```c
#include "SORT.H"
#include "BDSCTEST.H"
#define TRUE 1
#define FALSE 0

int sorted(arr, n)
int arr[];
int n;
{
  int i, prev;
  prev = arr[0];
  for (i = 1; i < n; i++)
    if (arr[i] < prev)
      return FALSE;
    else
      prev = arr[i];
  return TRUE;
}

int is_heap(arr, i, size)
int arr[];
int i, size;
{
  int lchild, rchild;
  lchild = 2 * i + 1;
  rchild = 2 * i + 2;
  if (lchild >= size) return TRUE; /* It's a leaf node */
  if (rchild >= size) return arr[i] >= arr[lchild]; /* Only has left child *
/
  if ((arr[i] < arr[lchild]) || (arr[i] < arr[rchild])) return FALSE;
  return is_heap(arr, lchild, size) && is_heap(arr, rchild, size);
}

main()
{
  int a[16];
  int b[7];
  int c[9];
  TEST_CASE("Bubble Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    bubble_sort(a, 16);
    ASSERT(sorted(a, 16));
  }
  TEST_CASE("Insertion Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    insertion_sort(a, 16);
    ASSERT(sorted(a, 16));
  }
  TEST_CASE("Selection Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    selection_sort(a, 16);
    ASSERT(sorted(a, 16));
  }
  TEST_CASE("Merge Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    merge_sort(a, 16);
    ASSERT(sorted(a, 16));
  }
  TEST_CASE("Quick Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    quick_sort(a, 16);
    ASSERT(sorted(a), 16);
  }
  TEST_CASE("Radix Sort") {
    initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
    radix_sort(a, 16);
    ASSERT(sorted(a, 16));
  }
  TEST_CASE("Test is_heap") {
    initw(b, "13, 8, 11, 4, 2, 7, 9");
    ASSERT(is_heap(b, 0, 7));
    initw(b, "8, 13, 11, 4, 2, 7, 9");
    ASSERT(!is_heap(b, 0, 7));
```

```
 73      initw(a, "42, 30, 36, 20, 8, 17, 12, 3, 9, 6, 7, 13, 5, 4, 11, 2");
 74      ASSERT(is_heap(a, 0, 16));
 75      initw(c, "13, 8, 11, 4, 2, 7, 9, 12, 42");
 76      ASSERT(is_heap(c, 0, 7));
 77      ASSERT(!is_heap(c, 0, 8));
 78    }
 79    TEST_CASE("Test reheap_up") {
 80      initw(c, "13, 8, 11, 4, 2, 7, 9, 12, 42");
 81      ASSERT(!is_heap(c, 0, 8));
 82      reheap_up(c, 8);
 83      ASSERT(is_heap(c, 0, 8));
 84      ASSERT(!is_heap(c, 0, 9));
 85      reheap_up(c, 9);
 86      ASSERT(is_heap(c, 0, 9));
 87    }
 88    TEST_CASE("Test heapify") {
 89      initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
 90      ASSERT(!is_heap(a, 0, 16));
 91      heapify(a, 16);
 92      ASSERT(is_heap(a, 0, 16));
 93      initw(c, "42, 13, 8, 11, 4, 2, 7, 9, 12");
 94      heapify(c, 9);
 95      ASSERT(is_heap(c, 0, 9));
 96    }
 97    TEST_CASE("Test reheap_down") {
 98      initw(a, "42, 30, 36, 20, 8, 17, 12, 3, 9, 6, 7, 13, 5, 4, 11, 2");
 99      ASSERT(is_heap(a, 0, 16));
100      swap(&a[0], &a[15]);
101      ASSERT(!is_heap(a, 0, 16));
102      reheap_down(a, 15);
103      ASSERT(is_heap(a, 0, 15));
104      swap(&a[0], &a[14]);
105      reheap_down(a, 14);
106      ASSERT(is_heap(a, 0, 14));
107    }
108    TEST_CASE("Heap Sort") {
109      initw(b, "1, 2, 3, 4, 5, 6, 7");
110      heap_sort(b, 7);
111      ASSERT(sorted(b, 7));
112      initw(c, "42, 13, 8, 11, 4, 2, 7, 9, 12");
113      heap_sort(c, 9);
114      ASSERT(sorted(c, 9));
115      initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
116      heap_sort(a, 16);
117      ASSERT(sorted(a, 16));
118    }
119    TEST_CASE("Tree Sort") {
120      initw(a, "31, 4, 15, 9, 26, 5, 35, 8, 97, 23, 84, 62, 64, 38, 35, 27");
121      tree_sort(a, 16);
122      ASSERT(sorted(a, 16));
123    }
124  }
```