**Code Used to Test Implementations:**

//test all functions of PQHeapTree to verify that they work  
PQHeapTree<int>\* T = new PQHeapTree<int>();  
if(T->empty())  
 cout<< "empty works" << endl;  
T->insert(7);  
cout << T->min()->key << endl;  
T->insert(4);  
cout << T->min()->key << endl;  
cout << T->size() << endl;  
T->removeMin();  
cout << T->min()->key << endl;  
cout << T->size() << endl;  
T->insert(3);  
cout << T->min()->key << endl;  
T->insert(18);  
cout << T->min()->key << endl;  
T->insert(5);  
cout << T->min()->key << endl;  
T->insert(6);  
cout << T->min()->key << endl;  
T->preOrder(T->min());  
cout << endl;  
while(!T->empty()) //make sure that the heap order property put them into min->max order  
{  
 cout << T->min()->key << endl;  
 T->removeMin();  
}  
  
//test all functions of PQHeapArray to verify that they work  
PQHeapArray<int>\* A = new PQHeapArray<int>(100);  
if(A->empty())  
 cout << "empty works" << endl;  
A->insert(7);  
cout << A->min() << endl;  
A->insert(4);  
cout << A->min() << endl;  
cout << A->size() << endl;  
A->removeMin();  
cout << A->min() << endl;  
cout << A->size() << endl;  
A->insert(3);  
cout << A->min() << endl;  
A->insert(18);  
cout << A->min() << endl;  
A->insert(5);  
cout << A->min() << endl;  
A->insert(6);  
cout << A->min() << endl;  
cout << endl;  
while(!A->empty()) //make sure that the heap order property put them into min->max order  
{  
 cout << A->min() << endl;  
 A->removeMin();  
}  
int test[] = {7,3,4,2,1,0,6,5};  
for(int i = 3;i>=0;i--) //test to make sure that heapify works  
 A = A->heapify(test,8,i);  
while(!A->empty())  
{  
 cout << A->min() << endl;  
 A->removeMin();  
}  
  
int test2[] = {0,4,5,3,6,7,1,2};  
for(int i = 0;i<8;i++) //test to make sure heapSort for Arrays works  
 cout << test2[i] << "\t";  
cout << endl;  
heapSortA(test2,8);  
for(int i = 0;i<8;i++)  
 cout << test2[i] << "\t";  
cout << endl;  
  
int test3[] = {0,4,5,3,6,7,1,2};  
for(int i = 0;i<8;i++) //test to make sure that heapsort for Trees works  
 cout << test3[i] << "\t";  
cout << endl;  
heapSortT(test3,8);  
for(int i = 0;i<8;i++)  
 cout << test3[i] << "\t";  
cout << endl;  
  
int test4[] = {0,4,5,3,6,7,1,2};  
for(int i = 0;i<8;i++) //test to make sure that heapsort with heapify works  
 cout << test4[i] << "\t";  
cout << endl;  
heapSortH(test4,8);  
for(int i = 0;i<8;i++)  
 cout << test4[i] << "\t";  
cout << endl;  
  
  
//all tests used to find runtime of algorithms  
//commented out all not currently in use so as to save time and memory  
//tested increasing, decreasing, and random  
//test in use denoted by printing to screen first  
int elems = 10;  
auto start\_time = chrono::high\_resolution\_clock::now();  
  
  
/\*  
cout << "--------PQSortT Increasing--------" << endl;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortT(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
  
cout << "--------PQSortA Increasing--------" << endl;  
elems = 10;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortA(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
  
cout << "--------PQSortH Increasing--------" << endl;  
elems = 10;  
while(elems <= 10000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortH(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
  
cout << "--------PQSortT Decreasing--------" << endl;  
elems = 10;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = elems; i > 0;i--)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortT(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
cout << "--------PQSortA Decreasing--------" << endl;  
elems = 10;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = elems; i > 0;i--)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortA(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
cout << "--------PQSortTH Decreasing------" << endl;  
elems = 10;  
while(elems <= 10000) {  
 int \*s = new int[elems];  
 for(int i = elems; i > 0;i--)  
 s[i] = i;  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortH(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
cout << "--------PQSortT Random--------" << endl;  
elems = 10;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = int(rand()\*100);  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortT(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
  
cout << "--------PQSortA Random--------" << endl;  
elems = 10;  
while(elems <= 10000000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = int(rand()\*100);  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortA(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}  
\*/  
cout << "--------PQSortH Random--------" << endl;  
elems = 10;  
while(elems <= 10000) {  
 int \*s = new int[elems];  
 for(int i = 0; i < elems;i++)  
 s[i] = int(rand()\*100);  
 start\_time = chrono::high\_resolution\_clock::now();  
 heapSortH(s,elems);  
 cout << "Time Used for " << elems << " elements:" << chrono::duration\_cast<chrono::nanoseconds>(chrono::high\_resolution\_clock::now()-start\_time).count() << endl;  
 elems\*=10;  
 delete s;  
}

Numbers were used for easy to understand comparison.

Checked to make sure all functions worked in varying orders.

Outputs were used to check and make sure that the order values were stored in the correct order.

The parts of code commented out are used for the experimental part of the report to check time.

These are commented out to not waste time during repeated testing.

Sample Output:

**empty works**

**7**

**4**

**2**

**7**

**1**

**3**

**3**

**3**

**3**

**3 5 7 6 18**

**3**

**5**

**6**

**7**

**18**

**empty works**

**7**

**4**

**2**

**7**

**1**

**3**

**3**

**3**

**3**

**3**

**5**

**6**

**7**

**18**

**0**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**0 4 5 3 6 7 1 2**

**0 1 2 3 4 5 6 7**

**0 4 5 3 6 7 1 2**

**0 1 2 3 4 5 6 7**

**0 4 5 3 6 7 1 2**

**0 1 2 3 4 5 6 7**

**--------PQSortH Random--------**

**Time Used for 10 elements:2600**

**Time Used for 100 elements:111200**

**Time Used for 1000 elements:10611200**

**Time Used for 10000 elements:750581900**