Exercicio Slide 43:

O algoritmo de ordenação por bolha padrão, conforme descrito no método implementado bubble, tem um problema: sempre irá percorrer todas as posições do vetor mesmo com o mesmo já estando ordenado. Isso se dá pelo laço for mais interno que percorre n-1 comparações.

Pode-se melhorar o desempenho do método fazendo com que o laço for mais interno não percorra n-1 vezes mas sim n-(size-1), ou seja, agora o laço irá comparar o tamanho do vetor menos uma posição, porque desse modo sabemos que o lado esquerdo do array já está ordenado e as comparações a partir do array indexado pelo indice dos elementos que não foram ordenados ainda.

Execício Slide 54:

Ao implementar os três métodos de ordenação e a testá-los juntamente com a planilha de testes de mesa para o *array* proposto no exercício observou-se que o método de ordenação por Inserção(Insertion Sort) foi quem menos realizou comparações, com o total de 11 comparações, e como consequência foi o método que gastou menos recursos computacionais. Isso se deve pela forma a qual o algoritmo de Inserção por Seleção funciona. Cada um dos algoritmos de ordenação manipula os índices do *array* de uma forma diferente, ambos os algoritmos de ordenação por bolha e o algoritmo de ordenação por seleção trabalham com o conceito de laços for aninhados, o que prende a execução do código durante a passagem por estes laços, diferente da forma a qual o algoritmo de seleção por Inserção utiliza tais índices: um laço for juntamente com um laço while que não irá necessariamente sempre percorrer o *array* todo para encontrar e mover para a posição correta durante o processo de ordenação

Observando a implementação do código *C* dos três algoritmos nota-se que o método de ordenação por bolha é o algoritmo mais simples, ele apenas acha o menor elemento e troca a sua posição dentro do array. No caso de o *array* já estiver ordenado o algoritmo ainda continua as comparações até passar por todos os índices, ordenação por bolha apresentou o total de comparações 31 sendo, assim o pior dos três. Os outros dois métodos são um pouco mais sofisticados e isso impactou diretamente na quantidade de comparações realizadas.

Vale notar que para o melhor caso os algoritmos de ordenação por bolha e o algoritmo de ordenação por inserção tem a complexidade n, enquanto o algoritmo de ordenação por seleção tem a complexidade n^2.

Bubble Sort									
Var	Value	Array						Compare	Result
i	0	22	24	22	38	18	31		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	#								
i	0	22	22	24	38	18	31		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	24								
i	0	22	22	24	38	18	31		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	24								
i	0	22	22	24	18	38	31		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	38								
i	0	22	22	24	18	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	38								
i	1	22	22	24	18	31	38		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	38								
i	1	22	22	24	18	31	38		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	38								
i	1	22	22	18	24	31	38		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	24								

i	1	22	22	18	24	31	38		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	24								
i	1	22	22	18	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	24								
i	2	22	22	18	24	31	38		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	24								
i	2	22	18	22	24	31	38		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	22								
i	2	22	18	22	24	31	38		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	2	22	18	22	24	31	38		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	2	22	18	22	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	3	18	22	22	24	31	38		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	T
helper	22								
i	3	18	22	22	24	31	38		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	F

helper	22								
i	3	18	22	22	24	31	38		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	3	18	22	22	24	31	38		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	3	18	22	22	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	4	18	22	22	24	31	38		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	4	18	22	22	24	31	38		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	4	18	22	22	24	31	38		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	4	18	22	22	24	31	38		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	4	18	22	22	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								

i	5	18	22	22	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	5	18	22	22	24	31	38		
j	0	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	5	18	22	22	24	31	38		
j	1	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	5	18	22	22	24	31	38		
j	2	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	5	18	22	22	24	31	38		
j	3	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								
i	5	18	22	22	24	31	38		
j	4	0	1	2	3	4	5	v[j] > v[j+1]	F
helper	22								

Selection_Sort(Arr[], size)									
Var	Value	Array						Compare	Result
i	0	22	24	22	38	18	31		
j	1	0	1	2	3	4	5	v[j] < v[min]	F
min	0								
helper	#								
i	0	22	24	22	38	18	31		
j	2	0	1	2	3	4	5	v[j] < v[min]	F
min	0								
helper	#								
i	0	22	24	22	38	18	31		
j	3	0	1	2	3	4	5	v[j] < v[min]	F
min	0								
helper	#								
i	0	22	24	22	38	18	31		
j	4	0	1	2	3	4	5	v[j] < v[min]	T
min	4								
helper	#								
i	0	18	24	22	38	22	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	4								
helper	22								
i	1	18	24	22	38	22	31		
j	2	0	1	2	3	4	5	v[j] < v[min]	T
min	2								
helper	22								
i	1	18	24	22	38	22	31		

j	3	0	1	2	3	4	5	v[j] < v[min]	F
min	2								
helper	22								
i	1	18	24	22	38	22	31		
j	3	0	1	2	3	4	5	v[j] < v[min]	F
min	2								
helper	22								
i	1	18	22	24	38	22	31		
j	3	0	1	2	3	4	5	v[j] < v[min]	F
min	2								
helper	24								
i	2	18	22	24	38	22	31		
j	3	0	1	2	3	4	5	v[j] < v[min]	F
min	2								
helper	24								
i	2	18	22	24	38	22	31		
j	4	0	1	2	3	4	5	v[j] < v[min]	T
min	4								
i	2	18	22	24	38	22	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	4								
i	2	18	22	22	38	24	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	4								
Helper	24								
i	3	18	22	22	38	24	31		
j	4	0	1	2	3	4	5	v[j] < v[min]	T

min	4								
Helper	24								
i	3	18	22	22	38	24	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	4								
Helper	24								
i	3	18	22	22	24	38	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	4								
Helper	38								
i	4	18	22	22	24	38	31		
j	5	0	1	2	3	4	5	v[j] < v[min]	T
min	5								
Helper	38								
i	4	18	22	22	24	31	38		
j	5	0	1	2	3	4	5	v[j] < v[min]	F
min	5								
Helper	38								

Insertion_Sort(Arr[], size)									
Var	Value	Array						Compare	Result
i	1	22	24	22	38	18	31		
j	0	0	1	2	3	4	5	v[j] > pivot	F
pivot	24								
i	2	22	24	22	38	18	31		
j	1	0	1	2	3	4	5	v[j] > pivot	T
pivot	22								
i	2	22	22	24	38	18	31		
j	0	0	1	2	3	4	5	v[j] > pivot	F
pivot	22								
i	3	22	22	24	38	18	31		
j	2	0	1	2	3	4	5	v[j] > pivot	F
pivot	38								
i	4	22	22	24	38	18	31		
j	3	0	1	2	3	4	5	v[j] > pivot	T
pivot	18								
i	4	22	22	24	38	38	31		
j	2	0	1	2	3	4	5	v[j] > pivot	T
pivot	18								
i	4	22	22	24	24	38	31		
j	1	0	1	2	3	4	5	v[j] > pivot	T
pivot	18								
i	4	22	22	22	24	38	31		
j	0	0	1	2	3	4	5	v[j] > pivot	T
pivot	18								

i	4	18	22	22	24	38	31		
j	-1	0	1	2	3	4	5	v[j] > pivot	T
pivot	18								
i	5	18	22	22	24	38	31		
j	4	0	1	2	3	4	5	v[j] > pivot	T
pivot	31								
i	5	18	22	22	24	31	38	v[j] > pivot	N
j	3	0	1	2	3	4	5		
pivot	31								