Sorting Lab

(The Grand Quiz)
Due by Wednesday 27th June 2017

OBJECTIVE

The GRAND objective is to make you compare the fundamental sorting algorithms HEAPSORT, MERGE-SORT, QUICK-SORT, AVL-SORT, Bubble-Sort, Selection-Sort and

DISK-SORT (also called External memory Sorting).

Each part from Task 1 to Task 7 carries 10 points.

TASK 1

First play with PRIORITY_QUEUE (a max heap) in the STL library, here is an example

```
#include <queue>
#include<vector>
#include<iostream>
#include<stdlib.h>
using namespace std;


int main()
{
    // Example 1 (Internally you can safely assume "priority_queue" a heap data structures).

    priority_queue<int> PQ;
    PQ.push(20);
    PQ.push(30);
    PQ.push(40);
    PQ.push(10);
    PQ.push(5);

    while(!PQ.empty())
    {
        cout<<PQ.top()<<" ";
        PQ.pop();
    }

   return 0;
}
```

TASK 2

```cpp
// Writing your own Custom based Compare

class Compare
{
    public:
    bool operator()(int &a, int & b)
    {
        return a>=b;
    }
};

int main()
{
    // If you want to internally use your own compare function
    priority_queue<int, vector<int>, Compare> PQ;
    PQ.push(20);
    PQ.push(30);
    PQ.push(40);
    PQ.push(10);
    PQ.push(5);

    while(!PQ.empty())
    {
        cout<<PQ.top()<<" ";
        PQ.pop();
    }

    return 0;
}
```

```cpp
template<class T>
void HeapSort(vector<T> & V)
{
    // The Idea is to add inside the Priority_Queue all the values and then Pop
    // all the values one by one V.size() times and keep adding in V and we have the solution


    /*
        // You may try with the following syntax too:

        priority_queue<int> PQ;
                // The remaining code remain the same
                // (this will generate all values in the reverse order as by default
                // PQ is a MAX-Heap)
    */


    priority_queue<int, vector<int>, Compare> PQ;

    for(int vi=0; vi<V.size(); vi++)
    {
        PQ.push(V[vi]);
    }
    V.clear();  // Clearing the previous Vector so that we can have
                // the memory free and we will assign values popping from the PQ (Heap)
    while(!PQ.empty())
    {
        V.push_back(PQ.top());
        PQ.pop();
    }
}
```

TASK 3

FOR TESTING YOUR HEAP-SORT YOU NEED TO WRITE THESE TWO FUNCTIONS FIRST

*main.cpp

```
104
105
106  void Randomized_Init(vector<int>& V)
107  {
108      for(int i=0; i<V.size(); i++)
109      {
110          V[i] = rand()%100;
111      }
112  }
113  void Vector_Print(vector<int>& V)
114  {
115      for(int i=0; i<V.size(); i++)
116      {
117          cout<<V[i]<<" ";
118      }
119
120      cout<<endl<<endl<<endl;
121  }
122
```

NOW ADD THIS MAIN FUNCTION

```
176
177  int main()
178  {
179      vector<int> Values(20);
180      Randomized_Init(Values);
181      cout<<"Before Sorting...: "<<endl;
182      Vector_Print(Values);
183
184      // HeapSort(Values);
185      // MergeSort(Values);
186      // QuickSort(Values);
187
188
189      cout<<"After Sorting...: "<<endl;
190      Vector_Print(Values);
191  }
192
193
```

TASK 4

WRITE THIS BUBBESORT FUNCTION

```
193  //-------------------------------------------------------
194
195  bool SwappingAllTheWay(vector<T>& V)
196  {
197      bool ChangeHappen = false;
198      for(int i=0; i<V.size()-1; i++)
199      {
200          if(V[i]>V[i+1])
201          {
202              swap(V[i], V[i+1]);
203              ChangeHappen = true;
204          }
205      }
206      return ChangeHappen;
207  }
208
209
210  void BubbleSort(vector<T>& V)
211  {
212      while(SwappingAllTheWay(V))
213      {
214          // This loop will break until the array is sorted and then SwappingAllTheWay
215          // will return false;
216      }
217  }
218
```

TASK 5

Write this SELECTIONSORT Algorithm and add the Switch case in the main so that user can enter which sorting algorithm one wants to run.

```
263
264  // Selection Sort
265  template<class T>
266  bool FindMinRange(vector<T>& V, int si, int ei)
267  {
268      bool mi = si;
269      for(int i=si+1; i<=ei; i++)
270      {
271          if(V[mi]>V[i])
272          {
273              mi = i;
274          }
275      }
276      return mi;
277  }
278
279  template<class T>
280  void SelectionSort(vector<T>& V)
281  {
282      for(int i=0; i<V.size()-1; i++)
283      {
284          // Find the minimum from i to the range up to V[last]
285          int mi = FindMinRange(V, i, V.size()-1);
286          swap(V[i], V[mi]);
287      }
288  }
289
290  //-------------------------------------------------------
291
```

TASK 6

AVL Sort

```
// AVL SORT
template<class T>
void AVLSort(vector<T> & V)
{
    map<int, int> M;
    for(int vi=0; vi<V.size(); vi++)
    {
        M[vi] = vi;
    }
    V.clear();
    for(map<int, int>::iterator i=M.begin(); i!=M.end(); i++)
    {
        V.push_back(i->first);
    }
}
```

TASK 7

Write the code for QuickSort.

The plot is given below

```
int Partition(vector<T>& V, int left, int right, int pi)
{   // Moving all the greater values at V[pi] to the right side and lesser values to left side.
    int pivot = V[pi];
    int i=left-1;
    int j=left;
    swap(V[pi], V[right]); // move pivot element to the end
    pi = right;
    int range=right-left+1;
    while(j <= range)
    {
        if(V[j] >= V[pi])
        {
            i = i+1;
            swap(V[i], V[j]);
        }
        j++;
    }
    swap(V[i+1], V[pi]);
    return i+1;

}
template<class T>   // In memory Sorting algorithm
void QuickSortRec(vector<T> & V, int si, int ei)
{
    if(si>=ei)
        return;   // The Size of the range is just one... So no need to do anything

    int pi = ( rand()%(ei-si+1) )  + si; // Generating a random Index between si to ei
    pi = Partition(V, si, ei, pi);
    QuickSortRec(V, si, pi-1);
    QuickSortRec(V, pi+1, ei);
}
```

```
template<class T>
void QuickSort(vector<T> & V)
{
    QuickSortRec(V, 0, V.size()-1);
}
```

TASK 8 **(20 points)**

Implement DISK-SORT

DISK-SORT (also called External memory Sorting).

https://pdfs.semanticscholar.org/.../14d9e7cc3f05e943934e8e47...

https://en.wikipedia.org/wiki/External_sorting

Here is the main task 60 points

After writing all these sorts, write a small program (using google) to measure the duration of a program being executed.

Then add that same functionality into your sorting main program such that it should Sort 1 GB data file loaded once memory and then sort using all the sorting algorithms one by one and see how the different algorithm duration is.

All the sorting algorithms I wrote are untested (I just wrote them down one by one, without testing) if there is any error in the code you have to remove it. Also you are allowed to take any code from google (JUST MAKE SURE YOU PUT THE REFERENCE IN THE CODE, that where you took the code from).