UNIVERSIDADE ESTÁCIO DE SÁ

BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

LUANN RICARDO CAETANO DE SOUZA

**Trabalho de Estrutura de Dados em C**

RIO DE JANEIRO – RECREIO

2022

LUANN RICARDO CAETANO DE SOUZA

**Trabalho de Estrutura de Dados em C**

Trabalho de Estrutura de Dados em C, para a Universidade Estácio de Sá, como requisito para o recebimento de nota em Ciências da Computação.

Orientador: Raphael Jesus

Rio de Janeiro – Recreio

2022

Bubble Sort

Bubble Sort é um algoritmo de ordenação que pode ser aplicado em Arrays e Listas dinâmicas. Se o objetivo é ordenar os valores em forma decrescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for maior que a posição posterior, é realizada a troca dos valores nessa posição. Caso contrário, não é realizada a troca, apenas passa-se para o próximo par de comparações.

Se o objetivo é ordenar os valores em forma crescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for menor que a posição posterior, é realizada a troca. Caso contrário, a troca não é feita e passa-se para o próximo par de comparação.

Insertion Sort

O Insertion Sort tem como rotina base a inserção ordenada. A ideia é executar várias vezes essa rotina para ordenar um array. Para ser exato, se executarmos N−1 vezes a rotina de inserção ordenada em um array o resultado é a ordenação completa do mesmo

Quick Sort

O algoritmo Quicksort utiliza o paradigma de programação Dividir para Conquistar, o algoritmo baseia a ordenação em sucessivas execuções de particionamento, uma rotina que escolhe um pivot e o posiciona no array de uma maneira em que os elementos menores ou iguais ao pivot estão à sua esquerda e os maiores estão à sua direita.

Selection Sort

Esse algoritmo seleciona o menor elemento da sequência e colocar esse elemento na primeira posição do array. A ideia é executar várias vezes esses dois passos para ordenar um array. Para ser exato, se executarmos N vezes esses dois passos em um array, controlando os índices em que os passos são executados, o resultado é a ordenação completa do mesmo.

Comparação

Foi realizado um teste utilizando os quatro métodos citados acima e testadas três ordens de listas com três tamanhos diferentes cada:

**Bubble sort** - Para listas já ordenadas em ordem crescente é o único algoritmo que não realiza movimentações, mas em compensação é o que tem o maior tempo e o maior número de comparações. Não só em listas já ordenadas, mas em todos os casos o bubble sort se mostrou um algoritmo ineficiente.

**Insertion Sort** - O Insertion sort se mostrou mais eficiente que todos os outros algoritmos em relação ao tempo e comparações.

**Quick Sort** - O quick sort certamente é o algoritmo mais eficiente em listas totalmente desordenadas, ele se torna muito eficiente em relação aos outros no quesito de tempo.

**Selection sort** - O selection sort foi o segundo pior algoritmo, mas se mostrou mais eficiente do que o Insertion sort em relação ao tempo e a quantidade de movimentações.

Dessa forma, concluo de forma pessoal que o método Insertion é o mais eficiente entre eles, levando em consideração comparações e movimentações.

Fonte dos testes: <https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>

Exemplos

Bubble Sort em C:

#include <stdio.h>

void swap(int\* xp, int\* yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void bubbleSort(int arr[], int n)

{

    int i, j;

    for (i = 0; i < n - 1; i++)

        for (j = 0; j < n - i - 1; j++)

            if (arr[j] > arr[j + 1])

                swap(&arr[j], &arr[j + 1]);

}

void printArray(int arr[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

int main()

{

    int arr[] = { 64, 34, 25, 12, 22, 11, 90 };

    int n = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, n);

    printf("Ordenar array: \n");

    printArray(arr, n);

    return 0;

}

Bubble Sort em C++:

#include <bits/stdc++.h>

using namespace std;

void bubbleSort(int arr[], int n)

{

    int i, j;

    for (i = 0; i < n - 1; i++)

        for (j = 0; j < n - i - 1; j++)

            if (arr[j] > arr[j + 1])

                swap(arr[j], arr[j + 1]);

}

void printArray(int arr[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    int arr[] = { 5, 1, 4, 2, 8};

    int N = sizeof(arr) / sizeof(arr[0]);

    bubbleSort(arr, N);

    cout << "Ordenar array: \n";

    printArray(arr, N);

    return 0;

}

Insertion Sort em C:

#include <math.h>

#include <stdio.h>

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++) {

        key = arr[i];

        j = i - 1;

        while (j >= 0 && arr[j] > key) {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void printArray(int arr[], int n)

{

    int i;

    for (i = 0; i < n; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

int main()

{

    int arr[] = { 12, 11, 13, 5, 6 };

    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);

    printArray(arr, n);

    return 0;

}

Insertion Sort em C++:

#include <bits/stdc++.h>

using namespace std;

void insertionSort(int arr[], int n)

{

    int i, key, j;

    for (i = 1; i < n; i++)

    {

        key = arr[i];

        j = i - 1;

        while (j >= 0 && arr[j] > key)

        {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

}

void printArray(int arr[], int n)

{

    int i;

    for (i = 0; i < n; i++)

        cout << arr[i] << " ";

    cout << endl;

}

 int main()

{

    int arr[] = { 12, 11, 13, 5, 6 };

    int N = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, N);

    printArray(arr, N);

    return 0;

}

Quick Sort em C:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void

troca (int vet[], int i, int j)

{

int aux = vet[i];

vet[i] = vet[j];

vet[j] = aux;

}

int

particiona (int vet[], int inicio, int fim)

{

int pivo, pivo\_indice, i;

pivo = vet[fim];

pivo\_indice = inicio;

for (i = inicio; i < fim; i++)

{

if (vet[i] <= pivo)

{

troca (vet, i, pivo\_indice);

pivo\_indice++;

}

}

troca (vet, pivo\_indice, fim);

return pivo\_indice;

}

int

particiona\_random (int vet[], int inicio, int fim)

{

int pivo\_indice = (rand () % (fim - inicio + 1)) + inicio;

troca (vet, pivo\_indice, fim);

return particiona (vet, inicio, fim);

}

void

quick\_sort (int vet[], int inicio, int fim)

{

if (inicio < fim)

{

int pivo\_indice = particiona\_random (vet, inicio, fim);

quick\_sort (vet, inicio, pivo\_indice - 1);

quick\_sort (vet, pivo\_indice + 1, fim);

}

}

int

main ()

{

int vet[] = { 25, 40, 55, 20, 44, 35, 38, 99, 10, 65, 50 };

int tam\_vet = sizeof (vet) / sizeof (int);

int i;

srand (time (NULL));

quick\_sort (vet, 0, tam\_vet - 1);

for (i = 0; i < tam\_vet; i++)

printf ("%d ", vet[i]);

return 0;

}

Quick Sort em C++:

#include <bits/stdc++.h>

using namespace std;

void swap(int\* a, int\* b)

{

    int t = \*a;

    \*a = \*b;

    \*b = t;

}

int partition(int arr[], int low, int high)

{

    int pivot = arr[high];

    int i

        = (low

           - 1);

    for (int j = low; j <= high - 1; j++) {

        if (arr[j] < pivot) {

            i++;

            swap(&arr[i], &arr[j]);

        }  }

    swap(&arr[i + 1], &arr[high]);

    return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high) {

        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i = 0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    int arr[] = { 10, 7, 8, 9, 1, 5 };

    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: \n";

    printArray(arr, n);

    return 0;

}

Selection Sort em C:

#include <stdio.h>

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void selectionSort(int arr[], int n)

{

    int i, j, min\_idx;

    for (i = 0; i < n-1; i++)

    {

        min\_idx = i;

        for (j = i+1; j < n; j++)

          if (arr[j] < arr[min\_idx])

            min\_idx = j;

                   if(min\_idx != i)

            swap(&arr[min\_idx], &arr[i]);

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", arr[i]);

    printf("\n");

}

int main()

{

    int arr[] = {64, 25, 12, 22, 11};

    int n = sizeof(arr)/sizeof(arr[0]);

    selectionSort(arr, n);

    printf("Sorted array: \n");

    printArray(arr, n);

    return 0;

}

Selection Sort em C++:

#include <bits/stdc++.h>

using namespace std;

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void selectionSort(int arr[], int n)

{

    int i, j, min\_idx;

    for (i = 0; i < n-1; i++)

    {

        min\_idx = i;

        for (j = i+1; j < n; j++)

        if (arr[j] < arr[min\_idx])

            min\_idx = j;

        if(min\_idx!=i)

            swap(&arr[min\_idx], &arr[i]);

    }

}

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        cout << arr[i] << " ";

    cout << endl;

}

int main()

{

    int arr[] = {64, 25, 12, 22, 11};

    int n = sizeof(arr)/sizeof(arr[0]);

    selectionSort(arr, n);

    cout << "Sorted array: \n";

    printArray(arr, n);

    return 0;

}