Heap Implementation

HeapInt.h

```
class HeapInt { // Heap of type int
        int n; // current number of elements
        int nmax; // Max number of elements
        int * array;
  public:
     HeapInt(int maxSize);
      void insert(int key);
      int removeMin();
      ~HeapInt();
}
#define ileft(p)  (2*(p)+1)
#define iright(p) (2*(p)+2)
#define iparent(ch) (((ch)-1)/2)
```

```
HeapInt::HeapInt(int maxSize)
    n = 0;
    nMax = maxSize;
    array = new int[maxSize);
}
HeapInt::~HeapInt()
     delete [] array;
}
```

```
void
HeapInt::insert(int key) {
        // insert keys in heap
       assert(n < nmax);
       //add key in the next available position in array
       array[n]=key;
       n++;
       // apply upheap. Swap parent and child atating at inserted
      // node up until we reach root or not necessary.
      int child = n-1;
      int parent = iparent(child);
```

```
        while ( child > 0) {
            if (array[child] > array[parent]) {
                // No need to swap. Stop upheap
                break;
            }
            // We need to swap parent anc hild
             int tmp= array[child];
            array[child] = array[parent];
            array[parent] = tmp;
             child = parent;
            parent = iparent(child);
        } // while
} // insert
```

```
 int
HeapInt::removeMin() {
    // returns minimum key after removing key from heap.
    assert(n>0);
    // Get min key at index 0
   int minkey = array[0];
   n--;
   if (n==0)
      // heap is empty. No need to fix heap.
     return minkey;
   }
```

```
// Move last element in heap to the top
array[0] = array[n];
// Fix heap doing down-heap
int parent = 0;
int left = ileft(parent);
Int right = iright(parent);
while ( left < n) {
  // Determine smallest child
  int minChild = left;
   if (right < n &&   array[right] < array[left]) {
          minchild = right;
   }
   // Check if we need to swap
   if ( array[parent] < array[minChild]) {
       // NO need to swap;
     break;
  }
```

```
            // we need to swap parent and minchild
            int tmp = array[minchild];
            array[minchild]=array[parent];
            array[parent]=tmp;
            // continue going downheap
            parent = minchild;
            left = ileft(parent);
            right = iright(parent);
        } // while
        return minkey;
}
// insert takes O(logn) in the worst case.
// removemin() takes O(logn) in the worst case.
```

```
void heapSort(int * array, int n) {
     // It isorts an array of int using heapsort.
    // It uses the priority queue sort but now using a Heap data structure
    // Create a heap of max size n
     HeapInt heap(n);
    // Phase 1. Put the elements in the array inside the heap
    // Each insert takes O(logn). n insert operations take O(nlogn)
    for (int I = 0; I < n; I++) {
         heap.insert(array[i]);
    }
   // Phase 2. Put the elements back from the heap into the array using
   // heap. Removemin(). A removemin()  operation takes O(logn). N removemins' take O(nlogn)
  for (int I = 0; I < ; i++) {
       array[i] = heap.removeMin();
 }
  // array is sorted.
}
// HeapSort takes O(nlogn) + O(nlogn) = O(nlogn)
```