

Data Structures and Algorithm Analysis

Lab 9, Priority Queue

Contents

- Using priority queue with algs4.
- Implement priority queue.

Priority queue with algs4

Maximum priority queue:

<https://algs4.cs.princeton.edu/code/javadoc/edu/princeton/cs/algs4/MaxPQ.html>

Just in case you cannot find the list of all algs4 classes:

<https://algs4.cs.princeton.edu/code/>

Using max priority queue with algs4

Using the MaxPQ class in algs4 library:

```
import edu.princeton.cs.algs4.MaxPQ;
import edu.princeton.cs.algs4.StdOut;

public class TestMaxPQ {
    public static void main( String[] args ) {

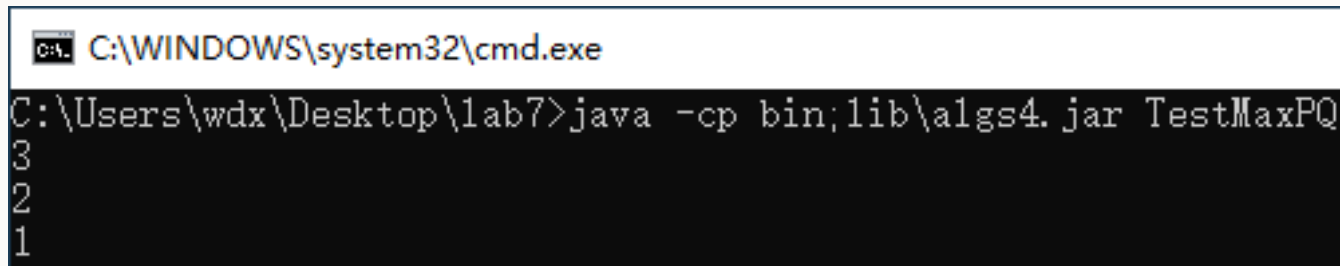
        MaxPQ<Integer> pq = new MaxPQ<>(20);

        pq.insert(1);
        pq.insert(3);
        pq.insert(2);

        while( pq.size() > 0 ) {
            StdOut.println(pq.delMax());
        }
    }
}
```

Using max priority queue with algs4

The above code should produce the following result:



```
C:\WINDOWS\system32\cmd.exe
C:\Users\wdx\Desktop\lab7>java -cp bin;lib\algs4.jar TestMaxPQ
3
2
1
```

Note that it "delMax()" returns and deletes the largest value in the priority queue.

Implement max priority queue

Let's implement our own max priority queue for integers.

It should support inserting elements into the queue, deleting and returning the largest elements in the queue.

```
public class MaxPQ {  
    public void insert( int value ) {  
    }  
    public int delMax() {  
    }  
}
```

Implement max priority queue

This time we will implement a binary heap using an array. So let's create a class member as an array. We should also write a constructor to initialize the array.

```
public class MaxPQ {  
  
    private int[] pq;  
    private int count;  
  
    public MaxPQ( int capacity ) {  
        pq = new int[capacity+1];  
        count = 0;  
    }  
  
    public void insert( int value ) {  
    }  
    public int delMax() {  
    }  
}
```

Implement max priority queue

Now implementing the insert of the of the queue. Remember that we are using `pq[1]` at the root of the tree. We will not use `pq[0]`.

```
private void resize( int newsize ) {}  
private void swim( int idx ) {}  
  
public void insert( int value ) {  
    if( count+1 >= pq.length )  
        resize( pq.length*2 );  
    pq[++count] = value;  
    swim( count );  
}
```


Implement max priority queue

The next methods we need to implement are "resize" and "swim".

Since frequently reallocating memory from the system could be time consuming, the usual way of resizing is to double the array size each time it is not enough.

```
private void resize( int newsize ) {  
    int[] tmp = new int[newsize];  
    System.arraycopy(pq, 1, tmp, 1, count);  
    pq = tmp;  
}
```

Implement max priority queue

Implementing swim:

```
private void swim( int idx ) {  
    int value = pq[idx];  
    while( idx > 1 && pq[idx/2] < value ) {  
        pq[idx] = pq[idx/2];  
        idx /= 2;  
    }  
    pq[idx] = value;  
}
```

As long as the parent node is smaller than the value we want to "swim" up, we move the parent node down to the current position.

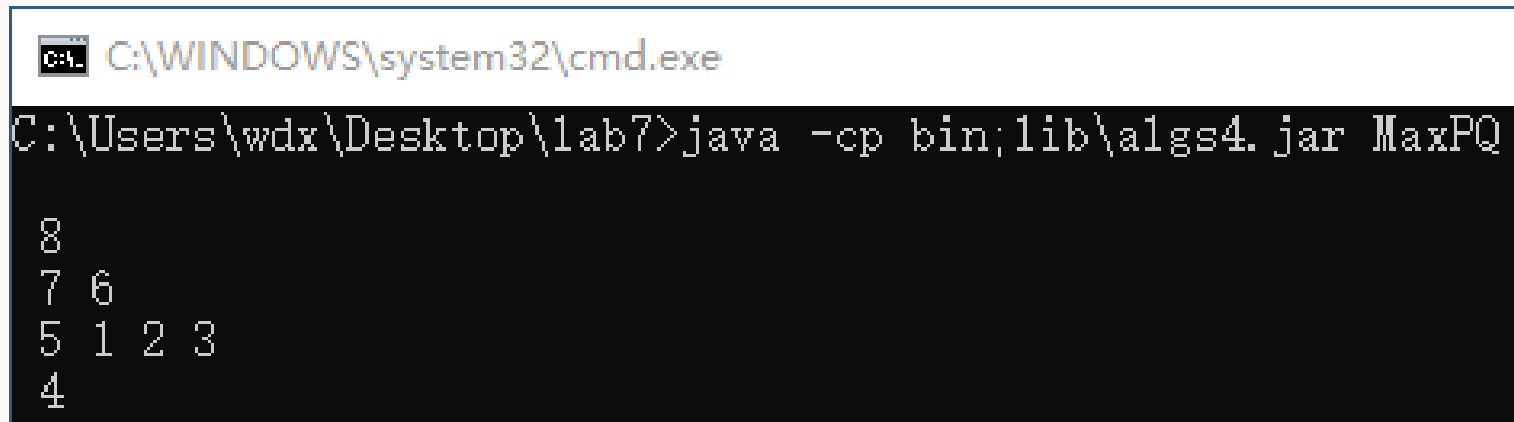
Implement max priority queue

Before we move on to the next step, we could write a simple test to make sure the above code is correct.

```
public static void main( String[] args ) {  
    MaxPQ queue = new MaxPQ(1);  
    int[] values = new int[] { 5, 6, 7, 8, 1, 2, 3, 4 };  
    for( int v : values )  
        queue.insert(v);  
    int j = 1;  
    for( int i = 1; i <=queue.count*2; i *= 2 ) {  
        for( ; j < i && j <= queue.count; ++ j )  
            System.out.print(" "+queue.pq[j]);  
        System.out.println();  
    }  
}
```

Implement max priority queue

From the result we see 8 is greater than 7 and 6. 7 is greater than 5 and 1. 6 is greater than 2 and 3. 5 is greater than 4.



```
C:\WINDOWS\system32\cmd.exe
C:\Users\wdx\Desktop\lab7>java -cp bin;lib\algs4.jar MaxPQ

8
7 6
5 1 2 3
4
```

Now we can move on to implement delMax().

Implement max priority queue

The `delMax()` method deletes and returns the max value in the priority queue. The max value is at `pq[1]`. After we remove that, we move the last element in the queue to the first position and then do a "sink" operation.

```
public int delMax() {
    if( count <= 0 )
        throw new NoSuchElementException("MaxPQ.delMax called
            when count="+count);
    int result = pq[1];
    pq[1] = pq[count--];
    sink( 1 );
    return result;
}
```

Implement max priority queue

Now let's implement sink operation.

```
private void sink( int idx ) {  
    int value = pq[idx];  
    while( idx*2 <= count ) {  
        int id2 = (idx*2+1 <= count && pq[idx*2]<pq[idx*2+1]) ?  
            (idx*2+1) : (idx*2);  
        if( pq[id2] > value ) {  
            pq[idx] = pq[id2];  
            idx = id2;  
        } else  
            break;  
    }  
    pq[idx] = value;  
}
```

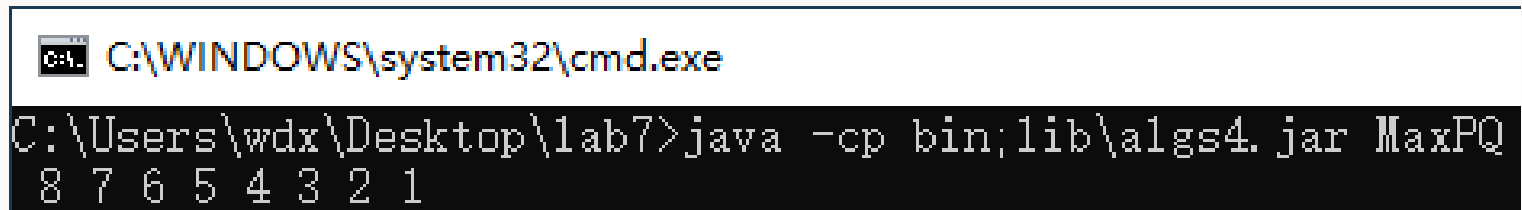
Implement max priority queue

Again let's test delMax() we just implemented before we move on to the next step.

```
public static void main( String[] args ) {  
    MaxPQ queue = new MaxPQ(1);  
    int[] values = new int[] { 5, 6, 7, 8, 1, 2, 3, 4 };  
    for( int v : values )  
        queue.insert(v);  
    while( queue.count > 0 )  
        System.out.print(" " + queue.delMax());  
}
```

Implement max priority queue

The result should look like following:



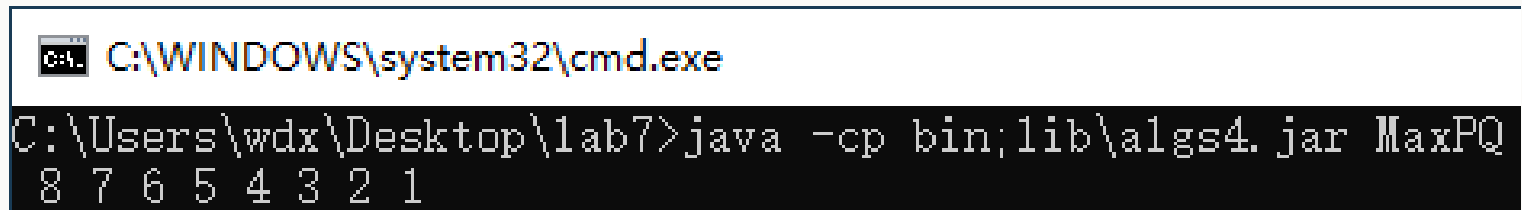
```
C:\WINDOWS\system32\cmd.exe
C:\Users\wdx\Desktop\lab7>java -cp bin;lib\algs4.jar MaxPQ
8 7 6 5 4 3 2 1
```

Now we know we write our `delMax()` correctly.

We may still need to write a `"size()"` method to return the `"count"` variable and write more tests. The details are omitted here.

Implement max priority queue

The result should look like following:



```
C:\WINDOWS\system32\cmd.exe
C:\Users\wdx\Desktop\lab7>java -cp bin;lib\algs4.jar MaxPQ
8 7 6 5 4 3 2 1
```

Now we know we write our `delMax()` correctly.

We may still need to write a `"size()"` method to return the `"count"` variable and write more tests. The details are omitted here.

Implement heapsort

Since we have implemented a binary heap here. It is reasonable that we also implement heapsort. Add another heapSort method here:

```
public static void heapSort( int[] array ) {  
}
```

The basic idea is that we build a heap with the array. Then each time we extract the largest element and put it at the end of the array.

Implement heapsort

But here is one problem: we don't want to sort $[1, \text{array.length}-1]$ in the array. We want to sort $[0, \text{array.length}-1]$ in the array.

One way to solve this is to add a layer of abstraction of index as the algs4 do.

But here we will just reimplement sink operation.

Implement heapsort

Static version of sink, just add -1 to every array access.

```
private static void sink( int[] array, int count, int idx )
{
    int value = array[idx-1];
    while( idx*2 <= count ) {
        int id2 = (idx*2+1 <= count && array[idx*2-1]<array[idx
            *2]) ? (idx*2+1) : (idx*2);
        if( array[id2-1] > value ) {
            array[idx-1] = array[id2-1];
            idx = id2;
        } else
            break;
    }
    array[idx-1] = value;
}
```

Implement heapsort

Finally implement our heap sort:

```
public static void heapSort( int[] array ) {  
    if( array.length <= 1 )  
        return;  
    int count = array.length;  
    // build a heap:  
    for( int k = count/2; k >= 1; k -- )  
        sink( array, count, k);  
    // do sorting:  
    while( count > 1 ) {  
        int tmp = array[count-1];  
        array[count-1] = array[0];  
        array[0] = tmp;  
        count --;  
        sink( array, count, 1 );  
    }  
}
```

Don't forget to write test to your sorting algorithm.

In class exercise: Computational number theory

Finish exercise 2.4.25 in "Algorithms FOURTH EDITION":

Write a program `CubeSum.java` that prints out all integers of the form $a^3 + b^3$ where a and b are integers between 0 and N in sorted order, without using excessive space. That is, instead of computing an array of the N^2 sums and sorting them, build a minimum-oriented priority queue, initially containing $(0^3, 0, 0)$, $(1^3, 1, 0)$, $(2^3, 2, 0)$, . . . , $(N^3, N, 0)$. Then, while the priority queue is nonempty, remove the smallest item $(i^3 + j^3, i, j)$, print it, and then, if $j < N$, insert the item $(i^3 + (j + 1)^3, i, j+1)$.

In class exercise: Index priority-queue implementation

Finish exercise 2.4.33 in "Algorithms FOURTH EDITION":

Implement the basic operations in the index priority-queue API as follows:

Change `pq[]` to hold indices, add an array `keys[]` to hold the key values, and add an array `qp[]` that is the inverse of `pq[]` — `qp[i]` gives the position of `i` in `pq[]` (the index `j` such that `pq[j]` is `i`). Then modify the code in Algorithm 2.6 to maintain these data structures. Use the convention that `qp[i] = -1` if `i` is not on the queue, and include a method `contains()` that tests this condition.

You need to modify the helper methods `exch()` and `less()` but not `sink()` or `swim()`.