SCC0201/501 (Introdução à Ciência de Computação II) — Prof. Moacir P. Ponti Junior

Trabalho 2

Implemente sua atividade sozinho sem compartilhar, olhar código de seus colegas, ou buscar na Internet. Procure usar apenas os conceitos já vistos nas aulas.

Quicksort iterativo

O objetivo deste trabalho é implementar uma versão *não-recursiva* (ou *iterativa*) do algoritmo de ordenação *quicksort*. O *quicksort* pertence à classe dos chamados métodos de *divisão-e-conquista*. Existem duas estratégias empregadas no desenvolvimento de algoritmos pertencentes a essa classe: a estratégia *recursiva* e a estratégia *iterativa*. Cada abordagem, no entanto, pode ser convertida na outra. No caso de algoritmos iterativos, o principal comando de repetição pode ser transformado em uma chamada recursiva, desde que cuidadosamente controlada por meio de contadores. Por outro lado, uma implementação recursiva pode ser "simulada" utilizando uma estrutura de dados *pilha* (de fato, a recursão é sempre convertida em uma pilha – a chamada *pilha de execução* – pelos compiladores).

No caso do algoritmo *quicksort*, a estratégia iterativa apresenta como vantagem o fato de estabelecer limites para o tamanho da pilha. Assim, armazenando-se na pilha (função `empilhar()`) a maior partição enquanto a menor é ordenada, pode-se estabelcer que a quantidade máxima de memória alocada na pilha é da ordem de $\mathcal{O}(log_2(n))$, onde n é o número de elementos do vetor a ser ordenado.

Tarefa

Desenvolva a versão iterativa do algoritmo *quicksort*. Para isso, deve ser implementada uma estrutura de dados pilha. O programa deve receber como entrada uma lista de tamanho n, tal que $2 \leq n \leq 50$ e gerar como saída uma lista ordenada. O programa também deve ser capaz de definir o tamanho máximo da pilha, de modo a evitar alocação inadequada de memória.

Deve-se implementar funções de manipulação de pilhas necessárias ao desenvolvimento do projeto. Funções que não serão utilizadas não precisam ser criadas.

Entrada e saída

A entrada será composta de:

1. um número inteiro n, sendo que $2 \leq n \leq 50$, que representa o número de elementos do vetor a ser ordenado;
2. uma sequência de inteiros, de tamanho n.

A saída será composta pela sequência de números ordenada.

Exemplo de entrada

```
9
8 10 3 1 9 8 8 0 7
```

Exemplo de saída

```
0 1 3 7 8 8 8 9 10
```

Instruções

O projeto será avaliado principalmente levando em consideração:

1. Processamento correto das entradas e saídas do programa;
2. Realização das tarefas descritas;
3. Bom uso das técnicas de programação;
4. Boa endentação e uso de comentários no código.

Restrições:

- **Não** deverá ser utilizada qualquer variável global.
- As seguintes funções deverão **obrigatoriamente** ser implementadas e utilizadas:
  1. `int desempilha(Pilha P[])` – deve receber como parâmetro uma pilha `P` e devolver o topo da mesma.
  2. `void empilha(Pilha P[], int elem)` – deve receber como parâmetro uma pilha `P` e um número que deve ser empilhado.
  3. `int vazia(Pilha P[])` – deve receber como parâmetro uma pilha `P` e devolver `1`, caso a pilha seja vazia, e `0`, caso contrário.
- Você poderá criar outras funções se quiser ou precisar — lembre-se de tornar a função criada útil para os propósitos de reuso e abstração, e de comentá-la corretamente.
- Não poderão ser utilizadas bibliotecas com funções prontas.

ATENÇÃO: o projeto deverá ser entregue apenas pelo SQTPM no formato `C`, escolhendo a opção `Trabalho2`. Há um caso de teste para serem baixados na página da disciplina. O sistema receberá trabalhos **apenas** entre os dias 06/11/2010, às 0h00, e 09/11/2010 às 23h59. Não serão aceitos trabalhos com atraso.

Podem utilizar o editor de sua preferência, mas compilem e executem o código utilizando o `gcc`:

```
gcc numusp.c -o numusp
```

Para executar e verificar os casos de teste, executem:

```
./numusp < caseX.in
```

Qualquer dúvida nos casos de teste disponibilizados, enviar e-mail para o estagiário PAE com o assunto `[trab2]duvida`.