SCC0201 - Introdução à Ciência de Computação II

Lista de exercícios para aula

Professor: Diego Raphael Amancio

Estagiárias PAE:
Xiomara Sulvey Quispe Chacon
Laura Vanessa Cruz Quispe

---

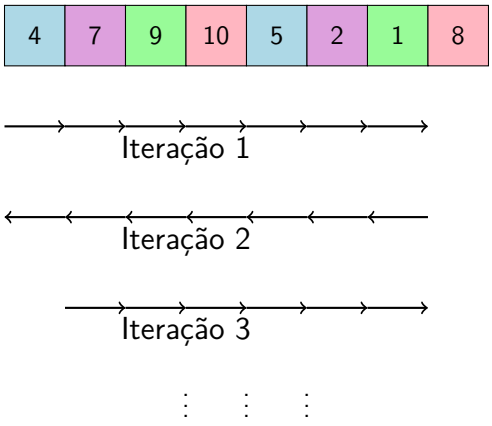
1 ***Cocktail Shaker Sort***

Desenvolva uma variação do algoritmo Bubble Sort chamada Cocktail Shaker Sort, projetada por Donal Knuth. Nesta variação, a direção da "subida" e "descida" dos elementos é alternada a cada iteração. Em uma iteração, o menor elemento é "subido" (bubbled up); na próxima, o maior é "descido" (bubbled down); na subsequente, o segundo menor é "subido"; e assim por diante, alternando.

Implemente este novo algoritmo e explore sua complexidade.

Fases

- **"Bubbled up"** (subido): O algoritmo começa da **esquerda para a direita** do array. A cada iteração, compara-se um elemento com seu próximo à direita. Se o elemento atual for maior que o próximo, eles são trocados. Isso faz com que o menor elemento "suba" em direção à posição inicial do array.
- **"Bubbled down"** (descido): O algoritmo começa da **direita para a esquerda** do array. A cada iteração, compara-se um elemento com seu anterior à esquerda. Se o elemento atual for menor que o anterior, eles são trocados. Isso faz com que o maior elemento "desça" em direção à posição final do array.



Observação: *Após a fase de "subida", o menor elemento estará na posição inicial. Para encontrar o segundo menor elemento, o algoritmo repete a fase de "subida", ignorando a posição inicial onde já está o menor elemento.*

Entrada

A primeira linha contém um inteiro N, representando o número de elementos no array. A segunda linha contém N inteiros separados por espaço, indicando os elementos do array.

Saída

Após cada iteração do algoritmo, imprima o estado atual do array.

Exemplo:

Entrada	Saída
8 4 7 9 10 5 2 1 8	4 7 9 5 2 1 8 10 1 4 7 9 5 2 8 10 1 4 7 5 2 8 9 10 1 2 4 7 5 8 9 10 1 2 4 5 7 8 9 10 1 2 4 5 7 8 9 10

2 ***Ternary Heap Sort***

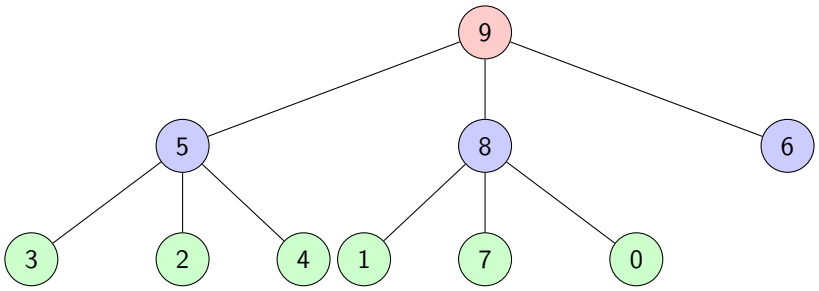
Defina uma árvore ternária quase completa, na qual cada nó tem, no máximo, três filhos, e na qual os nós podem ser numerados de 0 até $n-1$, de modo que os filhos de `node[i]` sejam `node[3i + 1]`, `node[3i + 2]` e `node[3i + 3]`. Defina um heap ternário como uma árvore ternária quase completa na qual o conteúdo de cada nó é maior ou igual ao conteúdo de todos os seus filhos.

Implemente o algoritmo de ordenação semelhante ao Heap Sort, mas usando um heap ternário.

Fases

- **Construção do Heap ternário** Transforme o array de entrada em um heap ternário, garantindo que cada nó seja maior ou igual a seus filhos.

  Inicie da última posição e percorra até a primeira posição do array. Para cada elemento, compare-o com seus filhos e ajuste sua posição, se necessário, para manter a propriedade do heap ternário.



- **Ordenação** Remova o maior elemento (raiz do heap) e o coloque no final da lista ordenada. Reorganize o heap para manter a propriedade do heap. Repita esse processo até que o heap esteja vazio.

Entrada

A primeira linha contém um inteiro M que indica o número de elementos no array a ser ordenado. A segunda linha contém os M inteiros separados por espaço.

Saída

Comece imprimindo o array inicial, seguido pelo estado do array após a construção do heap ternário. Após cada iteração do algoritmo, imprima o estado atual do array. Cada par de arrays separados por espaço representa uma iteração do algoritmo:

- A primeira linha de cada par corresponde ao array após remover o maior elemento e colocá-lo no final da lista ordenada, seguido do elemento removido (Ex. `0 4 8 6 3 2 5 1 7 - 9`)
- A segunda linha corresponde ao estado do array após reorganizar o heap para manter a propriedade do heap ternário.
- A última linha representa o array ordenado.

Exemplo:

Entrada	Saída
10 7 4 9 6 3 2 5 1 8 0	7 4 9 6 3 2 5 1 8 0 9 5 8 6 3 2 4 1 7 0 0 5 8 6 3 2 4 1 7 - 9 8 5 7 6 3 2 4 1 0 0 5 7 6 3 2 4 1 - 8 7 5 1 6 3 2 4 0 0 5 1 6 3 2 4 - 7 6 5 1 0 3 2 4 4 5 1 0 3 2 - 6 5 4 1 0 3 2 2 4 1 0 3 - 5 4 3 1 0 2 2 3 1 0 - 4 3 2 1 0 0 2 1 - 3 2 0 1 1 0 - 2 1 0 0 - 1 0 1 2 3 4 5 6 7 8 9