Bubble Sort

```
void bubbleSort(vector<int>& a) {
    int n = a.size();
    for (int i = 0; i < n-1; ++i)
        for (int j = 0; j < n-1-i; ++j)
            if (a[j] > a[j+1]) swap(a[j], a[j+1]);
}
```

Optimized Bubble (with early exit)

```
void bubbleSortOpt(vector<int>& a) {
    int n = a.size();
    for (int i = 0; i < n-1; ++i) {
        bool swapped = false;
        for (int j = 0; j < n-1-i; ++j)
            if (a[j] > a[j+1]) { swap(a[j], a[j+1]); swapped = true; }
        if (!swapped) break;
    }
}
```

Selection Sort

```
void selectionSort(vector<int>& a) {
    int n = a.size();
    for (int i = 0; i < n-1; ++i) {
        int mn = i;
        for (int j = i+1; j < n; ++j)
            if (a[j] < a[mn]) mn = j;
        swap(a[i], a[mn]);
    }
}
```

Insertion Sort

```
void insertionSort(vector<int>& a) {
    int n = a.size();
    for (int i = 1; i < n; ++i) {
        int key = a[i], j = i-1;
        while (j >= 0 && a[j] > key) { a[j+1] = a[j]; --j; }
        a[j+1] = key;
    }
}
```

Shell Sort

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

Merge Sort

```
void merge(vector<int>& a, int l, int m, int r) {
    int n1 = m-l+1, n2 = r-m;
    vector<int> L(a.begin()+l, a.begin()+m+1), R(a.begin()+m+1, a.begin()+r+1);
    int i=0,j=0,k=l;
    while (i<n1 && j<n2) a[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    while (i<n1) a[k++] = L[i++];
    while (j<n2) a[k++] = R[j++];
}
void mergeSort(vector<int>& a, int l, int r) {
    if (l >= r) return;
    int m = l + (r-l)/2;
    mergeSort(a, l, m);
    mergeSort(a, m+1, r);
    merge(a, l, m, r);
}
```

Quick Sort (Lomuto partition)

```
int partitionLomuto(vector<int>& a, int l, int r) {
    int pivot = a[r], i = l;
    for (int j = l; j < r; ++j)
        if (a[j] < pivot) swap(a[i++], a[j]);
    swap(a[i], a[r]);
    return i;
}
void quickSort(vector<int>& a, int l, int r) {
    if (l < r) {
        int p = partitionLomuto(a, l, r);
        quickSort(a, l, p-1);
        quickSort(a, p+1, r);
    }
}
```

Heap Sort

```
void heapify(vector<int>& a, int n, int i) {
    int largest = i, l = 2*i+1, r = 2*i+2;
    if (l < n && a[l] > a[largest]) largest = l;
    if (r < n && a[r] > a[largest]) largest = r;
    if (largest != i) { swap(a[i], a[largest]); heapify(a, n, largest); }
}
void heapSort(vector<int>& a) {
    int n = a.size();
    for (int i = n/2 - 1; i >= 0; --i) heapify(a, n, i);
    for (int i = n-1; i > 0; --i) { swap(a[0], a[i]); heapify(a, i, 0); }
}
```

Counting Sort (non-negative integers)

```
void countingSort(vector<int>& a) {
    if (a.empty()) return;
    int mx = *max_element(a.begin(), a.end());
    int mn = *min_element(a.begin(), a.end());
    int range = mx - mn + 1;
    vector<int> cnt(range);
    for (int v : a) cnt[v - mn]++;
    for (int i = 1; i < range; ++i) cnt[i] += cnt[i-1];
    vector<int> out(a.size());
    for (int i = a.size()-1; i >= 0; --i) { out[--cnt[a[i]-mn]] = a[i]; }
    a = move(out);
}
```

Radix Sort (LSD, for non-negative integers)

```
void countSortForRadix(vector<int>& a, int exp) {
    int n = a.size();
    vector<int> output(n), cnt(10,0);
    for (int i = 0; i < n; ++i) cnt[(a[i]/exp)%10]++;
    for (int i = 1; i < 10; ++i) cnt[i] += cnt[i-1];
    for (int i = n-1; i>=0; --i) { output[--cnt[(a[i]/exp)%10]] = a[i]; }
    a = move(output);
}
void radixSort(vector<int>& a) {
    if (a.empty()) return;
    int mx = *max_element(a.begin(), a.end());
    for (int exp = 1; mx/exp > 0; exp *= 10) countSortForRadix(a, exp);
}
```

Bucket Sort (for floating point in [0,1) or general with mapping)

```
void bucketSort(vector<float>& a) {
    int n = a.size();
    vector<vector<float>> buckets(n);
    for (float x : a) buckets[int(n*x)].push_back(x);
    for (auto &b : buckets) sort(b.begin(), b.end());
    int idx = 0;
    for (auto &b : buckets) for (float x : b) a[idx++] = x;
}
```

Cocktail Shaker Sort (bidirectional bubble)

```
void cocktailSort(vector<int>& a) {
    int n = a.size();
    bool swapped = true;
    int start = 0, end = n-1;
    while (swapped) {
        swapped = false;
        for (int i = start; i < end; ++i)
            if (a[i] > a[i+1]) { swap(a[i], a[i+1]); swapped = true; }
        if (!swapped) break;
        swapped = false; --end;
        for (int i = end-1; i >= start; --i)
            if (a[i] > a[i+1]) { swap(a[i], a[i+1]); swapped = true; }
        ++start;
    }
}
```

Gnome Sort

```
void gnomeSort(vector<int>& a) {
    int n = a.size(), index = 0;
    while (index < n) {
        if (index == 0 || a[index] >= a[index-1]) ++index;
        else { swap(a[index], a[index-1]); --index; }
    }
}
```

Cycle Sort (minimizes writes)

```
void cycleSort(vector<int>& a) {
    int n = a.size();
    for (int cycleStart = 0; cycleStart < n-1; ++cycleStart) {
        int item = a[cycleStart], pos = cycleStart;
        for (int i = cycleStart+1; i < n; ++i)
            if (a[i] < item) ++pos;
        if (pos == cycleStart) continue;
        while (item == a[pos]) ++pos;
        if (pos != cycleStart) swap(item, a[pos]);
        while (pos != cycleStart) {
            pos = cycleStart;
            for (int i = cycleStart+1; i < n; ++i)
                if (a[i] < item) ++pos;
            while (item == a[pos]) ++pos;
            swap(item, a[pos]);
        }
    }
}
```