排序定义

```
#include <vector>
#include <iostream>

typedef int Element;
using namespace std;

// 交换两个元素
void swap(Element *a, Element *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}

// 输出序列
void print(vector<Element> &A) {
    for (Element &it : A) {
        cout << it << ' ';
    }
    cout << endl;
}
```

选择排序

```
// 选择排序
void selectionSort(vector<Element> &A) {

    for (int j = 0; j < A.size() - 1; ++j) {
        int min = j;
        for (int i = j + 1; i < A.size(); ++i)
            if (A[i] < A[min])
                min = i;
        if (min != j)
            swap(&A[min], &A[j]);
    }
}
```

插入排序

```
void insertionSort(vector<Element> &A) {
    for (int j = 1; j < A.size(); ++j) {
        Element key = A[j]; // insert A[j] into sorted sequence A[1...j-
1].
        int i = j - 1;
```

```
        while (i >= 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i - 1;
        }
        A[i + 1] = key;
    }
}
```

折半插入排序

```
void binaryInsertionSort(vector<Element> &A) {
    for (int j = 1; j < A.size(); ++j) {
        Element key = A[j];
        int i = j - 1;
        int low = 0, high = i;
        while (low <= high) {
            int m = (low + high) / 2;
            if (key < A[m])
                high = m - 1;
            else
                low = m + 1;
        }
        while (i > high) {
            A[i + 1] = A[i];
            i = i - 1;
        }
        A[i + 1] = key;
    }
}
```

希尔排序

```
void shellSort(vector<Element> A) {
    for (int gap = A.size(); gap > 0; gap /= 2) {
        for (int j = gap; j < A.size(); ++j) {
            Element key = A[j];
            int i = j - gap;
            while (i >= 0 && A[i] > key) {
                A[i + gap] = A[i];
                i -= gap;
            }
            A[i + gap] = key;
        }
    }
}
```

冒泡排序

```
void bubbleSort(vector<Element> &A) {
    for (int i = 0; i < A.size() - 1; ++i) {
        for (int j = i; j < A.size(); ++j) {
            if (A[i] > A[j]) {
                Element tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
            }
        }
    }
}
```

堆排序

```
 // 堆排序
void heapSort(vector<Element> &A) {
    int size = A.size();
    for (int i = size / 2 - 1; i >= 0; --i) {
        heapify(A, size, i);
    }
    for (int i = size - 1; i >= 0; --i) {
        Element tmp = A[0];
        A[0] = A[i];
        A[i] = tmp;
        heapify(A, i, 0);
    }
}


void heapify(vector<Element> &heap, int size, int hole) {
    maxHeapify(heap, size, hole);
}

// 生成最小堆
void minHeapify(vector<Element> &heap, int size, int hole) {
    int child;
    Element tmp = heap[hole];
    for (; hole * 2 + 1 < size; hole = child) {
        child = hole * 2 + 1;
        if (child + 1 != size && heap[child + 1] < heap[child])
            child++;
        if (heap[child] < tmp
                )
            heap[hole] = heap[child];
        else
            break;
    }
    heap[hole] = tmp;
}
```

```
// 生成最大堆
void maxHeapify(vector<Element> &heap, int size, int hole) {
    int child;
    Element tmp = heap[hole];
    for (; hole * 2 + 1 < size; hole = child) {
        child = hole * 2 + 1;
        if (child + 1 != size && heap[child + 1] > heap[child])
            ++child;
        if (heap[child] > tmp)
            heap[hole] = heap[child];
        else
            break;
    }
    heap[hole] = tmp;
}
```

归并排序

```
/**
 * 归并排序
 */
void mergeSort(vector<Element> &A) {
    vector<Element> tmp(A.size());
    mergeSort(A, tmp, 0, A.size() - 1);
}

void mergeSort(vector<Element> &A, vector<Element> &tmp, int left, int
right) {
    if (left < right) {
        int center = (left + right) / 2;
        mergeSort(A, tmp, left, center);
        mergeSort(A, tmp, center + 1, right);
        merge(A, tmp, left, center + 1, right);
    }
}

void merge(vector<Element> &A, vector<Element> &tmp,
        int leftPos, int rightPos, int rightEnd) {
    int leftEnd = rightPos - 1;
    int pos = leftPos;
    int count = rightEnd - leftPos + 1;
    while (leftPos <= leftEnd && rightPos <= rightEnd) {
        if (A[leftPos] <= A[rightPos])
            tmp[pos++] = A[leftPos++];
        else
            tmp[pos++] = A[rightPos++];
    }

    while (leftPos <= leftEnd)
        tmp[pos++] = A[leftPos++];
```

```
    while (rightPos <= rightEnd)
        tmp[pos++] = A[rightPos++];


    for (int i = 0; i < count; ++i, --rightEnd) {
        A[rightEnd] = tmp[rightEnd];
    }
}
```

快速排序

```
// 快速排序
void quickSort(vector<Element> &A) {
    quickSort(A, 0, A.size() - 1);
}

void quickSort(vector<Element> &A, int p, int r) {
    if (p < r) {
        int q = partition(A, p, r);
        quickSort(A, p, q - 1);
        quickSort(A, q + 1, r);
    }
}

int partition(vector<Element> &A, int p, int r) {
    Element x = A[r];
    int i = p - 1;
    for (int j = p; j < r; ++j) {
        if (A[j] <= x) {
            i = i + 1;
            swap(&A[i], &A[j]);
        }
    }
    swap(&A[i + 1], &A[r]);
    return i + 1;
}
```