

Ajeitasort e Juntasort

O objetivo dessa aula é exercitar:

1. a manipulação de dois apontadores (ajeita)e três apontadores(merge).
2. a construção de funções e a passagem de parâmetros (ajeita) e (merge).
3. a combinação de funções (ajeitasort) e (juntasort). No juntasort, combinamos as funções ajeitasort e merge.

Ajeitando Lista

Considere o seguinte problema. Você recebe uma lista L de tamanho n em que a lista $L[0..n-2]$ está ordenada e o elemento $L[n-1]$ está fora do lugar. Por exemplo, a lista abaixo tem essa propriedade:

	0	1	2	3	4
L	1	5	7	8	3

Queremos modificar essa lista para obter:

	0	1	2	3	4
L	1	3	5	7	8

Vamos realizar essa tarefa da seguinte maneira. A variável x vai guardar o valor de $L[n-1]$ e o apontador j vai procurar a posição correta do elemento. Durante o laço, vamos descobrindo o valores que serão empurrados para o fim da lista. Os elementos que serão empurrados serão todos que são maiores que x .

A ideia seria posicionar o apontador j na última posição e toda vez que o $L[j-1]$ for maior que x , $L[j-1]$ será copiado para $L[j]$ e o apontador j retrocede uma unidade.

```
1  #include <stdio.h>
2
3  void imprime(int L[], int n){
4      printf("[");
5      for(int i = 0; i < n; i++){
6          printf(" %d", L[i]);
7      }
8      printf("]\n");
9  }
10
11 void ajeita(int L[], int n);
12 int main()
13 {
14     int L[] = {1,3,7,9,4};
15     int n = sizeof(L)/sizeof(int);
16     ajeita(L, n);
17     return 0;
18 }
```

```

1 void ajeita(int L[], int n);
2     int j, x;
3     x = L[n-1];
4     j = n-1;
5     imprime(L, n);
6     while(j > 0 && L[j-1] > x){
7         L[j] = L[j-1];
8         printf("j %d:", j);
9         imprime(L, n);
10        j--;
11
12    }
13    L[j] = x;
14    imprime(L, n);
15 }

```

A saída desse programa será a seguinte:

```

1 [ 1 3 7 9 4]
2 j 4:[ 1 3 7 9 9]
3 j 3:[ 1 3 7 7 9]
4 [ 1 3 4 7 9]

```

Observe que durante o processo de ajeitar a lista, alguns números aparecem duplicados que são os números que estão sendo "empurrados".

Ajeita Sort

A ideia do `ajeita_sort` é aumentando o tamanho da lista em cada passo. Inicialmente, a lista $L[0 \dots 0]$ de tamanho 1 está ordenada, então podemos aplicar o algoritmo `ajeita` na lista $L[0 \dots 1]$ de tamanho 2. Agora, teremos a $L[0 \dots 1]$ ordenada. O processo pode continuar até chegamos na lista $L[0 \dots n-1]$.

```
1 void ajeita_sort(int L[], int n){
2     int i;
3     for( i = 2; i <=n; i++){
4         printf("i %d\n", i);
5         ajeita(L, i);
6     }
7 }
8 int main()
9 {
10     int L[] = {8,5,3,2,1,10,9};
11     int n = sizeof(L)/sizeof(int);
12     ajeita_sort(L, n);
13     imprime(L, n);
14     return 0;
15 }
16
```

```
1 i 2
2 [ 8 5]
3 j 1:[ 8 8]
4 [ 5 8]
5 i 3
6 [ 5 8 3]
7 j 2:[ 5 8 8]
8 j 1:[ 5 5 8]
9 [ 3 5 8]
10 i 4
11 [ 3 5 8 2]
12 j 3:[ 3 5 8 8]
13 j 2:[ 3 5 5 8]
14 j 1:[ 3 3 5 8]
15 [ 2 3 5 8]
16 i 5
17 [ 2 3 5 8 1]
18 j 4:[ 2 3 5 8 8]
19 j 3:[ 2 3 5 5 8]
20 j 2:[ 2 3 3 5 8]
21 j 1:[ 2 2 3 5 8]
22 [ 1 2 3 5 8]
23 i 6
24 [ 1 2 3 5 8 10]
25 [ 1 2 3 5 8 10]
26 i 7
27 [ 1 2 3 5 8 10 9]
28 j 6:[ 1 2 3 5 8 10 10]
29 [ 1 2 3 5 8 9 10]
```

O algoritmo acima tem um comportamento quadrático com relação ao número de instruções.

Juntando Listas Ordenadas

Considere o seguinte problema:

	0	1	2	3	4	6	7	8	9	10
L	1	3	5	7	8	2	6	8	9	11

A lista é composta por duas sublistas ordenadas $L[0..4]$ e $L[5..10]$. A entrada do problema é composta pela lista L e três inteiros: início, meio e fim. A lista $L[\text{início} \dots \text{meio}-1]$ está ordenada e $L[\text{meio} \dots \text{fim}]$ está ordenada.

Queremos produzir uma lista ordenada com todos os números das duas sublistas.

Note que os seguintes problemas são fáceis de resolver quando resolvidos sequencialmente:

1. Encontrar o menor.
2. Encontrar o segundo menor.
3. Encontrar o terceiro menor.
4. assim por diante

Neste problema, vamos utilizar uma lista auxiliar A para guardar o resultados dos problemas resolvidos.

A ideia geral é usar três apontadores: i apontando para o primeiro elemento da primeira metade, j apontando para o primeiro elemento da segunda metade e k apontando para a primeira posição da lista auxiliar.

Note que o menor elemento será o menor entre os elementos apontado por i e j . O apontador que aponta para o menor pode ser avançado e podemos encontrar facilmente o segundo menor.

```
1 void merge(int L[], int inicio, int meio, int fim){
2     int n = fim - inicio + 1;
3     int A[n];
4     int i, j, k;
5     i = inicio;
6     j = meio;
7     k = 0;
8     while(i < meio && j <= fim){
9         if(L[i] < L[j]){ A[k] = L[i]; k++; i++;}
10        else{ A[k] = L[j]; k++; j++;}
11    }
12    while( i < meio){ A[k] = L[i]; i++; k++;}
13    while( j <= fim){ A[k] = L[j]; j++; k++;}
14    i = inicio;
15    k = 0;
16    while( i <= fim){ L[i] = A[k]; i++; k++;}
17
18 }
```

Juntasort

Primeiramente, vamos modificar a função `ajeita` e `ajeitasort` para trabalhar com sublistas de L .

```
1 void ajeita(int L[], int inicio, int fim){
2     int j, x;
3     x = L[fim];
4     j = fim;
5     while(j > inicio && L[j-1] > x){
6         L[j] = L[j-1];
7         j--;
8     }
9     L[j] = x;
10 }
11 void ajeitasort(int L[], int inicio, int fim){
12     int i;
13     for( i = inicio+1; i <= fim; i++){
14         ajeita(L, inicio, i);
15     }
16 }
```

Agora, estamos pronto para usar definir o `juntasort`. Vamos utilizar o `ajeitasort` para ordenar duas partes da lista para em seguida utilizar a função `merge`.

```
1 void juntasort(int L[], int n){
2     int meio = n/2;
3     ajeitasort(L, 0, meio-1);
4     ajeitasort(L, meio, n-1);
5     printf("Entrada merge\n");
6     imprime(L, n);
7     merge(L, 0, meio, n-1);
8     printf("Saida merge\n");
9     imprime(L, n);
10 }
11
12 int main()
13 {
14     int L[] = {1,5,3,2,4,6,8,7};
15     int n = sizeof(L)/sizeof(int);
16     juntasort(L, n);
17     return 0;
18 }
```