**Selection Sort**

Selection sort é um algoritmo de ordenação que opera em uma complexidade C(n) = O(n²) , pois o número de operações sempre é o mesmo no melhor caso (em que os valores já estão ordenados) e no pior caso (em que os valores estão na ordem contrária da requerida) e é feito sempre n² vezes em relação ao tamanho do array.

O algoritmo funciona sempre pegando o menor valor da lista e colocando ele no começo da lista e depois o segundo menor e colocando ele na segunda posição e assim até o último elemento(isso contando com que o programa deseja ordenar em ordem crescente, já que se fosse em ordem decrescente seria o maior valor trabalhado no lugar dor menor).

Possui como uma das principais vantagens o fato de ser implementado de uma forma mais simples em relação aos demais algoritmos de comparação, não necessita de um vetor auxiliar o que torna melhor para aplicações que não podem usar muita memória, e é também um dos mais velozes na ordenação de vetores pequenos. Mas possui desvantagens também, que são o fato de que sempre faz n² comparações não importa se o vetor está ordenado ou não, é lento para vetores que possuem muitos elementos e dependendo do algoritmo não é estável.

Exemplo de algoritmo Selection Sort:

void selection\_sort (int vetor[],int max) {

  int i, j, min, aux;

  for (i = 0; i < (max - 1); i++) {

    /\* O minimo é o primeiro número não ordenado ainda \*/

    min = i;

    for (j = i+1; j < max; j++) {

      /\* Caso tenha algum numero menor ele faz a troca do minimo\*/

      if (vetor[j] < vetor[min]) {

   min = j;

      }

    }

    /\* Se o mínimo for diferente do primeiro numero não ordenado ele faz a troca para ordena-los\*/

    if (i != min) {

      aux = vetor[i];

      vetor[i] = vetor[min];

      vetor[min] = aux;

    }

  }

  /\* Imprime o vetor ordenado \*/

  for (i = 0; i < max; i++) {

    printf ("%d ",vetor[i]);

  }

  printf ("\n");

}

**Insertion Sort**

É um algoritmo de ordenação que assim como o selection sort é menos eficiente que algoritmos mais avançados, como por exemplo quicksort e o merge sort, para vetores de grande escala. E possui uma complexidade de O(n²) (assim como o Selection sort) mas apenas para os “casos médios” e “piores casos” já que se for o pior caso (ou seja ordem inversa da desejada ) e o “caso médio” (em que a lista está bagunçada) fazem com que cada iteração do looping passe por toda a parte do vetor e desloque ele, já no melhor caso em que o vetor já está ordenado ele fará uma execução linear, ou seja O(n).

O algoritmo de insertion sort começa analisando já o segundo valor do array com o primeiro e então, depois o terceiro com os dois anteriores, depois o quarto com os três anteriores e assim sucessivamente até o ultimo elemento do vetor. O código faz isso para analisar onde o valor será inserido, se será na ponta, no meio, ou se o valor fica na mesma posição.

Este algoritmo, apesar de possuir um alto custo computacional na movimentação dos elementos do vetor e não ser muito efetivo em vetores de grande tamanhos é um método bastante usado para coisas em que já está ‘quase’ ordenado e se for para adicionar poucos elementos em um arquivo já ordenado é uma boa opção também por ter um custo linear nesse caso.

void insertionSort(int arr[], int n){

int i, j, valor;

for(i = 1; i < n; i++){

valor = arr[i];

j = i – 1;

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

arr[j+1] = arr[j];

j = j – 1;

}

arr[j+1] = valor;

}

}

Bubble Sort

É um algoritmo de ordenação em que consiste percorrer o vetor várias vezes e então fazer em cada uma delas com que os valores maiores, se no caso o foco for organizar em ordem crescente o array, o valor ‘flutue’, literalmente como se fosse uma bolha, até a extremidade possível do vetor. E assim como os algoritmos supracitados também possui complexidade O(n²), mas no melhor caso isso se torna linear, ou seja, O(n).

E assim como o insertion sort, e selection sort, possui a desvantagem de que é mais funcional em vetores menores, pois, a sua ordem ser quadrática(O[n²]) para a execução dessa ordenação faz com que não seja tão efetivo quanto outros já existentes para grandes vetores.

Mas possui uma boa vantagem, de ser um código de fácil compreensão, adaptação e fácil de se conseguir usar.

Considerando que a ordenação é para ordem crescente, o código consiste em percorrer o vetor, e sempre verificar se o valor que “está na vez” é maior que o próximo valor e se isso for verdade uma variável auxiliar recebe o valor que está sendo ‘analisado’ e na posição do valor que está sendo ‘analisado’ o próximo fica no lugar, e no lugar do próximo a partir do valor salvo na variável auxiliar ele é colocado nessa posição. E isso se repete N vezes (sendo N o tamanho do vetor)