**4)** The worst-case number of comparisons performed during an insertion into a max heap can be reduced to O(loglog *n*) by performing a binary search on the path from the new leaf to the root. This does not affect the number of data moves though. Write an insertion algorithm that uses this strategy. Based on your experiments, what can you say about the value of this strategy over the one used in following program?

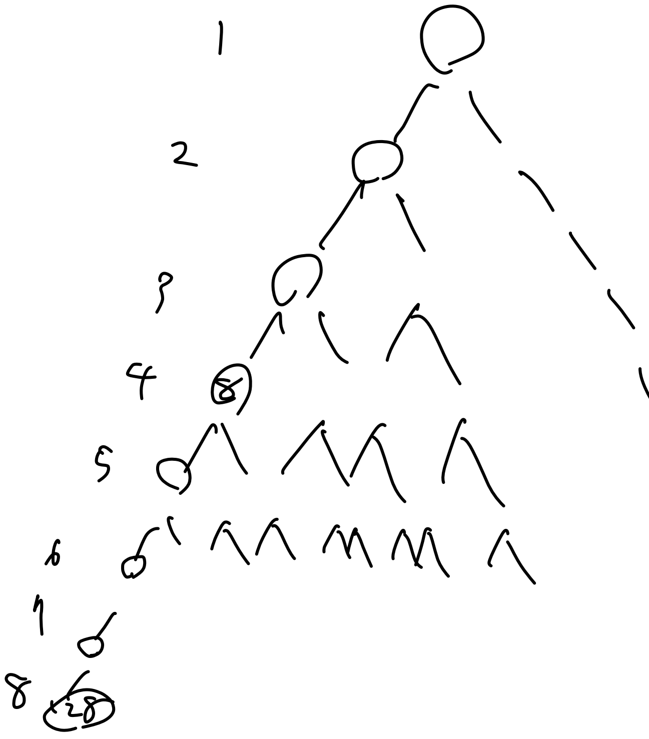
```
template <class T>
void MaxHeap<T>::Push(const T& e)
{ // Insert e into the max heap.
      if (heapSize == capacity) {// double the capacity
            ChangeSize 1D (heap, capacity, 2*capacity);
             capacity * = 2;

}

        int currentNode = ++heapSize;
       while (currentNode !=1 && heap [currentNode/2] <e)
       {// bubble up
            heap [currentNode ] = heap [currentNode /2]; // move parent down
            currentNode /= 2;

}

       heap [currentNode ] = e;
}
```



```
int h = log currentNode ;
int top = 1 , bottom = currentNode

=> while( bottom != top) {
        if (heap[bottom / 2^((h+top)/2)] < e)
            bottom /= 2^((h+top)/2) , h /= 2
        else if (heap[bottom / 2^((h+top)/2)] > e)
            top *= 2^((h+top)/2 - 1) , h /= 2

    currentNode = bottom ;
    heap[currentNode] = e
```

USE the binary search to find the position to insert e. Although the total moves are not change, we can find the right position in O(log h) which means O(loglog n). It's better.