#ifndef \_SORTING\_H\_

#define \_SORTING\_H\_

#include <stdio.h>

#include <stdlib.h>

/\*\* Doubly Linked List Structure \*/

/\*\*

\* struct listint\_s - Node in a doubly linked list

\*

\* @n: Integer value stored in the node

\* @prev: Pointer to the previous node in the list

\* @next: Pointer to the next node in the list

\*/

typedef struct listint\_s

{

const int n;

struct listint\_s \*prev;

struct listint\_s \*next;

} listint\_t;

/\*\* Provided Function Prototypes \*/

/\*\*

\* print\_array - Prints an array of integers

\* @array: The array to be printed

\* @size: Number of elements in the array

\*/

void print\_array(const int \*array, size\_t size);

/\*\*

\* print\_list - Prints a doubly linked list of integers

\* @list: The list to be printed

\*/

void print\_list(const listint\_t \*list);

/\*\* Task-Specific Function Prototypes \*/

void bubble\_sort(int \*array, size\_t size);

int len\_list(listint\_t \*head);

void swap(listint\_t \*node\_a, listint\_t \*node\_b, listint\_t \*\*head);

void insertion\_sort\_list(listint\_t \*\*list);

void swap(int \*a, int \*b);

int partition(int \*array, int low, int high, size\_t size);

void lomuto\_qsort(int \*array, int low, int high, size\_t size);

void quick\_sort(int \*array, size\_t size)

void swap(int \*a, int \*b);

int partition(int \*array, int low, int high, size\_t size);

void lomuto\_qsort(int \*array, int low, int high, size\_t size);

void quick\_sort(int \*array, size\_t size)

void swap\_ints(int \*a, int \*b);

int lomuto\_partition(int \*array, size\_t size, int left, int right);

void lomuto\_sort(int \*array, size\_t size, int left, int right);

void quick\_sort(int \*array, size\_t size);

void selection\_sort(int \*array, size\_t size);

void quick\_recursion(int \*array, int left, int right, size\_t size);

int partition(int \*array, int left, int right, size\_t size);

void shell\_sort(int \*array, size\_t size);

void switch\_nodes(listint\_t \*\*list, listint\_t \*\*p);

void cocktail\_sort\_list(listint\_t \*\*list);

void counting\_sort(int \*array, size\_t size);

void merge\_sort(int \*array, size\_t size);

void merge\_recursion(int \*arr, int \*array, size\_t left, size\_t right);

void merge\_subarray(int \*arr, int \*array, size\_t left,

size\_t middle, size\_t right);

void heap\_sort(int \*array, size\_t size);

void heapify(int \*array, size\_t s, size\_t root, size\_t size);

void radix\_sort(int \*array, size\_t size);

void count\_sort\_LSD(int \*array, size\_t size, size\_t lsd);

void quick\_sort\_hoare(int \*array, size\_t size);

void bitonic\_sort(int \*array, size\_t size);

void bitonic\_recursion(int \*array, int l, int r, int direction, size\_t size);

void bitonic\_merge(int \*array, int l, int r, int direction);

#endif /\* \_GHabwe\_SORTING\_H\_ \*/