```
// Name: Connor Marler
// Instructor: Gerry Howser
// Date: 10/13/2022
// Project 2 Heap Sort Analysis

#include<iostream>
#include<vector>
#include <chrono>
#include <cassert>

using namespace std;

//Source: https://www.geeksforgeeks.org/heap-sort/

#include <iostream>
using namespace std;


//Purpose: To heapify a subtree rooted with node i which is an index in arr[].
//PreCondition: takes in a templated array and, an int of the length n and node
//              index of i that is also an interger
//PostCondition: A heap is created
//Invariant: no loop
template<typename T>
void heapify(T arr[], int N, int i)
{
      int largest = i; // Initialize largest as root
      int l = 2 * i + 1; // left = 2*i + 1
      int r = 2 * i + 2; // right = 2*i + 2


      if (l < N && arr[l] > arr[largest]) // If left child is larger than root
            largest = l;

      if (r < N && arr[r] > arr[largest]) // If right child is larger than largest
so far
            largest = r;

      if (largest != i) { // If largest is not root
            swap(arr[i], arr[largest]);
    heapify(arr, N, largest);// Recursively heapify the affected sub-tree
      }
}

//Purpose: Main function to do heap sort
//PreCondition: takes in a templated array and an int of the size of the array
//PostCondition: The array taken in is sorted
//Invariant: For all 1 <= i < (N-1), arr[i] >= arr[i-1]
template<typename T>
void heapSort(T arr[], int N)
{
      for (int i = N / 2 - 1; i >= 0; i--) // Build heap (rearrange array)
            heapify(arr, N, i);

      for (int i = N - 1; i > 0; i--) { // One by one extract an element from heap
            swap(arr[0], arr[i]); // Move current root to end
    assert(arr[i] >= arr[i-1]);
            heapify(arr, i, 0); // call max heapify on the reduced heap
      }
```

```
}

//Purpose: Template function to print array
//PreCondition: takes in an array and a value of the size of the array
//PostCondition: outputs the array givens
template<typename T>
void PrintArray(T arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n\n";
}


//Purpose: take in an array and fill it with a predetermined element set
//PreCondition: must receive arr[size] and a string value determining how to fill
the set
//PostCondition: must fill arr[size] with the values in the appropriate as
determined by preSortOrder
void populateArray(int *arr, const int size, const string preSortOrder)
{

  // This if else tree could also be re-written as a switch case
  if(preSortOrder == "assPreSortOrder")
  {
    for(int i = 0; i <= size; i++)
      {
        arr[i] = i + 1;
      }
  }
  else if(preSortOrder == "decPreSortOrder")
  {
    for(int j = 0; j <= size; j++)
      {
        arr[j] = size - j;
      }
  }
  else if(preSortOrder == "randPreSortOrder")
  {
    for(int k = 0; k <= size; k++)
      {
        arr[k] = rand() % size + 1;
      }
  }
  else // error handling
  {
    cout << "ERROR: Something broke. Please quit and try again." << endl;
  }
}

int main()
{

  // These are what we will use for the time measurement
  using chrono::high_resolution_clock;
  using chrono::duration_cast;
  using chrono::duration;
  using chrono::nanoseconds;
  srand (time(NULL));
  const int SIZE = 500;
  string preSortOrder;
  int innerLoop = 1000;
  int outerLoop = 10;
  int averageTime = 0;
  int time = 0;


  int mainArray[SIZE]; // This the hard, original copy of the array
  int workingArray[SIZE]; // This is the copy used to work with and innerLoop
through


  cout << "HEAP SORT AVG TIMES" << endl;
  for(int i = 0; i <= 2; i ++) // loops through each case
  {
    if(i == 0)
    {
      preSortOrder = "assPreSortOrder";
      cout << endl << "Ascending Order Time:" << endl;
    }
    else if(i == 1)
    {
      preSortOrder = "decPreSortOrder";
      cout << endl << "Descending Order Time:" << endl;
    }
    else if(i == 2)
    {
      preSortOrder = "randPreSortOrder";
      cout << endl << "Random Order Time:" << endl;
    }

    populateArray(mainArray, SIZE, preSortOrder);
    averageTime = 0;

    for(int k = 0; k < outerLoop; k++) // takes 10 data point
    {
      averageTime = 0;
      for(int j = 0; j < innerLoop; j++) // perform the sort 1000 times and take an
average for one data point
      {
        for (int a = 0; a < SIZE; a++) // This copies over the original array onto
a working copy
        {
          workingArray[a] = mainArray[a];
        }
        //cout << "Before Sort" << endl;
        //PrintArray(workingArray, SIZE);
        auto time1 = high_resolution_clock::now(); // take initial time
        heapSort(workingArray, SIZE);  // do the sort
        auto time2 = high_resolution_clock::now(); // time the after time
        //cout << "After Sort" << endl;
        //PrintArray(workingArray, SIZE);
        auto nanoSeconds = duration_cast<nanoseconds>(time2 - time1); // find the
difference of the times
        time = nanoSeconds.count();  // convert to an int
        averageTime += time;
      }
      averageTime /= innerLoop;
      cout << "Trial # " << k + 1 << ": " << averageTime << " ns" << endl;
    }
  }
}
```