Projeto e Análise de Algoritmos

Algoritmos de Ordenação

Matheus Gabriel

Agosto de 2024

1 Merge Sort

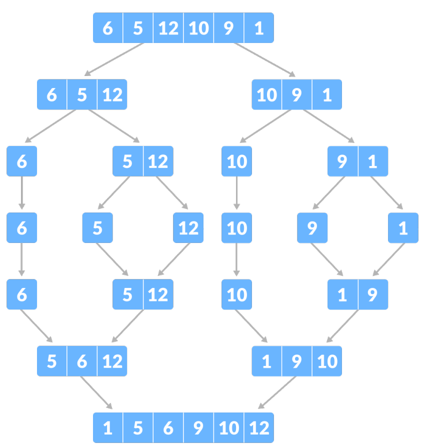
1.1 Explicação

O Merge Sort é um algoritmo de ordenação baseado na técnica de **divisão e conquista**. O algoritmo segue os seguintes passos:

1. **Divisão:** Divide o array (ou lista) em duas metades.
2. **Conquista:** Ordena cada metade recursivamente.
3. **Combinação:** Mescla as duas metades ordenadas para produzir o array ordenado final.

A eficiência do Merge Sort é garantida pela sua abordagem recursiva e pela técnica de mesclagem, que resulta numa complexidade de tempo de $\Theta(n \log n)$ tanto no pior caso quanto no melhor caso. Especificamente, a complexidade no pior caso é $\Omega(n \log n)$, o que significa que o Merge Sort é garantidamente eficiente na ordenação de qualquer conjunto de dados.

1.2 Exemplo visual



Passos do Merge Sort. Fonte: Programiz.

1.3 Código em C

1.3.1 Implementação

```c
#include <stdio.h>
#include <stdlib.h>

void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int* L = (int*)malloc(n1 * sizeof(int));
    int* R = (int*)malloc(n2 * sizeof(int));

    for (int i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
        } else {
            arr[k++] = R[j++];
        }
    }

    while (i < n1) {
        arr[k++] = L[i++];
    }

    while (j < n2) {
        arr[k++] = R[j++];
    }

    free(L);
    free(R);
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSort(arr, l, m);
```

```c
        mergeSort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr) / sizeof(arr[0]);

    printf("Array original:\n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nArray ordenado:\n");
    printArray(arr, arr_size);
    return 0;
}
```

1.4 Código em Common Lisp

1.4.1 Implementação

```lisp
(defun merge (left right)
  (cond
    ((null left) right)
    ((null right) left)
    ((<= (car left) (car right))
     (cons (car left) (merge (cdr left) right)))
    (t (cons (car right) (merge left (cdr right))))))

(defun merge-sort (list)
  (if (or (null list) (null (cdr list)))
      list
      (let* ((mid (/ (length list) 2))
             (left (subseq list 0 mid))
             (right (subseq list mid)))
        (merge (merge-sort left) (merge-sort right)))))

;; Exemplo de uso
```

```
(let ((unsorted-list '(12 11 13 5 6 7)))
  (format t "Lista original: ~a~%" unsorted-list)
  (format t "Lista ordenada: ~a~%" (merge-sort unsorted-list)))
```

2 Encontrando a fórmula fechada

Seção Importante

Essa é uma das partes mais importantes da matéria!

2.1 Primeiro Exemplo (Mais explicativo)

Vamos resolver a recorrência dada por:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$
$$T(n) = 2T\left(\frac{n}{2}\right) + n$$
$$T(1) = 1$$

Solução da Recorrência

Para resolver a recorrência $T(n) = 2T\left(\frac{n}{2}\right) + n$, utilizaremos o método de expansão:

1. **Expandindo a Recorrência:**

Começamos substituindo $T\left(\frac{n}{2}\right)$ na expressão de $T(n)$:

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{4}\right) + \frac{n}{2}$$
$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n$$
$$T(n) = 2^2T\left(\frac{n}{4}\right) + 2\cdot\frac{n}{2} + n$$
$$T(n) = 2^2T\left(\frac{n}{4}\right) + n + n$$

Continuamos expandindo para o próximo nível:

$$T\left(\frac{n}{4}\right) = 2T\left(\frac{n}{8}\right) + \frac{n}{4}$$
$$T(n) = 2^2\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + n + n$$
$$T(n) = 2^3T\left(\frac{n}{8}\right) + 2^2\cdot\frac{n}{4} + n + n$$
$$T(n) = 2^3T\left(\frac{n}{8}\right) + n + n + n$$

2. **Generalizando:**
Observamos que após i expansões, temos:

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + \sum_{k=0}^{i-1} 2^k \cdot \frac{n}{2^k}$$

O somatório pode ser simplificado para:

$$\sum_{k=0}^{i-1} n = n \cdot i$$

Então:

$$T(n) = 2^i T\left(\frac{n}{2^i}\right) + n \cdot i$$

3. **Determinação do Valor de i:**
A recursão termina quando $\frac{n}{2^i} = 1$, ou seja:

$$\begin{aligned}\frac{n}{2^i} &= 1 \\ 2^i &= n \\ i &= \log_2 n\end{aligned}$$

4. **Substituindo i na Fórmula Geral:**
Quando $i = \log_2 n$, temos:

$$\begin{aligned}T(n) &= 2^{\log_2 n} T(1) + n \cdot \log_2 n \\ T(n) &= n \cdot 1 + n \cdot \log_2 n \\ T(n) &= n + n \log_2 n\end{aligned}$$

5. **Complexidade Assintótica:**
Combinando os termos, obtemos:

$$T(n) = \Theta(n \log_2 n)$$

Portanto, a fórmula fechada para a recorrência é:

$$\boxed{T(n) = \Theta(n \log n)}$$

2.2 Segundo Exemplo

Vamos resolver a seguinte recorrência:

$$T(n) = T\left(\frac{n}{2}\right) + 1$$
$$\text{com } T(1) = 1$$

Solução da Recorrência

Para resolver a recorrência $T(n) = T\left(\frac{n}{2}\right) + 1$, utilizaremos o método de expansão:

1. **Expandindo a Recorrência:**

Começamos substituindo $T\left(\frac{n}{2}\right)$ na expressão de $T(n)$:

$$T\left(\frac{n}{2}\right) = T\left(\frac{n}{2^2}\right) + 1$$
$$T(n) = T\left(\frac{n}{2^2}\right) + 1 + 1$$
$$T(n) = T\left(\frac{n}{2^2}\right) + 2$$

Continuamos expandindo para o próximo nível:

$$T\left(\frac{n}{2^2}\right) = T\left(\frac{n}{2^3}\right) + 1$$
$$T(n) = T\left(\frac{n}{2^3}\right) + 1 + 1 + 1$$
$$T(n) = T\left(\frac{n}{2^3}\right) + 3$$

2. **Generalizando:**

Observamos que após i expansões, temos:

$$T(n) = T\left(\frac{n}{2^i}\right) + i$$

A recursão termina quando $\frac{n}{2^i} = 1$, ou seja:

$$\frac{n}{2^i} = 1$$
$$2^i = n$$
$$i = \log_2 n$$

3. **Substituindo i na Fórmula Geral:**

Quando $i = \log_2 n$, temos:

$$
\begin{aligned}
T(n) &= T(1) + \log_2 n \\
T(n) &= 1 + \log_2 n
\end{aligned}
$$

4. **Complexidade Assintótica:**
O termo constante 1 pode ser desconsiderado na notação assintótica, então obtemos:

$$T(n) = \Theta(\log n)$$

Portanto, a fórmula fechada para a recorrência é:

$$\boxed{T(n) = \Theta(\log n)}$$