Sorting Algorithms in C++ ✍■

Bubble Sort

```
#include <iostream>
using namespace std;
void bubbleSort(int arr[], int n) {
    for(int i=0; i<n-1; i++) {
        for(int j=0; j<n-i-1; j++) {
            if(arr[j] > arr[j+1]) swap(arr[j], arr[j+1]);
        }
    }
}
```

Selection Sort

```
#include <iostream>
using namespace std;
void selectionSort(int arr[], int n) {
    for(int i=0; i<n-1; i++) {
        int minIndex = i;
        for(int j=i+1; j<n; j++) {
            if(arr[j] < arr[minIndex]) minIndex = j;
        }
        swap(arr[i], arr[minIndex]);
    }
}
```

Insertion Sort

```
#include <iostream>
using namespace std;
void insertionSort(int arr[], int n) {
    for(int i=1; i<n; i++) {
        int key = arr[i];
        int j = i-1;
        while(j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}
```

Merge Sort

```
#include <iostream>
using namespace std;
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int L[n1], R[n2];
    for(int i=0; i<n1; i++) L[i] = arr[l+i];
    for(int j=0; j<n2; j++) R[j] = arr[m+1+j];
    int i=0, j=0, k=l;
    while(i<n1 && j<n2) {
        if(L[i] <= R[j]) arr[k++] = L[i++];
        else arr[k++] = R[j++];
```

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

Quick Sort

```
#include <iostream>
using namespace std;
int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low-1;
    for(int j=low; j<high; j++) {
        if(arr[j] < pivot) 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);
    }
}
```

Heap Sort

```
#include <iostream>
using namespace std;
void heapify(int arr[], int n, int i) {
    int largest = i, l = 2*i+1, r = 2*i+2;
    if(l < n && arr[l] > arr[largest]) largest = l;
    if(r < n && arr[r] > arr[largest]) largest = r;
    if(largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}
void heapSort(int arr[], int n) {
    for(int i=n/2-1; i>=0; i--) heapify(arr, n, i);
    for(int i=n-1; i>=0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}
```

Counting Sort

```
#include <iostream>
using namespace std;
```

```
void countingSort(int arr[], int n, int k) {
    int count[k+1] = {0}, output[n];
    for(int i=0; i<n; i++) count[arr[i]]++;
    for(int i=1; i<=k; i++) count[i] += count[i-1];
    for(int i=n-1; i>=0; i--) output[--count[arr[i]]] = arr[i];
    for(int i=0; i<n; i++) arr[i] = output[i];
}
```

Radix Sort

```
#include <iostream>
using namespace std;
int getMax(int arr[], int n) {
    int mx = arr[0];
    for(int i=1; i<n; i++) if(arr[i] > mx) mx = arr[i];
    return mx;
}
void countingSortRadix(int arr[], int n, int exp) {
    int output[n], count[10] = {0};
    for(int i=0; i<n; i++) count[(arr[i]/exp)%10]++;
    for(int i=1; i<10; i++) count[i] += count[i-1];
    for(int i=n-1; i>=0; i--) output[--count[(arr[i]/exp)%10]] = arr[i];
    for(int i=0; i<n; i++) arr[i] = output[i];
}
void radixSort(int arr[], int n) {
    int m = getMax(arr, n);
    for(int exp=1; m/exp>0; exp*=10) countingSortRadix(arr, n, exp);
}
```

Bucket Sort

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
void bucketSort(float arr[], int n) {
    vector<float> b[n];
    for(int i=0; i<n; i++) {
        int bi = n*arr[i];
        b[bi].push_back(arr[i]);
    }
    for(int i=0; i<n; i++) sort(b[i].begin(), b[i].end());
    int idx = 0;
    for(int i=0; i<n; i++) {
        for(float x : b[i]) arr[idx++] = x;
    }
}
```

Time & Space Complexities

Algorithm	Best	Average	Worst	Space	Stable?
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	No
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes
Quick Sort	O(n log n)	O(n log n)	O(n^2)	O(log n)	No
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	No
Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(k)	Yes
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)	Yes
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)	Yes