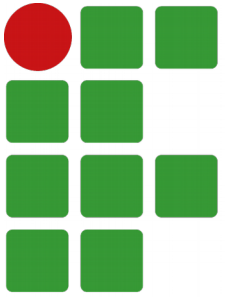
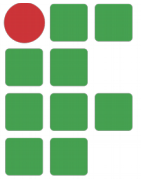


INSTITUTO FEDERAL
Ceará

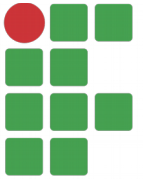
Ordenação
(Divisão e Conquista - MergeSort)

Definição



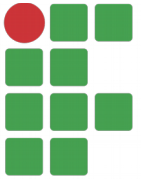
- Mergesort é um algoritmo de ordenação recursivo
- Ele recursivamente ordena as duas metades do vetor
- Usa a estratégia de **divisão e conquista**
- Mergesort é um algoritmo eficiente

Divisão e Conquista



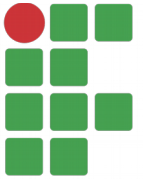
- Método de Divisão e Conquista
  - **Divisão**: Divida o problema em duas ou mais partes, criando subproblemas menores.
  - **Conquista**: Os subproblemas são resolvidos recursivamente usando divisão e conquista. Caso os subproblemas sejam suficientemente pequenos resolva-os de forma direta.
  - **Combina**: Tome cada uma das partes e junte-as todas de forma a resolver o problema original.

Algoritmo Mergesort



- Caso o tamanho do vetor seja maior que 1
  - 1. divida o vetor no meio
  - 2. ordene a primeira metade recursivamente
  - 3. ordene a segunda metade recursivamente
  - 4. intercale as duas metades
- Senão devolva o elemento

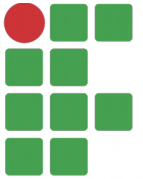
Código do Mergesort



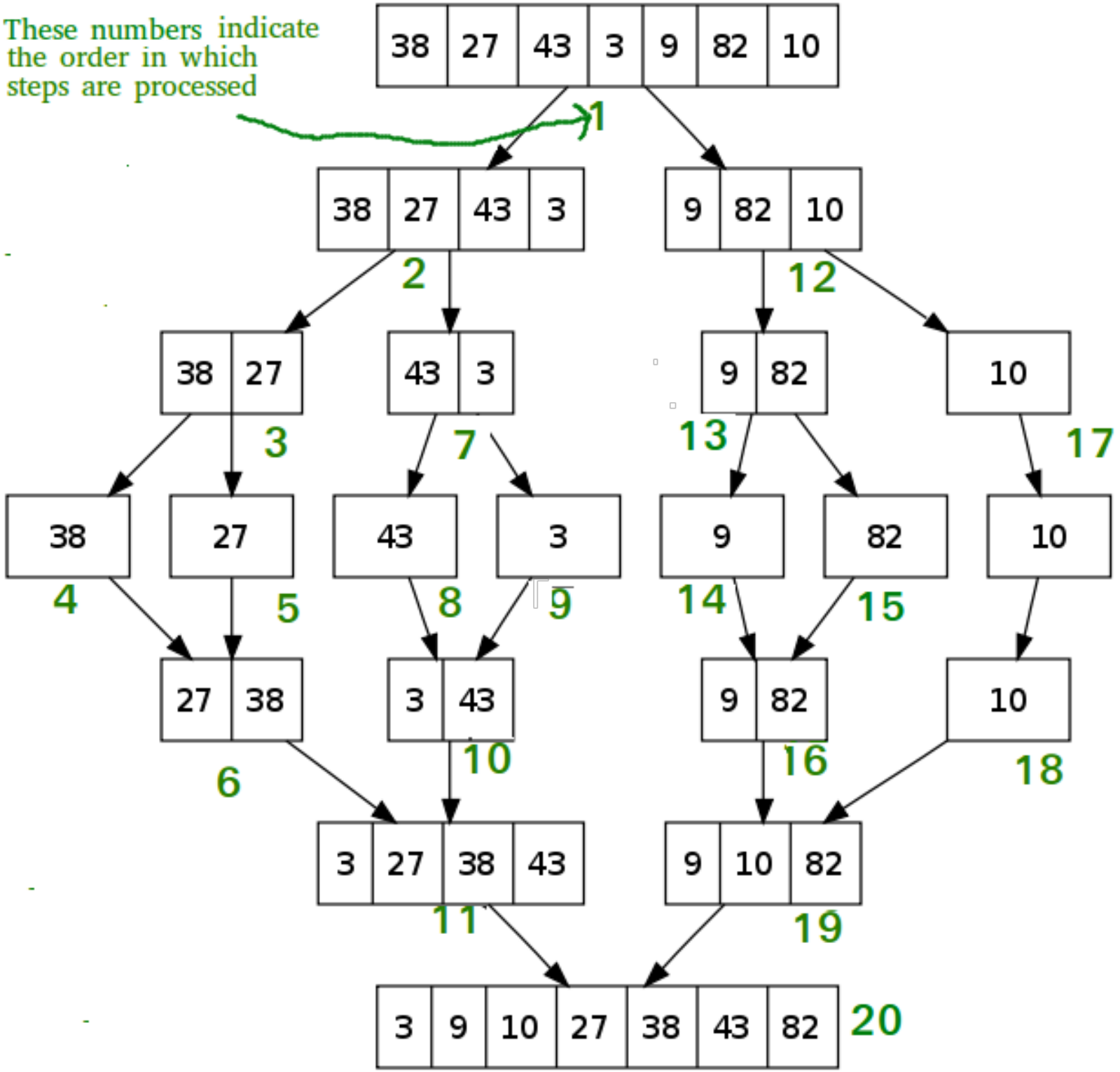
```
MergeSort (A,p,r)
if p < r
    q = (p + q)/2
    Mergesort(A,p,q)
    Mergesort(A,q + 1,r)
    Merge(A,p,q,r)
```

Chamada Inicial: Mergesort(A, 1,n)

merge/intercalação



- A intercalação de dois vetores ordenados:
  - Uma variável em cada vetor **indica o próximo elemento** a ser inserido a lista intercalada
  - Enquanto ambas os vetores tiverem elementos
  - Coloque o **menor entre os dois elemento** indicados no vetor intercalado e **incremente** índice respectivo
  - Quando um dois vetores não tiver mais elementos, concatene o outro no final do vetor intercalado.


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
9 82 10
2
12
38 27
43 3
9 82
10
3
7
13
17
38
27
43
3
9
82
10
4
5
8
9
14
15
27 38
3 43
9 82
10
6
10
16
18
3 27 38 43
9 10 82
11
19
3 9 10 27 38 43 82
20

MergeSort (A,p,r)

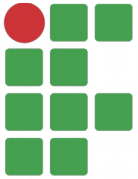
if p < r

q = (p + q)/2

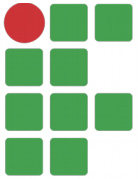
Mergesort(A,p,q)

Mergesort(A,q + 1,r)

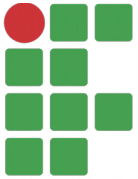
Merge(A,p,q,r)



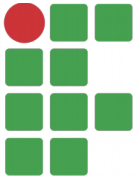
```
void mergeSort (int *v, int inicio, int fim) {
    if (inicio < fim) {
        int meio =(inicio+fim)/2;
        mergeSort(v, inicio, meio);
        mergeSort(v, meio+1, fim);
        merge(v, inicio, meio, fim);
    }
}
```



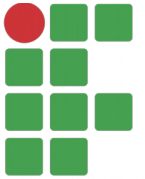
```c
void merge(int *v, int inicio, int meio, int fim) {
    int n1 = meio-inicio+1; //tamanho subarray 1 => do inicio até o meio (inclusive)
    int n2 = fim-meio; //tamanho subarray 2 => do meio+1 até o fim
    int sa1[n1];
    int sa2[n2];
    //printf("inicio=%d meio=%d fim=%d\n", inicio, meio, fim);
    //printf("n1=%d n2=%d\n", n1, n2);

    //preencher os sub-arrays
    for (int i=0; i < n1; i++) {
        sa1[i] = v[inicio+i]; }
    for (int j=0; j<n2; j++) {
        sa2[j] = v[meio+1+j]; }
    int i=0; //indice sub-array 1
    int j=0; //indice sub-array 2
    int k=inicio; //indice merged sub-array = INICIO => vai manipular o vetor original
```



```
//compara os 2 sub-arrays entre si e descobre quem possui o menor número
//insere o menor número na posição k do vetor original
while (i < n1 && j < n2) {
    if (sa1[i] <= sa2[j]) {
        v[k] = sa1[i];
        i++;
    } else {
        v[k] = sa2[j];
        j++;}
    k++;}
```



```
//add os números restantes de sa1 e sa2
while (i < n1) {
    v[k] = sa1[i];
    i++;
    k++;
}
while (j < n2) {
    v[k] = sa2[j];
    j++;
    k++;
}
```



```c
void merge(int *v, int inicio, int meio, int fim) {
    int n1 = meio-inicio+1; //tamanho subarray 1 => do inicio até o meio (inclusive)
    int n2 = fim-meio; //tamanho subarray 2 => do meio+1 até o fim
    int sa1[n1];
    int sa2[n2];
    //printf("inicio=%d meio=%d fim=%d\n", inicio, meio, fim);
    //printf("n1=%d n2=%d\n", n1, n2);

    //preencher os sub-arrays
    for (int i=0; i < n1; i++) {
        sa1[i] = v[inicio+i]; }
    for (int j=0; j<n2; j++) {
        sa2[j] = v[meio+1+j]; }
    int i=0; //indice sub-array 1
    int j=0; //indice sub-array 2
    int k=inicio; //indice merged sub-array = INICIO => vai manipular o vetor original

    //compara os 2 sub-arrays entre si e descobre quem possui o menor número
    //insere o menor número na posição k do vetor original
    while (i < n1 && j < n2) {
        if (sa1[i] <= sa2[j]) {
            v[k] = sa1[i];
            i++;
        } else {
            v[k] = sa2[j];
            j++;}
        k++;}

    //add os números restantes de sa1 e sa2
    while (i < n1) {
        v[k] = sa1[i];
        i++;
        k++;
    }
    while (j < n2) {
        v[k] = sa2[j];
        j++;
        k++;
    }
}
```