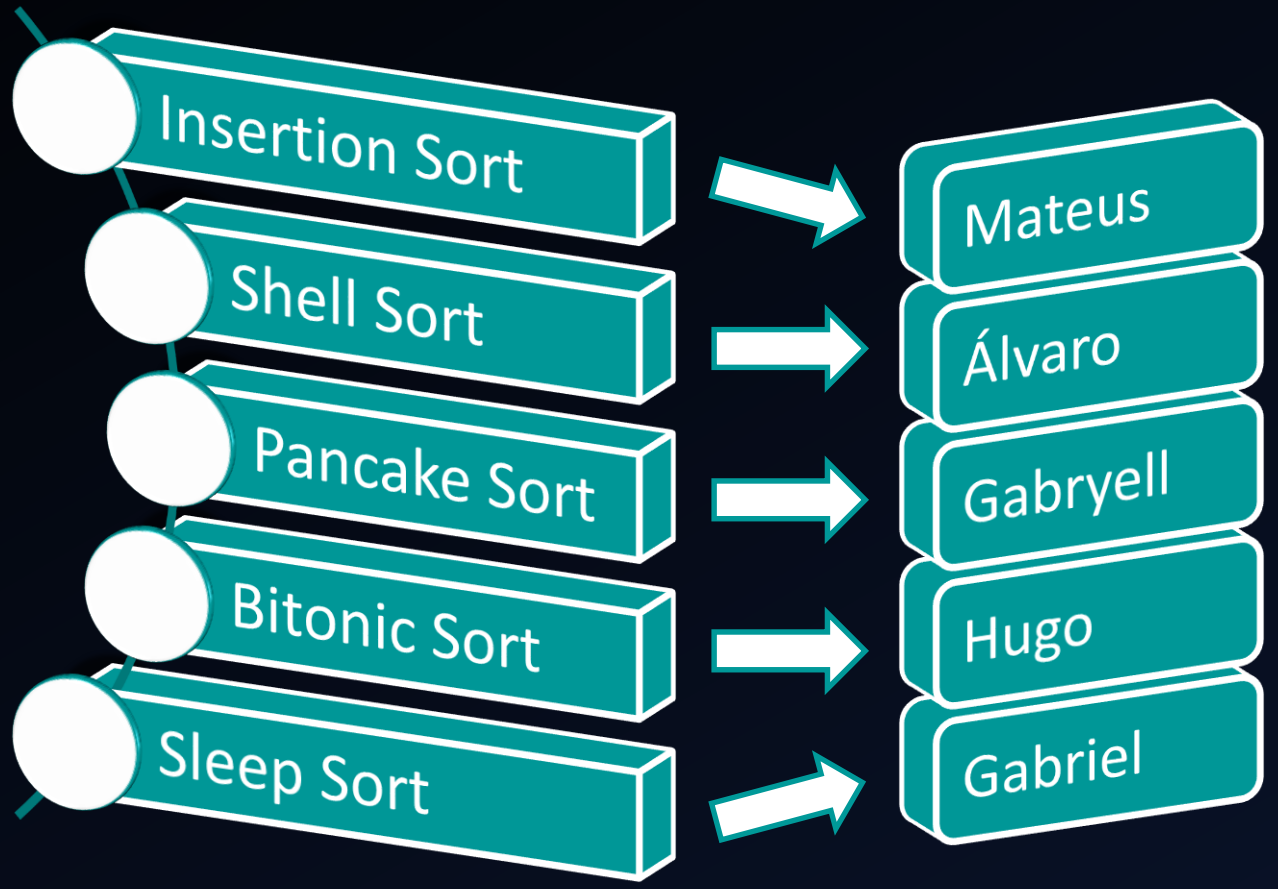
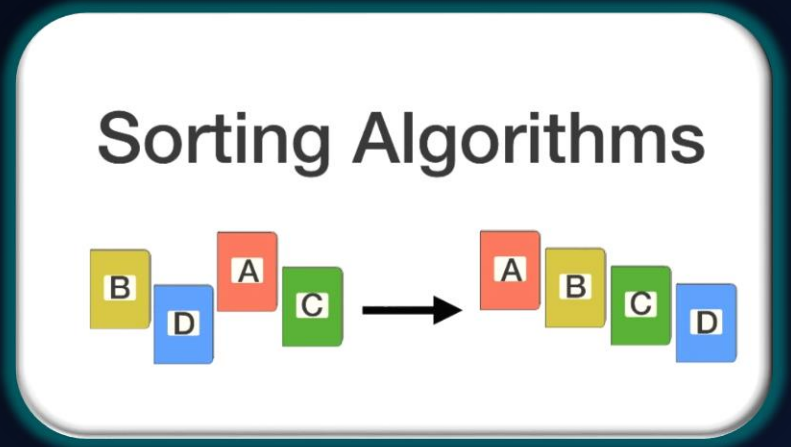
Algoritmos de Ordenação

INSERTION, SHELL, PANCAKE, BITONIC, SLEEP

Av2 Estrutura de Dados 4º Período





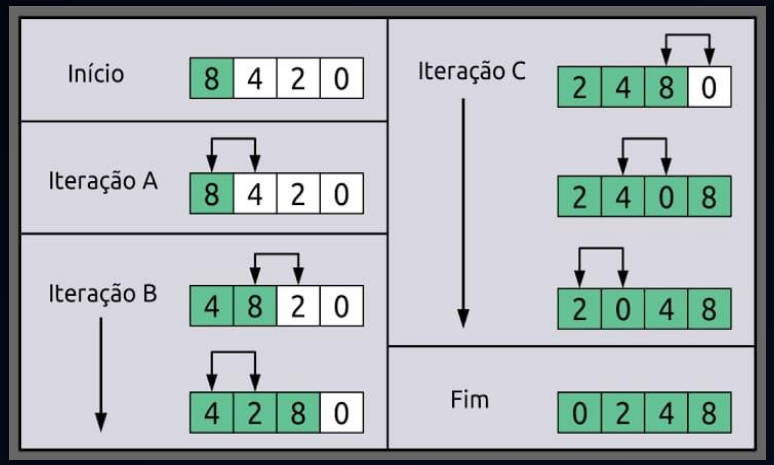
Insertion Sort

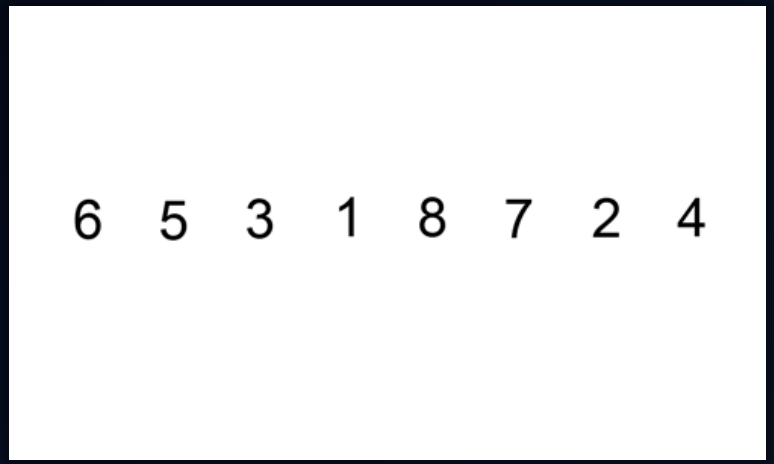
ANDERSON MATEUS

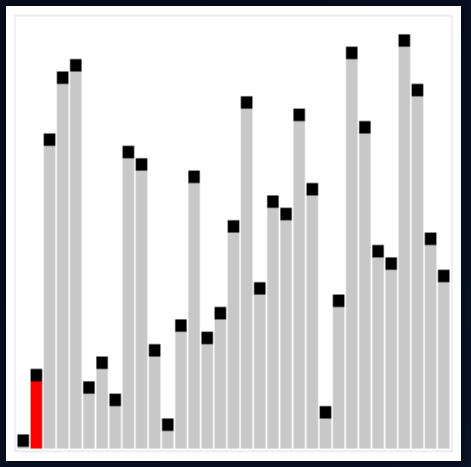
Insertion Sort (Ordenação por Inserção)

- História:
  - Jonh Von Neumann, 1945
  - Um dos primeiros algoritmos

- Funcionamento
  - A partir de inserções
  - “jogo de cartas de baralhos”
  - Posição correta







Insertion Sort (Ordenação por Inserção)

```cpp
// Insertion Sort in C++
#include <iostream>
using namespace std;

// Function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;

    // Compare key with each element on the left of it until an element
smaller than it is found.
    // For descending order, change key<array[j] to key>array[j].
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}
```

```cpp
// Driver code
int main() {
  int data[] = {9, 5, 1, 4, 3};
  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);
  cout << "\nSorted array in ascending order: ";
  printArray(data, size);
  cout << "\n";
}

// Console:
// Sorted array in ascending order: 1 3 4 5 9
```

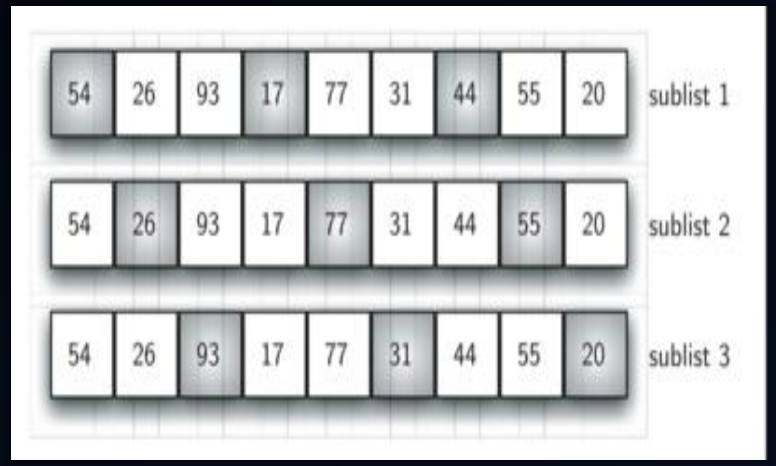
Shell Sort

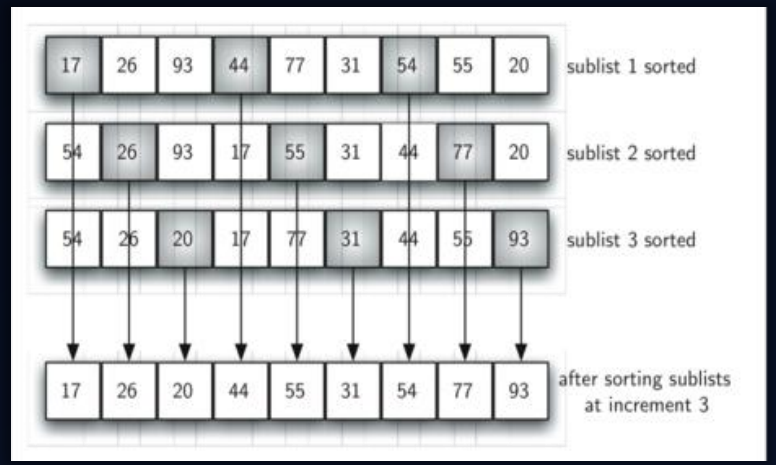
ÁLVARO GABRIEL

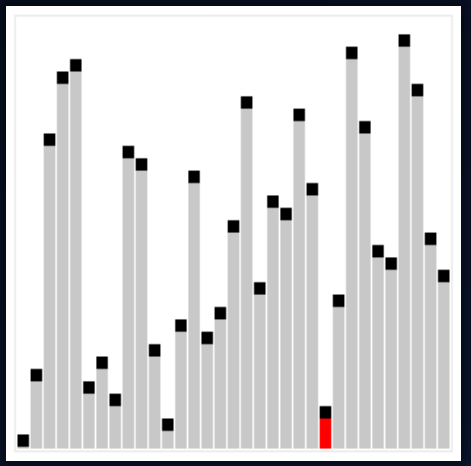
Shell Sort (Ordenação por Incremento)

- História:
  - Donald Shell, 1959
  - Melhorar o Insertion Sort

- Funcionamento
  - Sublistas
  - Gap
  - Usa o Insertion Sort







Shell Sort (Ordenação por Incremento)

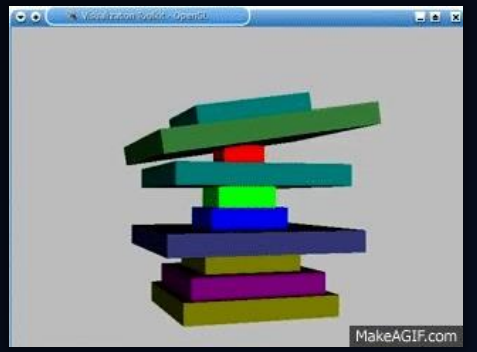
```cpp
// Shell Sort in C++
#include <iostream>
#include <vector>
using namespace std;

void gapInsertionSort(vector<int>& alist, int start, int gap) {
    for (int i = start + gap; i < alist.size(); i += gap) {
        int currentvalue = alist[i];
        int position = i;
        while (position >= gap && alist[position - gap] > currentvalue) {
            alist[position] = alist[position - gap];
            position = position - gap;
        }
        alist[position] = currentvalue;
    }
}

void shellSort(vector<int>& alist) {
    int sublistcount = alist.size() / 2;
    while (sublistcount > 0) {
        for (int startposition = 0; startposition < sublistcount;
++startposition) {
            gapInsertionSort(alist, startposition, sublistcount);
        }
        cout << "After increments of size " << sublistcount << " The list is:
";
        for (int element : alist) {
            cout << element << " ";
        }
        cout << endl;
        sublistcount = sublistcount / 2;
    }
}
```

```cpp
int main() {
    vector<int> alist = {54, 26, 93, 17, 77, 31, 44, 55, 20};
    shellSort(alist);
    cout << "Sorted list: ";
    for (int element : alist) {
        cout << element << " ";
    }
    cout << endl;
    return 0;
}
```

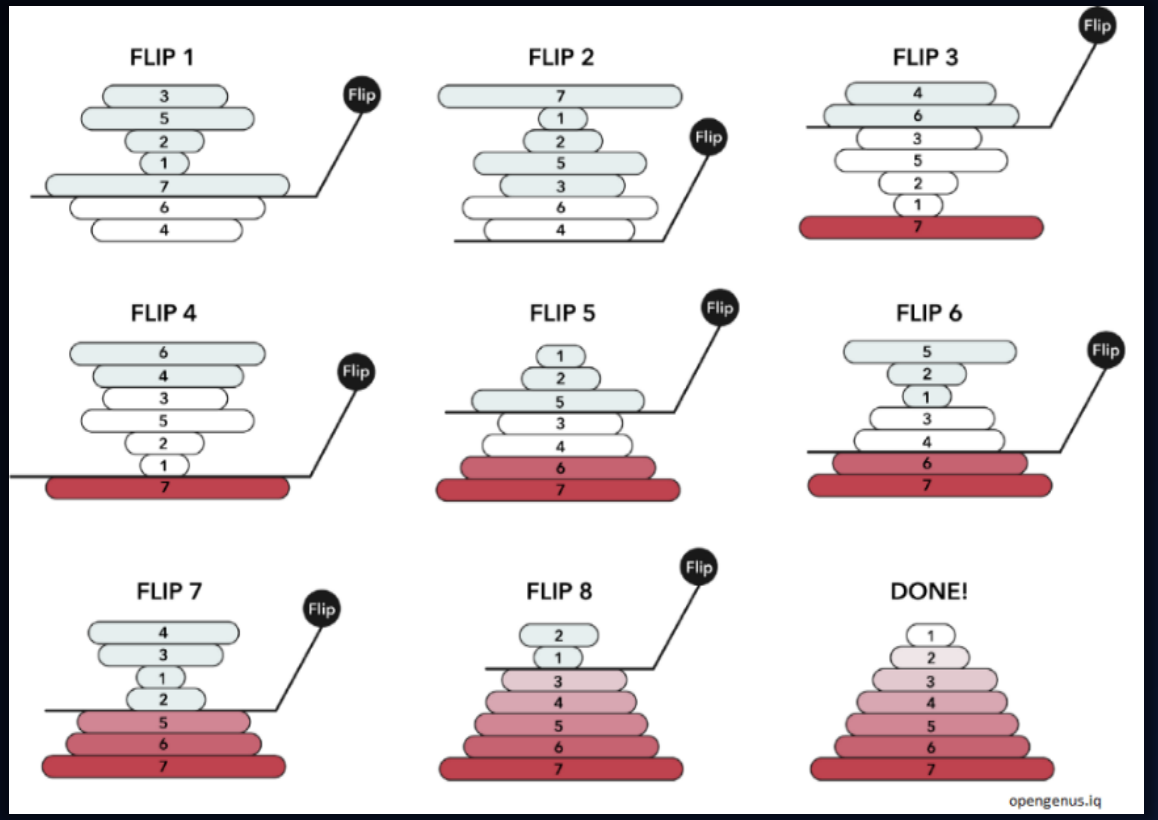
Pancake Sort

GABRYELL GUERRA

Pancake Sort (Ordenação por Inversão)



- História:
  - Jacob E Goodman, 1975
  - Problema Toalhas/ Panquecas



- Funcionamento
  - Inverte porções da lista
  - Maiores em baixo/ no fim
  - Menores em cima/ no começo

Pancake Sort (Ordenação por Inversão)

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

// Inverte arr[0..i]
void flip(int array[], int i) {
    int temp, start = 0;
    while (start < i){  // Troca os elementos nas posições start e i
        temp = array[start];
        array[start] = array[i];
        array[i] = temp;
        start++;
        i--;
    }
}

// Retorna o índice do elemento máximo em array[0..n-1]
int findMax(int array[], int n) {
    int m, i;
    for (m = 0, i = 0; i < n; ++i)
        if (array[i] > array[m])
            m = i;
    return m;
}

// Função principal que ordena o array usando operações de flip
void pancake_Sort(int *array, int n) {
    for (int curr_size = n; curr_size > 1; --curr_size) {
        // Encontra o índice do elemento máximo em arr[0..curr_size-1]
        int m = findMax(array, curr_size);
        // Move o elemento máximo para o final do array, se já não estiver lá
        if (m != curr_size-1) {
            // Move o número máximo para o início primeiro
            flip(array, m);
            // Agora move o número máximo para o final (inverte array atual)
            flip(array, curr_size-1);
        }
    }
}
```

```cpp
// Função de utilidade para imprimir um array de tamanho n
void printArray(int array[], int n) {
    for (int i = 0; i < n; ++i)
        cout << array[i] << " ";
}

// Função principal (Driver program)
int main() {
    int array[] = {20, 5, 2, 13, 15, 6, 14};
    int n = sizeof(array) / sizeof(array[0]);
    // Chama a função para ordenar o array
    pancake_Sort(array, n);
    cout << "Sorted Array:" << endl;
    printArray(array, n);
    return 0;
}
```

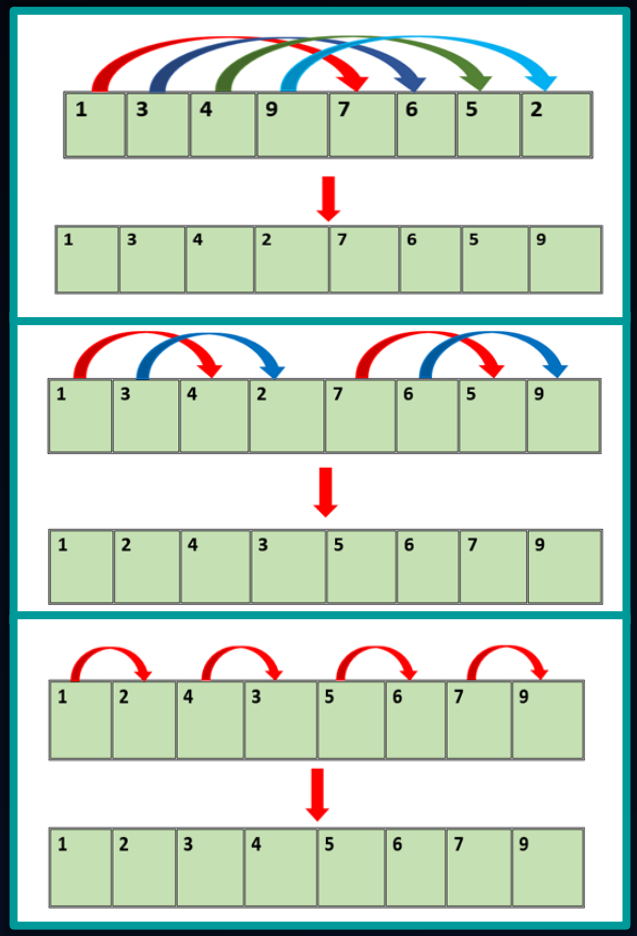
Bitonic Sort

HUGO ALVES

Bitonic Sort (Ordenação por Paralelidade)

- História:
  - Ken Batcher, 1968
  - Eficiente para arquiteturas paralelas e redes de ordenação

- Funcionamento
  - Potências de 2
  - Sequências bitônicas
  - Construção e ordenação



Bitonic Sort (Ordenação por Paralelidade)

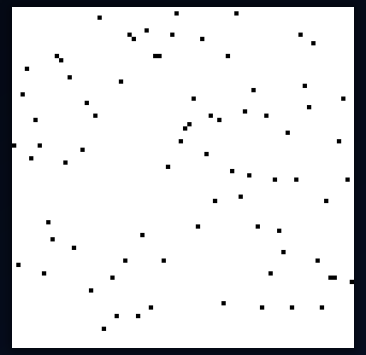
```cpp
// Bitonic Sort C++. Tamanho da entrada deve ser uma potência de 2.
#include<bits/stdc++.h>
using namespace std;

// dir: direção (Ascendente ou Descendente);
// se a direção concordar com a[i] > a[j], a[i] e a[j] serão trocados.
void compAndSwap(int a[], int i, int j, int dir) {
    if (dir==(a[i]>a[j]))
        swap(a[i],a[j]);
}

// Ordena uma sequência bitônica em ascendente (dir=1) ou descendente (dir=0)
// A sequência inicia no índice low e cnt é o número de elementos a ordenar
void bitonicMerge(int a[], int low, int cnt, int dir) {
    if (cnt>1) {
        int k = cnt/2;
        for (int i=low; i<low+k; i++)
            compAndSwap(a, i, i+k, dir);
        bitonicMerge(a, low, k, dir);
        bitonicMerge(a, low+k, k, dir);
    }
}

// Função geral: produz uma sequência bitônica por recursividade,
// ordenando suas duas metades em ordens diferentes,
// então chama bitonicMerge para mesclá-las na mesma ordem.
void bitonicSort(int a[],int low, int cnt, int dir) {
    if (cnt>1) {
        int k = cnt/2;
        bitonicSort(a, low, k, 1); // ordena na direção ascendente (1)
        bitonicSort(a, low+k, k, 0); // ordena na direção descendente (0)
        bitonicMerge(a,low, cnt, dir); // Mescla ambas em ascendência (dir=1)
    }
}
```

```cpp
// Chama a bitonicSort para ordenar a lista de tamanho N em ordem ascendente
void sort(int a[], int N, int up) {
    bitonicSort(a,0, N, up);
}

// Driver code
int main() {
    int a[]= {3, 7, 4, 8, 6, 2, 1, 5};
    int N = sizeof(a)/sizeof(a[0]);
    int up = 1; // significa ordenar em ordem ascendente
    sort(a, N, up);
    printf("Sorted array: \n");
    for (int i=0; i<N; i++)
        printf("%d ", a[i]);
    return 0;
}
```

Sleep Sort

GABRIEL VIEIRA

Sleep Sort (Ordenação por Tempo)



- História:
  - Usuários de fóruns, meados 2000
  - Nova abordagem interessante e simples
- Funcionamento
  - Threads para os elementos
  - Impressão após tempo
  - Não é eficiente

Sleep Sort (Ordenação por Tempo)

```cpp
// Shell Sort in C++
#include <iostream>
#include <vector>
using namespace std;

void gapInsertionSort(vector<int>& alist, int start, int gap) {
    for (int i = start + gap; i < alist.size(); i += gap) {
        int currentvalue = alist[i];
        int position = i;
        while (position >= gap && alist[position - gap] > currentvalue) {
            alist[position] = alist[position - gap];
            position = position - gap;
        }
        alist[position] = currentvalue;
    }
}

void shellSort(vector<int>& alist) {
    int sublistcount = alist.size() / 2;
    while (sublistcount > 0) {
        for (int startposition = 0; startposition < sublistcount;
++startposition) {
            gapInsertionSort(alist, startposition, sublistcount);
        }
        cout << "After increments of size " << sublistcount << " The list is:
";
        for (int element : alist) {
            cout << element << " ";
        }
        cout << endl;
        sublistcount = sublistcount / 2;
    }
}
```

```cpp
int main() {
    vector<int> alist = {54, 26, 93, 17, 77, 31, 44, 55, 20};
    shellSort(alist);
    cout << "Sorted list: ";
    for (int element : alist) {
        cout << element << " ";
    }
    cout << endl;
    return 0;
}
```

rpo ançoeãt! oabigOrd asu