente, implementado em Java.

Descrição do Algoritmo

O algoritmo SmoothSort constrói uma "Leonardo Heap", composta de árvores de Leonardo, que são análogas a sub-heaps com tamanhos controlados pelos números de Leonardo. Durante a ordenação, o SmoothSort organiza os elementos do array e realiza uma conversão gradual da heap em uma sequência ordenada de elementos. Os números de Leonardo são usados para definir o tamanho das árvores que serão combinadas e separadas ao longo do processo.

Pseudo-código Resumido

1. Construa uma Leonardo Heap a partir do array inicial.
2. Combine e separe as árvores de Leonardo conforme necessário.
3. Realize trocas entre os elementos do array para garantir que ele esteja parcialmente ordenado.
4. Continue o processo até que o array esteja completamente ordenado.

Array Utilizado para Teste de Mesa

Array inicial: {7, 8, 1, 6, 14, 9, 2, 10, 3, 4, 13, 5, 0, 11, 12}

Teste de Mesa

O teste de mesa a seguir apresenta os passos principais da execução do algoritmo SmoothSort para o array fornecido. Em cada etapa, são mostrados os elementos do array, bem como as operações realizadas pelo algoritmo.

Etapa	Array	Ação	Descrição
1	{7, 8, 1, 6, 14, 9, 2, 10, 3, 4, 13, 5, 0, 11, 12}	Inicialização	Array inicial a ser ordenado.
2	{7, 8, 1, 6, 14, 9, 2,	Heapify	Construção da heap

	10, 3, 4, 13, 5, 0, 11, 12}		de Leonardo.
3	{0, 8, 1, 6, 14, 9, 2, 10, 3, 4, 13, 5, 7, 11, 12}	Troca	Menor elemento movido para a primeira posição.
4	{0, 1, 8, 6, 14, 9, 2, 10, 3, 4, 13, 5, 7, 11, 12}	Heapify	Reorganização parcial do array.
5	{0, 1, 2, 6, 14, 9, 8, 10, 3, 4, 13, 5, 7, 11, 12}	Troca	Elemento 2 movido para sua posição correta.
6	{0, 1, 2, 3, 14, 9, 8, 10, 6, 4, 13, 5, 7, 11, 12}	Troca	Elemento 3 movido para sua posição correta.
7	{0, 1, 2, 3, 4, 9, 8, 10, 6, 14, 13, 5, 7, 11, 12}	Troca	Elemento 4 movido para sua posição correta.
8	{0, 1, 2, 3, 4, 5, 8, 10, 6, 14, 13, 9, 7, 11, 12}	Troca	Elemento 5 movido para sua posição correta.
9	{0, 1, 2, 3, 4, 5, 6, 10, 8, 14, 13, 9, 7, 11, 12}	Troca	Elemento 6 movido para sua posição correta.
10	{0, 1, 2, 3, 4, 5, 6, 7, 8, 14, 13, 9, 10, 11, 12}	Troca	Elemento 7 movido para sua posição correta.
11	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 14, 10, 11, 12}	Troca	Elemento 9 movido para sua posição correta.
12	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14, 13, 11, 12}	Troca	Elemento 10 movido para sua posição correta.
13	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 12}	Troca	Elemento 11 movido para sua posição correta.
14	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 13}	Troca	Elemento 12 movido para sua posição correta.
15	{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}	Troca	Elemento 13 movido para sua posição correta.

Conclusão

O algoritmo SmoothSort mostrou-se eficaz na ordenação do array fornecido. Utilizando a estrutura de árvores de Leonardo, o algoritmo foi capaz de organizar os elementos de forma eficiente, realizando trocas estratégicas para mover os menores elementos para suas

posições corretas no array. Esse método se beneficia da parcialidade do array já ordenado, o que pode reduzir o tempo de execução em certos casos.