MCTA028-15: Programação Estruturada

Aula 11: Ordenação (Segunda Parte)

Wagner Tanaka Botelho
wagner.tanaka@ufabc.edu.br / wagtanaka@gmail.com
Universidade Federal do ABC (UFABC)
Centro de Matemática, Computação e Cognição (CMCC)

Quick Sort

Quick Sort

- Também conhecido como ordenação por “partição”;
- Algoritmo recursivo que usa a ideia de dividir para conquistar para ORDENAR os dados de um *array*;
- A ideia é REARRANJAR um *array* de modo que:
  - Os valores **MENORES** do que certo valor, chamado de PIVÔ, fiquem na parte ESQUERDA do *array*;
  - Os valores **MAIORE**S do que o PIVÔ, fiquem na parte DIREITA.

pivô = 23

0	1	2	3	4	5	6
23	4	67	-8	90	54	21

while_1(esq<dir)

esq						dir	
23	4	67	-8	90	54	21	V[esq]<=pivô (V) esq++
	esq					dir	
23	4	67	-8	90	54	21	
		esq				dir	
23	4	67	-8	90	54	21	V[esq]<=pivô (F)
		esq				dir	
23	4	67	-8	90	54	21	V[dir]>pivô (F)
		esq				dir	
23	4	67	-8	90	54	21	esq<dir (V) troca
		esq				dir	
23	4	21	-8	90	54	67	

while_2(esq<dir)

		esq				dir	
23	4	21	-8	90	54	67	V[esq]<=pivô (V) esq++
			esq			dir	
23	4	21	-8	90	54	67	
				esq		dir	
23	4	21	-8	90	54	67	V[esq]<=pivô (F)

				esq		dir	
23	4	21	-8	90	54	67	V[dir]>pivô (V) dir--
				esq		dir	
23	4	21	-8	90	54	67	V[dir]>pivô (V) dir--
				esq	dir		
23	4	21	-8	90	54	67	V[dir]>pivô (V) dir--
			dir	esq			
23	4	21	-8	90	54	67	
			dir	esq			
23	4	21	-8	90	54	67	esq<dir F)

V[inicio]			dir	esq			
-8	4	21	-8	90	54	67	V[inicio]=v[dir]
			dir	esq			
-8	4	21	23	90	54	67	V[dir]=pivo

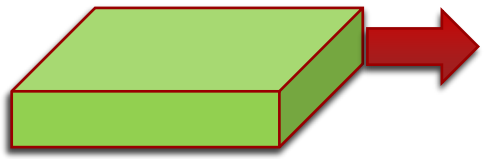
***Array* rearranjado!**

```
25  void quickSort(int *V, int inicio, int fim){
26      int pivo=0;
27      if(fim>inicio){
28          pivo = particiona(V, inicio, fim);
29          quickSort(V, inicio, pivo-1);
30          quickSort(V, pivo+1, fim);
31      }
32  }
33
34  void main(){
35      int V[4] = {23, 4, 67, -8};
36      quickSort(V, 0, 3);
37  }
```

0	1	2	3
23	4	67	-8

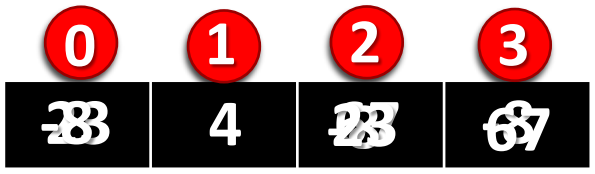
Linha	V	inicio	fim	pivo
36	Execução 1: quickSort(V, 0, 3)			
25	(23, 4, 67, -8)	0	3	
28	particiona(V, 0, 3)			

Pilha de Recursão



Execução 1: quickSort(V, 0, 3)

```
1  int particiona(int *V, int inicio, int final1){
2      int esq, dir, pivo, aux;
3      esq = inicio;
4      dir = final1;
5      pivo = V[inicio];
6
7      while(esq < dir){
8          while(V[esq] <= pivo){
9              esq++;
10         }
11         while(V[dir] > pivo){
12             dir--;
13         }
14         if(esq < dir){
15             aux = V[esq];
16             V[esq] = V[dir];
17             V[dir] = aux;
18         }
19     }
20     V[inicio] = V[dir];
21     V[dir] = pivo;
22     return dir;
23 }
```



Linha	Inicio	final1	esq	dir	pivo	aux	V	fim
1	0	3						
3			0					
4				3				
5					23			
9			1					
9			2					
15						67		
16							[2]=-8	
17							[3]=67	
9			3					
12				2				
20							[0]=-8	
21							[2]=23	
22				2				

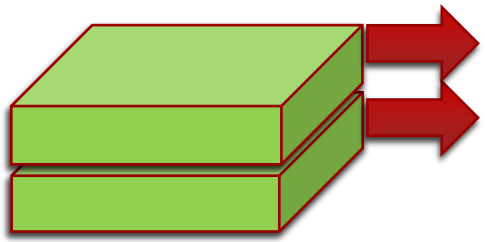
```
25  void quickSort(int *V, int inicio, int fim){
26      int pivo=0;
27      if(fim>inicio){
28          pivo = particiona(V, inicio, fim);
29          quickSort(V, inicio, pivo-1);
30          quickSort(V, pivo+1, fim);
31      }
32  }
33
34  void main(){
35      int V[4] = {23, 4, 67, -8};
36      quickSort(V, 0, 3);
37  }
```

0	1	2	3
-8	4	23	67

Linha	V	inicio	fim	pivo
36	Execução 1: quickSort(V, 0, 3)			
25	(23, 4, 67, -8)	0	3	
28				2

Linha	V	inicio	fim	pivo
29	Execução 2: quickSort(V, 0, 1)			
25	(-8, 4, 23, 67)	0	1	
28	particiona(V, 0, 1)			

Pilha de Recursão



Execução 2: quickSort(V, 0, 1)

Execução 1: quickSort(V, 0, 3)

```
1  int particiona(int *V, int inicio, int final1){
2      int esq, dir, pivo, aux;
3      esq = inicio;
4      dir = final1;
5      pivo = V[inicio];
6
7      while(esq < dir){
8          while(V[esq] <= pivo){
9              esq++;
10         }
11         while(V[dir] > pivo){
12             dir--;
13         }
14         if(esq < dir){
15             aux = V[esq];
16             V[esq] = V[dir];
17             V[dir] = aux;
18         }
19     }
20     V[inicio] = V[dir];
21     V[dir] = pivo;
22     return dir;
23 }
```

Linha	Inicio	final1	esq	dir	pivo	aux	V	fim
1	0	1						
3			0					
4				1				
5					-8			
9			1					
12				0				
20							[0]=-8	
21							[0]=-8	
22				0				

0	1	2	3
-8	4	23	67

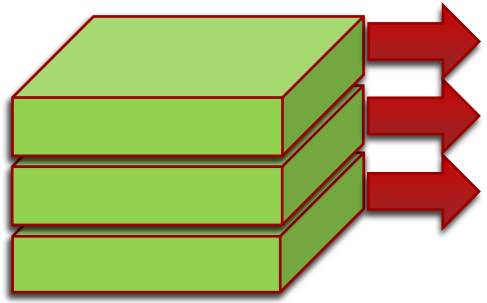
```
25  void quickSort(int *V, int inicio, int fim){
26      int pivo=0;
27      if(fim>inicio){
28          pivo = particiona(V, inicio, fim);
29          quickSort(V, inicio, pivo-1);
30          quickSort(V, pivo+1, fim);
31      }
32  }
33
34  void main(){
35      int V[4] = {23, 4, 67, -8};
36      quickSort(V, 0, 3);
37  }
```

0	1	2	3
-8	4	23	67

Execução 1 Encerrada!

Execução 2 Encerrada!

Pilha de Recursão



- Execução 3: quickSort(V, 0, -1)
- Execução 2: quickSort(V, 0, 1)
- Execução 1: quickSort(V, 0, 3)

Linha	V	inicio	fim	pivo
36	Execução 1: quickSort(V, 0, 3)			
25	(23, 4, 67, -8)	0	3	
28				2

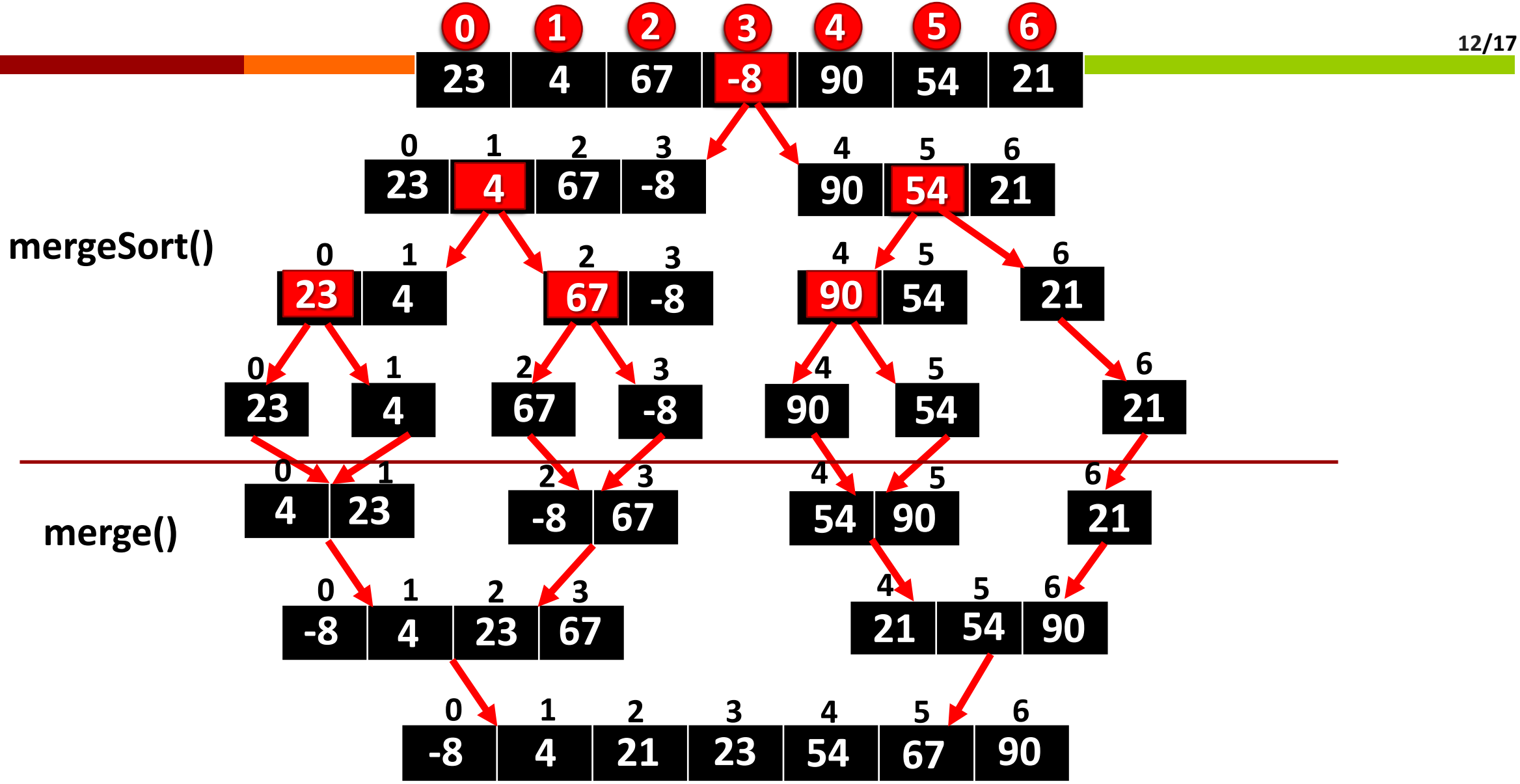
Linha	V	inicio	fim	pivo
30	Volta para a Execução 1: quickSort(V, 3, 3)			
25	(-8, 4, 23, 67)	3	3	

Linha	V	inicio	fim	pivo
30	Volta para a Execução 2: quickSort(V, 1, 1)			
25	(-8, 4, 23, 67)	1	1	

Merge Sort

Merge Sort

- Também conhecido como ordenação por “intercalação”;
- Algoritmo recursivo que usa a ideia de DIVIDIR PARA CONQUISTAR para ORDENAR os dados de um *array*;
- O algoritmo parte do princípio que é mais fácil ordenar um conjunto com poucos dados do que um conjunto com muitos:
  - Portanto, os dados são DIVIDIDOS em conjuntos cada vez MENORES para depois ordená-los e combiná-los por meio da intercalação (MERGE).


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
mergeSort()
merge()
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
0 1 2 3 4 5 6
4 23 -8 67 54 90 21
0 1 2 3 4 5 6
-8 4 23 67 21 54 90
0 1 2 3 4 5 6
-8 4 21 23 54 67 90

```
45  void mergeSort(int *V, int inicio, int fim){
46      int meio = 0;
47
48      if(inicio < fim){
49          meio = (inicio+fim)/2;
50          mergeSort(V, inicio, meio);
51          mergeSort(V, meio+1, fim);
52          merge(V, inicio, meio, fim);
53      }
54  }
55
56  void main(){
57      int vetor[7]={23, 4, 67, -8, 90, 54, 21};
58
59      mergeSort(vetor, 0, 6);
60  }
```


23	4	67	-8	90	54	21

Linha	V	inicio	fim	meio
50	(2) mergeSort(V, 0, 3)			
45	(23, 4, 67, -8, 90, 54, 21)	0	3	
49				1

Linha	V	inicio	fim	meio
50	(3) mergeSort(V, 0, 1)			
45	(23, 4, 67, -8, 90, 54, 21)	0	1	
49				0
52	merge(V, 0, 0, 1)			

Linha	V	inicio	fim	meio
51	(5) mergeSort(V, 1, 1)			
45	(23, 4, 67, -8, 90, 54, 21)	1	1	

Linha	V	inicio	fim	meio
57				
59				
45				
49				

Não é uma chamada para a função recursiva, Então, não é armazenada na PILHA DE RECURSÃO!! Se no lugar da função merge(), tivesse um printf("%i, %i e %i", inicio, fim, meio);, a SAÍDA seria 0, 1 e 0.

```
4  void merge(int *V, int inicio, int meio, int fim
5      int *temp = NULL, p1, p2, tam, i, j, k;
6      int fim1 = 0, fim2 = 0;
7
8      tam = fim - inicio + 1;
9      p1 = inicio;
10     p2 = meio + 1;
11     temp = (int *) malloc(tam*sizeof(int));
12
13     if(temp!=NULL){
14         for(i=0;i<tam; i++){
15             if(!fim1 && !fim2){
16                 if(V[p1] < V[p2]){
17                     temp[i]=V[p1++];
18                 }else{
19                     temp[i]=V[p2++];
20                 }
21                 if(p1>meio){
22                     fim1=1;
23                 }
24                 if(p2>fim){
25                     fim2=1;
26                 }
27             }
28             else{
29                 if(!fim1){
30                     temp[i]=V[p1++];
31                 }else{
32                     temp[i]=V[p2++];
33                 }
34             }
35         }
36         k=inicio;
37         for(j=0; j<tam;j++){
38             V[k]=temp[j];
39             k++;
40         }
41     }
42     free(temp);
43 }
```

V=(4, 23, 67, -8, 90, 54, 21)

Linha	V	Inicio	meio	fim	*temp	p1	p2	tam	I	J	k	fim1	fim2
4		0	0	1									
6												0	0
8								2					
9						0							
10							1						
11					(?, ?)								
14									0				
19					[0]=4		2						
25													1
14									1				
30					[1]=[23]	1							
14									2				
36											0		
37										0			
38	[0]=4												
39											1		
37										1			
38	[1]=23												
39											2		
37										2			

```
45  void mergeSort(int *V, int inicio, int fim){
46      int meio = 0;
47
48      if(inicio < fim){
49          meio = (inicio+fim)/2;
50          mergeSort(V, inicio, meio);
51          mergeSort(V, meio+1, fim);
52          merge(V, inicio, meio, fim);
53      }
54  }
55
56  void main(){
57      int vetor[7]={23, 4, 67, -8, 90, 54, 21};
58
59      mergeSort(vetor, 0, 6);
60  }
```

Linha	V	inicio	fim	meio
50	(2) mergeSort(V, 0, 3)			
45	(23, 4, 67, -8, 90, 54, 21)	0	3	
49				1

Linha	V	inicio	fim	meio
50	(3) mergeSort(V, 0, 1)			
45	(23, 4, 67, -8, 90, 54, 21)	0	1	
49				0
52	merge(V, 0, 0, 1)			

Linha	V
57	(23, 4, 67, -8,
59	(1) merg
45	(23, 4, 67, -8,
49	

Não é uma chamada para a função recursiva, Então, não é armazenada na PILHA DE RECURSÃO!!

```
45  void mergeSort(int *V, int inicio, int fim){
46      int meio = 0;
47
48      if(inicio < fim){
49          meio = (inicio+fim)/2;
50          mergeSort(V, inicio, meio);
51          mergeSort(V, meio+1, fim);
52          merge(V, inicio, meio, fim);
53      }
54  }
55
56  void main(){
57      int vetor[7]={23, 4, 67, -8, 90, 54, 21};
58
59      mergeSort(vetor, 0, 6);
60  }
```

Linha	V	inicio	fim	meio
57	(4, 23, 67, -8, 90, 54, 21)			
59	(1) mergeSort(vetor, 0, 6)			
45	(4, 23, 67, -8, 90, 54, 21)	0	6	
49				3

Linha	V	inicio	fim	meio
50	(2) mergeSort(V, 0, 3)			
45	(4, 23, 67, -8, 90, 54, 21)	0	3	
49				1

Linha	V	inicio	fim	meio
51	mergeSort(V, 2, 3)			
45	(4, 23, 67, -8, 90, 54, 21)	2	3	
49				2

Referências

- Slides do Prof. Luiz Rozante;
- SALES, André Barros de; AMVAME-NZE, Georges. Linguagem C: roteiro de experimentos para aulas práticas. 2016;
- BACKES, André. Linguagem C Completa e Descomplicada. Editora Campus. 2013;
- SCHILDT, Herbert. C Completo e Total. Makron Books. 1996;
- DAMAS, Luís. Linguagem C. LTC Editora. 1999;
- DEITEL, Paul e DEITEL, Harvey. C Como Programar. Pearson. 2011;
- BACKES, André. Estrutura de Dados Descomplicada em Linguagem C. GEN LTC. 2016.