DATA STRUCTURES & ALGORITHMS

PRACTICE
Week 2 - Sort

Lecture: **MCS. Mai Thai Quoc**
Email: **maithaiquoc@gmail.com**

GIA DINH UNIVERSITY

Practice 1

QuickSort (pivot = last)

(1)

```cpp
#include <bits/stdc++.h>
using namespace std;
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)
    {
        if (arr[j] < pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

Practice 1

QuickSort (pivot = last)

(2)

```
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

Practice 2

QuickSort (pivot = mid, no recursion)

```
#include <iostream>
#include <math.h>
using namespace std;
int a[] = {2, 5, 7, 9, 4, 8, 20};
int n = sizeof(a)/sizeof(a[0]);

void Quicksort(int l, int r) {
    int i = l;
    int j = r;
    int Pivot = (i + j) / 2;
    while (i <= j) {
        while (a[i] < a[Pivot]) i++;
        while (a[j] > a[Pivot]) j--;
        if (i <= j) {
            int tmp = a[i];
            a[i] = a[j];
            a[j] = tmp;
            i++;
            j--;
        }
    }
    if (j>l) Quicksort(l,j);
    if (i<r) Quicksort(i,r);
 }

void printAll(){
    for (int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<endl;
}

void main(){
    Quicksort(0,n-1);
    printAll();
}
```

Practice 3
MergeSort
(1)

```c
#include<stdlib.h>
#include<stdio.h>
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
```

Practice 3
MergeSort
(2)

```
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Practice 4
CountingSort

```
#include<bits/stdc++.h>
#include<string.h>
using namespace std;
#define RANGE 255
void countSort(char arr[])
{
    char output[strlen(arr)];
    int count[RANGE + 1], i;
    memset(count, 0, sizeof(count));
    for(i = 0; arr[i]; ++i)
        ++count[arr[i]];
    for (i = 1; i <= RANGE; ++i)
        count[i] += count[i-1];
    for (i = 0; arr[i]; ++i)
    {
        output[count[arr[i]]-1] = arr[i];
        --count[arr[i]];
    }

    for (i = 0; arr[i]; ++i)
        arr[i] = output[i];
}

int main()
{
    char arr[] = "HutechUniversityHutech";
    countSort(arr);

    cout<< "Sorted character array is " << arr;
    return 0;
}
```

Practice 5
CountingSort (negative number)

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

void countSort(vector <int>& arr)
{
    int max = *max_element(arr.begin(), arr.end());
    int min = *min_element(arr.begin(), arr.end());
    int range = max - min + 1;

    vector<int> count(range), output(arr.size());
    for(int i = 0; i < arr.size(); i++)
        count[arr[i]-min]++;

    for(int i = 1; i < count.size(); i++)
          count[i] += count[i-1];

    for(int i = arr.size()-1; i >= 0; i--)
    {
         output[ count[arr[i]-min] -1 ] = arr[i];
              count[arr[i]-min]--;
    }

    for(int i=0; i < arr.size(); i++)
           arr[i] = output[i];
}

void printArray(vector <int> & arr)
{
    for (int i=0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << "\n";
}

int main()
{
    vector<int> arr = {-5, -10, 0, -3, 8, 5, -1, 10};
    countSort (arr);
    printArray (arr);
    return 0;
}
```

Practice 6 (1)

```
#include<iostream>
#include<limits.h>
using namespace std;

#define n 4

// A min heap node
struct MinHeapNode
{
    int element; // The element to be stored
    int i; // index of the array from which the element is taken
    int j; // index of the next element to be picked from array
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode *x, MinHeapNode *y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(MinHeapNode x) { harr[0] = x;  MinHeapify(0); }
};
```

Practice 6 (2)

```
// This function takes an array of arrays as an argument and
// All arrays are assumed to be sorted. It merges them together
// and prints the final sorted output.
int *mergeKArrays(int arr[][n], int k)
{
    int *output = new int[n*k];  // To store output array

    // Create a min heap with k heap nodes.  Every heap node
    // has first element of an array
    MinHeapNode *harr = new MinHeapNode[k];
    for (int i = 0; i < k; i++)
    {
        harr[i].element = arr[i][0]; // Store the first element
        harr[i].i = i;  // index of array
        harr[i].j = 1;  // Index of next element to be stored from array
    }
    MinHeap hp(harr, k); // Create the heap

    // Now one by one get the minimum element from min
    // heap and replace it with next element of its array
    for (int count = 0; count < n*k; count++)
    {
        // Get the minimum element and store it in output
        MinHeapNode root = hp.getMin();
        output[count] = root.element;

        // Find the next elelement that will replace current
        // root of heap. The next element belongs to same
        // array as the current root.
        if (root.j < n)
        {
            root.element = arr[root.i][root.j];
            root.j += 1;
        }
        // If root was the last element of its array
        else root.element =  INT_MAX; //INT_MAX is for infinite

        // Replace root with next element of array
        hp.replaceMin(root);
    }

    return output;
}
```

Practice 6 (3)

```
// FOLLOWING ARE IMPLEMENTATIONS OF STANDARD MIN HEAP METHODS
// FROM CORMEN BOOK
// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
    harr = a;  // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root at given index
// This method assumes that the subtrees are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size && harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(&harr[i], &harr[smallest]);
        MinHeapify(smallest);
    }
}

// A utility function to swap two elements
void swap(MinHeapNode *x, MinHeapNode *y)
{
    MinHeapNode temp = *x;  *x = *y;  *y = temp;
}
```

Practice 6 (4)

```
// A utility function to print array elements
void printArray(int arr[], int size)
{
   for (int i=0; i < size; i++)
       cout << arr[i] << " ";
}

// Driver program to test above functions
int main()
{
    // Change n at the top to change number of elements
    // in an array
    int arr[][n] =  {{1, 3, 5, 7},
                     {2, 4, 6, 8},
                     {0, 9, 10, 11}};
    int k = sizeof(arr)/sizeof(arr[0]);

    int *output = mergeKArrays(arr, k);

    cout << "Merged array is " << endl;
    printArray(output, n*k);

    return 0;
}
```