


```

}


template <typename E, typename C>    // minimum element
const E& ListPriorityQueue<E,C>::min() const
{
    return L.front();                // minimum is at the front
}

template <typename E, typename C>    // remove minimum
void ListPriorityQueue<E,C>::removeMin()
{
    L.pop_front();
}

#endif

```

Exercise 1 -- need to submit source code and I/O

-- check if completely done ; otherwise, discuss issues below

Source code below:

```

/*  Program: PA_12_exercise_1
    Author: Nero Li
    Class: CSCI 220
    Date: 11/23/2021
    Description:
        Put together list priority queue and use a test1 driver to perform
        some operations to confirm it is working correctly. You can use a
        PQ with integer as element. Create two PQ objects - one with
largest
        value having highest priority and one with lowest value having
        highest priority. Be sure to use a comparator for the PQ.

```

I certify that the code below is my own work.

Exception(s): N/A

```

*/
#include <iostream>
#include "ListPriorityQueue.h"

using namespace std;

template <typename E>
class isLess
{
public:
    bool operator()(const E& p, const E& q) const
    {
        return p < q;
    }
};

```

```

template <typename E>
class isMore
{
public:
    bool operator()(const E& p, const E& q) const
    {
        return p > q;
    }
};

int main()
{
    ListPriorityQueue<int, isLess<int>> test1;
    ListPriorityQueue<int, isMore<int>> test2;

    test1.insert(5);
    test1.insert(4);
    test1.insert(7);
    test1.insert(1);
    cout << test1.min() << ' ';
    test1.removeMin();
    test1.insert(3);
    test1.insert(6);
    cout << test1.min() << ' ';
    test1.removeMin();
    cout << test1.min() << ' ';
    test1.removeMin();
    test1.insert(8);
    cout << test1.min() << ' ';
    test1.removeMin();
    test1.insert(2);
    cout << test1.min() << ' ';
    test1.removeMin();
    cout << test1.min() << ' ';
    test1.removeMin();
    cout << endl;

    test2.insert(5);
    test2.insert(4);
    test2.insert(7);
    test2.insert(1);
    cout << test2.min() << ' ';
    test2.removeMin();
    test2.insert(3);
    test2.insert(6);
    cout << test2.min() << ' ';
    test2.removeMin();
    cout << test2.min() << ' ';
    test2.removeMin();
    test2.insert(8);
    cout << test2.min() << ' ';
}

```

```
test2.removeMin();
test2.insert(2);
cout << test2.min() << ' ';
test2.removeMin();
cout << test2.min() << ' ';
test2.removeMin();
cout << endl;

cout << "Modified by: Nero Li\n";
return 0;
}
```


Input/output below:

1 3 4 5 2 6

7 6 5 8 4 3

Modified by: Nero Li

Exercise 2 (with extra credit) -- need to submit source code and I/O

-- check if completely done ; otherwise, discuss issues below

Source code below:

```
/* Program: PA_12_exercise_1
   Author: Nero Li
   Class: CSCI 220
   Date: 11/23/2021
   Description:
       Use your priority queue from exercise 1 to sort data in ascending
order. Sort
       the data file small1k.txt, containing a list of 1,000 integer
values, and output
       the first 5 and last 5 values to the screen (5 values on one line
and at least
       one space between the 2 values). Sort the data file large100k.txt,
containing
       a list of 100,000 integer values, and output the first 5 and last
5 values to
       the screen (5 values on one line and at least one space between
the 2 values).
       For each set of data, collect actual run times in milliseconds and
display to
       the screen as well.
```

I certify that the code below is my own work.

Exception(s): N/A

```
*/
#include <iostream>
#include <fstream>
#include <string>
#include <chrono>
#include "ListPriorityQueue.h"

using namespace std;

template <typename E>
class isless
{
public:
    bool operator()(const E& p, const E& q) const
    {
        return p < q;
    }
};

void func(string inputName, string outputName)
{
```

```

ListPriorityQueue<int, isLess<int>> pq;
ifstream fin;
ofstream fout;
int n{0};
int i{0};

fin.open(inputName, ios::binary);
fout.open(outputName, ios::binary);

auto start = chrono::high_resolution_clock::now();
while (!fin.eof())
{
    fin >> n;
    pq.insert(n);
}

n = pq.size();

while (!pq.empty())
{
    if (i < 5 || i > n - 6)
    {
        cout << pq.min() << ' ';
        fout << pq.min() << ' ';
    }
    if (i == 5 || i == n - 1)
    {
        cout << endl;
        fout << endl;
    }
    ++i;
    pq.removeMin();
}

auto end = chrono::high_resolution_clock::now();
cout << (chrono::duration_cast<chrono::nanoseconds>(end -
start).count() * (double)1e-6) << " ms" << endl;
fout << (chrono::duration_cast<chrono::nanoseconds>(end -
start).count() * (double)1e-6) << " ms" << endl;
}

int main()
{
    func("small1k.txt", "small1k_output.txt");
    func("large100k.txt", "large100k_output.txt");

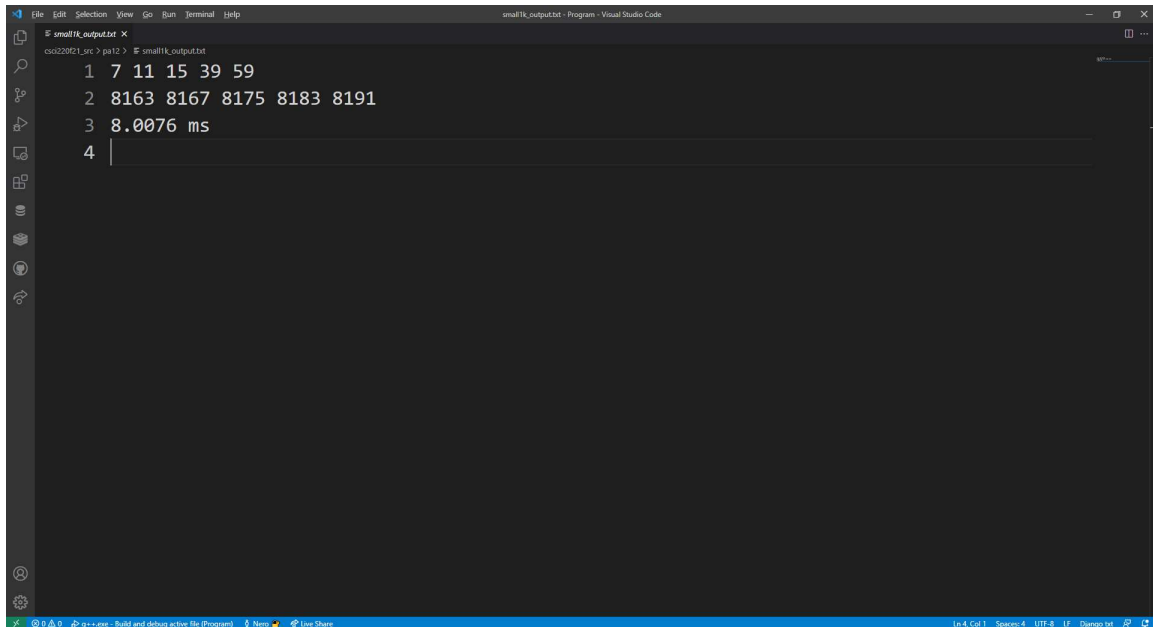
    cout << "Modified by: Nero Li\n";
    return 0;
}

```

Input/output below:

```
7 11 15 39 59
8163 8167 8175 8183 8191
8.0076 ms
1 2 3 4 5
99996 99997 99998 99999 100000
76069.5 ms
Modified by: Nero Li
```

Screenshot for file below:



```
1 7 11 15 39 59
2 8163 8167 8175 8183 8191
3 8.0076 ms
4
```

Answer for Question 1:

PQ is called priority queue, which the data that at the front of the queue should be the smallest. Because of that, we need to modify our queue again and again to maintain the smallest value will become the value that we popped out. This is also the biggest difference that priority is different than a queue.

Answer for Question 2:

The running time for me sorting algorithm in exercise 2 should be $O(n^2)$, and the time output that we have for exercise 2 shows this running time that the gap between 1k to 100k is 10k ms. The reason for this is because when we use list to implement that, we can think the sorting as a bubble sort algorithm since we need to go through all the value that we have and then insert that value in the proper place.