Heapsort Inputs:

```
  // testing functions

  vector<int> heap1{-1,1,4,5,7,8,2,8,534,8,4358,23,5,72};

  cout << "Unsorted list: " << endl;
  for (int i = 1; i < heap1.size(); i++)
      cout << heap1[i] << " ";
  cout << endl;

  buildHeap(heap1);

  cout << "Sorted into heap: " << endl;
  for (int i = 1; i < heap1.size(); i++)
      cout << heap1[i] << " ";
  cout << endl;

heapSort(heap1);

  cout << "After heapSort: " << endl;
  for (int i = 1; i < heap1.size(); i++)
      cout << heap1[i] << " ";
  cout << endl;

  vector <int> heap2{};

  cout << "Unsorted list: " << endl;
```

```
for (int i = 1; i < heap2.size(); i++)
    cout << heap2[i] << " ";
cout << endl;

buildHeap(heap2);

cout << "Sorted into heap: " << endl;
for (int i = 1; i < heap2.size(); i++)
    cout << heap2[i] << " ";
cout << endl;

heapSort(heap2);

cout << "After heapSort: " << endl;
for (int i = 1; i < heap2.size(); i++)
    cout << heap2[i] << " ";
cout << endl;
```

Heapsort Outputs:

```
Unsorted list:
1 4 5 7 8 2 8 534 8 4358 23 5 72
Sorted into heap:
4358 534 72 8 23 5 8 7 1 8 4 5 2
After heapSort:
1 2 4 5 5 7 8 8 8 23 72 534 4358
Unsorted list:

Sorted into heap:

After heapSort:
```

First Quicksort Inputs:

```
// testing functions

vector<int> array{0, 15, 16, 72, 13, 6, 91, 4};

cout << "Unsorted list: " << endl;
for (int i = 0; i < array.size(); i++)
    cout << array[i] << " ";
cout << endl;

quickSort(array);

cout << "Sorted list: " << endl;
for (int i = 0; i < array.size(); i++)
    cout << array[i] << " ";
cout << endl;

vector <int> array2{};

cout << "Unsorted list: " << endl;
for (int i = 1; i < array2.size(); i++)
    cout << array2[i] << " ";
cout << endl;

quickSort(array2);

cout << "Sorted list: " << endl;
```

```
for (int i = 1; i < array2.size(); i++)
    cout << array2[i] << " ";
cout << endl;
```

First Quicksort Output:

```
Unsorted list:
0 15 16 72 13 6 91 4
Sorted list:
0 4 6 13 15 16 72 91
Unsorted list:


Sorted list:
```

Random Quicksort Inputs:

```
vector<int> array{0, 15, 16, 72, 13, 6, 91, 4};

cout << "Unsorted list: " << endl;
for (int i = 0; i < array.size(); i++)
    cout << array[i] << " ";
cout << endl;

quickSort(array);

cout << "Sorted list: " << endl;
for (int i = 0; i < array.size(); i++)
    cout << array[i] << " ";
cout << endl;

vector <int> array2{};

cout << "Unsorted list: " << endl;
for (int i = 1; i < array2.size(); i++)
    cout << array2[i] << " ";
cout << endl;

quickSort(array2);
cout << "Sorted list: " << endl;
for (int i = 1; i < array2.size(); i++)
    cout << array2[i] << " ";
cout << endl;
```

Random Quicksort Outputs:

```
Unsorted list:
0 15 16 72 13 6 91 4
Sorted list:
0 4 6 13 15 16 72 91
Unsorted list:

Sorted list:
```

I chose the above test cases for my code because I wanted to ensure that the program could handle any inputs, ordinary or not, to prove its robustness. The heapsort, upon receiving a vector without a negative 1 at index 0 (which denotes an input that will go untouched, as typically heaps are maintained without an index 0 for simplicity), will output "Vector is implemented incorrectly - there should be a -1 at index 0.". The quicksort implementations have much fewer restrictions on their data sets (including the ability to handle negative numbers as inputs) and therefore more additional testing becomes redundant. Each algorithm was tested in the edge case in which no values were inputted into the list before it was sorted and each handled it properly.