Bubble Sort – Explicação geral

O Bubble Sort compara pares de elementos adjacentes e os troca se estiverem fora de ordem. Esse processo é repetido até o vetor estar completamente ordenado, empurrando os maiores elementos para o final a cada passagem.

```
#include <stdio.h>

int main() {
  int vetor[6] = {23, 5, 12, 8, 31, 17}; // Vetor com valores aleatórios

  for (int i = 0; i < 6 - 1; i++) { // Controla o número de passagens
    for (int j = 0; j < 6 - i - 1; j++) { // Compara pares adjacentes
      if (vetor[j] > vetor[j + 1]) { // Se estiver fora de ordem, troca
        int temp = vetor[j]; // Armazena valor temporário
        vetor[j] = vetor[j + 1]; // Troca os elementos
        vetor[j + 1] = temp; // Finaliza a troca
      }
    }
  }

  for (int i = 0; i < 6; i++) { // Exibe vetor ordenado
    printf("%d ", vetor[i]);
  }

  return 0;
}
```

Selection Sort – Explicação geral

O Selection Sort percorre o vetor procurando o menor elemento e o move para a primeira posição. Depois, repete o processo para as posições seguintes até o fim do vetor.

```
#include <stdio.h>

void selectionSort(int vetor[], int tamanho) {
  for (int i = 0; i < tamanho - 1; i++) { // Percorre todo o vetor
    int menor = i; // Assume que o menor valor é o atual

    for (int j = i + 1; j < tamanho; j++) { // Busca menor valor à frente
      if (vetor[j] < vetor[menor]) {
        menor = j; // Atualiza menor encontrado
      }
```

```
      }

      int temp = vetor[i]; // Troca os elementos
      vetor[i] = vetor[menor];
      vetor[menor] = temp;
   }
}

int main() {
   int vetor[6] = {15, 42, 8, 23, 4, 19}; // Vetor base
   selectionSort(vetor, 6); // Chama a função de ordenação

   for (int i = 0; i < 6; i++) { // Exibe vetor
      printf("%d ", vetor[i]);
   }

   return 0;
}
```

Insertion Sort – Explicação geral

O Insertion Sort percorre o vetor da esquerda para a direita, inserindo cada elemento na posição correta da parte que já está ordenada à esquerda.

```
#include <stdio.h>

void insertionSort(int vetor[], int tamanho) {
   for (int i = 1; i < tamanho; i++) { // Começa do segundo elemento
      int chave = vetor[i]; // Valor a ser inserido
      int j = i - 1;

      while (j >= 0 && vetor[j] > chave) { // Move valores maiores para a direita
         vetor[j + 1] = vetor[j];
         j--;
      }

      vetor[j + 1] = chave; // Insere chave na posição correta
   }
}

int main() {
```

```
  int vetor[6] = {27, 3, 18, 45, 12, 9}; // Vetor base
  insertionSort(vetor, 6); // Chama função de ordenação

  for (int i = 0; i < 6; i++) { // Exibe resultado
    printf("%d ", vetor[i]);
  }

  return 0;
}
```

Merge Sort – Explicação geral

O Merge Sort divide recursivamente o vetor em metades, ordena cada uma delas e as junta (merge) de forma ordenada.

```
#include <stdio.h>

void merge(int vetor[], int inicio, int meio, int fim) {
  int n1 = meio - inicio + 1; // Tamanho da metade esquerda
  int n2 = fim - meio; // Tamanho da metade direita

  int esquerda[n1], direita[n2]; // Vetores auxiliares

  for (int i = 0; i < n1; i++)
    esquerda[i] = vetor[inicio + i]; // Copia metade esquerda
  for (int j = 0; j < n2; j++)
    direita[j] = vetor[meio + 1 + j]; // Copia metade direita

  int i = 0, j = 0, k = inicio; // Índices

  while (i < n1 && j < n2) { // Junta vetores em ordem
    if (esquerda[i] <= direita[j]) {
      vetor[k++] = esquerda[i++];
    } else {
      vetor[k++] = direita[j++];
    }
  }

  while (i < n1) { // Copia elementos restantes da esquerda
    vetor[k++] = esquerda[i++];
  }
```

```
    while (j < n2) { // Copia elementos restantes da direita
        vetor[k++] = direita[j++];
    }
}

void mergeSort(int vetor[], int inicio, int fim) {
    if (inicio < fim) {
        int meio = inicio + (fim - inicio) / 2; // Calcula meio
        mergeSort(vetor, inicio, meio); // Ordena parte esquerda
        mergeSort(vetor, meio + 1, fim); // Ordena parte direita
        merge(vetor, inicio, meio, fim); // Junta ordenados
    }
}

int main() {
    int vetor[6] = {34, 7, 19, 2, 46, 13}; // Vetor aleatório
    mergeSort(vetor, 0, 5); // Chamada principal

    for (int i = 0; i < 6; i++) {
        printf("%d ", vetor[i]); // Exibe resultado
    }

    return 0;
}
```

Quick Sort – Explicação geral

O Quick Sort escolhe um pivô e divide o vetor de modo que os menores fiquem à esquerda e os maiores à direita. Depois, repete esse processo recursivamente para as partes restantes.

```
#include <stdio.h>

void quickSort(int vetor[], int inicio, int fim) {
    if (inicio < fim) {
        int pivo = vetor[fim]; // Define pivô
        int i = inicio - 1;

        for (int j = inicio; j < fim; j++) {
            if (vetor[j] < pivo) { // Se menor que pivô
                i++;
```

```
            int temp = vetor[i]; // Troca elementos
            vetor[i] = vetor[j];
            vetor[j] = temp;
        }
    }

    int temp = vetor[i + 1]; // Posiciona pivô corretamente
    vetor[i + 1] = vetor[fim];
    vetor[fim] = temp;

    int indicePivo = i + 1;

    quickSort(vetor, inicio, indicePivo - 1); // Ordena esquerda
    quickSort(vetor, indicePivo + 1, fim); // Ordena direita
  }
}

int main() {
  int vetor[6] = {42, 17, 5, 23, 9, 31}; // Vetor de teste
  quickSort(vetor, 0, 5); // Chamada inicial

  for (int i = 0; i < 6; i++) {
    printf("%d ", vetor[i]); // Exibe resultado
  }

  return 0;
}
```