Sorting

Q Bubble Sort(氣泡排序)

▲ 原理解釋

氣泡排序是每次比較相鄰兩元素,若順序錯誤就交換,重複多次直到整體有序。

○ C++ 實作

```
void bubbleSort(std::vector<int>& arr) {
  int n = arr.size();
  bool swapped;
  for (int i = 0; i < n - 1; ++i) {
    swapped = false;
    for (int j = 0; j < n - i - 1; ++j) {
        if (arr[j] > arr[j + 1]) {
            std::swap(arr[j], arr[j + 1]);
            swapped = true;
        }
    }
    if (!swapped) break; // 如果這一輪沒換就提早結束
    }
}
```

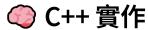
☑ 複雜度分析

Case	Time Complexity	資料情形說明	
Best	O(n)	已排序的資料	
Average	O(n ²)	隨機順序	
Worst	O(n ²)	完全反序	



🔍 原理解釋

每次選出最小的元素放到正確位置。



```
void selectionSort(std::vector<int>& arr) {
  int n = arr.size();
  for (int i = 0; i < n - 1; ++i) {
    int minIdx = i;
    for (int j = i + 1; j < n; ++j) {
        if (arr[j] < arr[minIdx]) {
            minIdx = j;
        }
    }
    std::swap(arr[i], arr[minIdx]);
}</pre>
```

☑ 複雜度分析

Case	Time Complexity	資料情形說明
Best	O(n ²)	任意情況都會完整比較
Average	O(n ²)	隨機順序
Worst	O(n ²)	完全反序

🗱 Insertion Sort(插入排序)

🔍 原理解釋

每次將一個元素插入到前面已排序區間中的適當位置。

○ C++ 實作

```
void insertionSort(std::vector<int>& arr) {
  int n = arr.size();
```

```
for (int i = 1; i < n; ++i) {
  int key = arr[i];
  int j = i - 1;
  while (i \ge 0 \&\& arr[i] > key) {
    arr[i+1] = arr[i];
    --j;
  }
  arr[j + 1] = key;
}
```

☑ 複雜度分析

Case	Time Complexity	資料情形說明	
Best	O(n)	已排序資料	
Average	O(n ²)	隨機順序	
Worst	O(n ²)	完全反序	



🧳 Merge Sort(合併排序)



🔍 原理解釋

採用分治法:將陣列遞迴拆分為小區段,排序後再合併。



○ C++ 實作

```
void merge(std::vector<int>& arr, int left, int mid, int right) {
  std::vector<int> leftArr(arr.begin() + left, arr.begin() + mid + 1);//vector iterator是左
閉右開(包含start不含end)
  std::vector<int> rightArr(arr.begin() + mid + 1, arr.begin() + right + 1);
  int i = 0, j = 0, k = left;
  while (i < leftArr.size() && j < rightArr.size()) {</pre>
    arr[k++] = (leftArr[i] <= rightArr[j]) ? leftArr[i++] : rightArr[j++];</pre>
  }
  while (i < leftArr.size()) arr[k++] = leftArr[i++];</pre>
```

```
while (j < rightArr.size()) arr[k++] = rightArr[j++];
}

void mergeSort(std::vector<int>& arr, int left, int right) {
   if (left < right) {
      int mid = (left + right) / 2;
      mergeSort(arr, left, mid);
      mergeSort(arr, mid + 1, right);
      merge(arr, left, mid, right);
   }
}</pre>
```

📈 複雜度分析

Case	Time Complexity	資料情形說明
Best	O(n log n)	任意資料皆一樣效率
Average	O(n log n)	適合大資料
Worst	O(n log n)	適合幾乎所有資料情況

♦ Quick Sort(快速排序)

🔾 原理解釋

挑選一個 pivot,將小於 pivot 的放左邊,大於的放右邊,遞迴處理。


```
int partition(std::vector<int>& arr, int low, int high) {
  int pivot = arr[high]; // 最右邊為 pivot
  int i = low - 1;
  for (int j = low; j < high; ++j) {
    if (arr[j] < pivot) {
      std::swap(arr[++i], arr[j]);
    }
}</pre>
```

```
std::swap(arr[i + 1], arr[high]);
return i + 1;
}

void quickSort(std::vector<int>& arr, int low, int high) {
  if (low < high) {
    int p = partition(arr, low, high);
     quickSort(arr, low, p - 1);
     quickSort(arr, p + 1, high);
  }
}</pre>
```

☑ 複雜度分析

Case	Time Complexity	資料情形說明
Best	O(n log n)	pivot 恰好分兩邊均等
Average	O(n log n)	隨機資料,pivot 分區合理
Worst	O(n ²)	每次 pivot 為最小/最大(如已排序資料)

🔽 總結比較表

演算法	Best	Average	Worst	穩定性	適合情況
Bubble Sort	O(n)	O(n ²)	O(n ²)	✓ 穩定	小規模資料,有序 最佳
Selection Sort	O(n ²)	O(n ²)	O(n ²)	★ 不穩定	對寫入次數敏感
Insertion Sort	O(n)	O(n ²)	O(n ²)	✓ 穩定	小資料,部分有序
Merge Sort	O(n log n)	O(n log n)	O(n log n)	✓ 穩定	適合大資料排序
Quick Sort	O(n log n)	O(n log n)	O(n²)	★ 不穩定	效率高,空間小