CS-250

Data Structures and Algorithms

Lab 13

Binary Heap

Name: Mahum Samar

Class: BSCS-9B

CMS ID: 290647

# Code:

|  |
| --- |
| #include <iostream> #include <cmath>   using namespace std;   class Heap { public:  int sizeOfHeap = 0;   int LeftChild(int index) {  *//method to return keft child* return (2 \* index);  }   int RightChild(int index) {  *//method to return right child* return ((2 \* index) + 1);  }   void Swap(int \*x, int \*y) {  *//method to swap the two values of the array by pointing to the address of the array index* int temp = \*x;  \*x = \*y;  \*y = temp;  }   void MaxHeapify(int array[], int index, int sizeOfArray) { *//method to place one value at its logical position* int left = LeftChild(index);  int right = RightChild(index);  int largestIndex;  *//conditions for checking index having largest value* if (left <= sizeOfArray && array[left] > array[index]) {  largestIndex = left;  } else {  largestIndex = index;  }  if (right <= sizeOfArray && array[right] > array[largestIndex]) {  largestIndex = right;  }  if (largestIndex != index) {  Swap(&array[index], &array[largestIndex]);  *// cout << "swapping index"; //recursive call* MaxHeapify(array, largestIndex, sizeOfArray);  }   }   void InsertMaxHeapify(int array[], int index, int sizeOfArray) {  *//method to max heapify heap during insertion* if (index >= 1) {  int parentIndex = floor(index / 2); *//finding parent* if (index > 1 && array[parentIndex] < array[index]) {  *//if parent has less value than swap* Swap(&array[parentIndex], &array[index]);  *//recursive call* InsertMaxHeapify(array, parentIndex, sizeOfArray);  }  }  }   void BuildMaxHeap(int array[], int sizeOfArray) {   *//sizeOfArray/2 because comparison start from the left most subtree  //to avoid the condition to be checked for the leaf nodes  //the elements in range from array[n/2 + 1,...,n] are leaf nodes* for (int index = sizeOfArray / 2; index > 0; index--) {  *//max heapify the current element* MaxHeapify(array, index, sizeOfArray);  }  }   void InsertValueInHeap(int array[], int sizeOfArray, int value) {  *//method to insert value in heap* sizeOfHeap++;  if (sizeOfHeap < sizeOfArray) {  array[sizeOfHeap] = value; *// heapify value* InsertMaxHeapify(array, sizeOfHeap, sizeOfArray);   } else {  cout << "Array max length reached. More values can't be added.\n";  return;  }   }   int FindMax(int array[]) {  *//method to return maximum element in heap* if (sizeOfHeap != 0) {  return array[1];  } else {  return -1;  }  }   bool IsEmpty(int array[]) {  *//method to check either heap is empty or not* return sizeOfHeap == 0;  }   int Size(int array[]) {  *//method to return size of heap* return sizeOfHeap;  }   void BuildHeap(int array[], int sizeOfArray) {  *//method to build heap from the given array* BuildMaxHeap(array, sizeOfArray);  }   void HeapSort(int array[]) {  *//method to sort heap* if (sizeOfHeap == 0) {  return;  } else {  *//recursive call case* Swap(&array[1], &array[sizeOfHeap]);  sizeOfHeap--;  *//max heapify 1st element* MaxHeapify(array, 1, sizeOfHeap);  HeapSort(array);  }   }   void DeleteMax(int array[], int sizeOfArray) {  *//method to delete maximum value from heap* array[1] = array[sizeOfHeap];  sizeOfHeap--;  *//heapify 1t element* MaxHeapify(array, 1, sizeOfArray);  }   void PrintHeap(int array[]) {  *//method to print heap* for (int i = 1; i <= sizeOfHeap; ++i) {  cout << array[i] << "\t";  }  cout << endl;  }  };   int main() {   Heap \*heap = new Heap();  int heapArray[30];   int inputArray[] = {3, 2, 4, 5, 6, 1};  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  cout << "The original values are:\n";   int lengthOfInputArray = sizeof(inputArray) / 4;   for (int i = 0; i < lengthOfInputArray; i++) {  cout << inputArray[i] << "\t";  }  for (int i = 0; i < lengthOfInputArray; i++) {  *//inserting values in heap* heap->InsertValueInHeap(heapArray, lengthOfInputArray + 1, inputArray[i]);  }  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  cout << "\nArray values after inserting into heap:\n";  heap->PrintHeap(heapArray);  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  int maxValue = heap->FindMax(heapArray);  cout << "\nMaximum value of heap is:\t" << maxValue << "\n";  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  cout << "\nDeleting the max value from heap.";  heap->DeleteMax(heapArray, heap->Size(heapArray));   cout << "\nArray values after deleting from heap:\n";  heap->PrintHeap(heapArray);  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";  cout << "\nValues of array after using heap sort:\n";  int heapLength = heap->sizeOfHeap;  heap->HeapSort(heapArray);  heap->PrintHeap(heapArray);  for (int i = 1; i <= heapLength; i++) {  cout << heapArray[i] << "\t";  }  cout << "\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";   return 0; } |

# Output:

