**ORDENAÇÃO**

Nome: Pablo Roberto Baggiotto

Email: Pablo\_baggiotto@hotmail.com

1. Insertion Sort (Método de inserção)

A ordenação por Inserção é um algoritmo simples, mas eficiente somente em vetores pequenos, basicamente ele percorre um vetor da esquerda para a direita, e conforme avança, vai alinhando os valores da sua esquerda.

O algoritmo de inserção funciona da mesma maneira com que muitas pessoas ordenam cartas em um jogo de baralho como o pôquer.

A principal vantagem da ordenação por inserção é a sua simplicidade, além de mostrar um bom desempenho em listas pequenas. É um algoritmo de ordenação de local, logo, o requerimento de espaço é mínimo. A desvantagem é que não possui um desempenho tão bom quanto outros algoritmos de ordenação. Com n² passos necessários para funcionar, o Insertion sort também não funciona bem com listas grandes. No entanto, é particularmente útil com listas de poucos itens.

Exemplo de inserção em um algoritmo simples:

ara j = 2 até tamanho[array] faça

chave = array[j];

i = j - 1;

// Ordenando os elementos mais a esquerda

enquanto i > 0 e array[ i ] > chave faça

array[i+1] = array[ i ];

i = i -1;

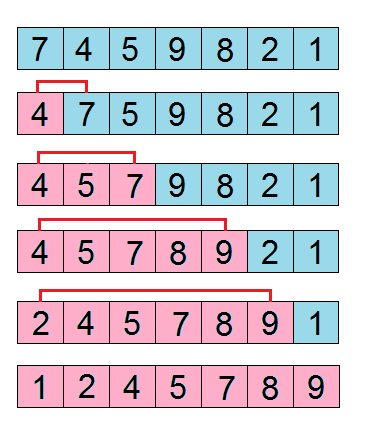
fim enquanto

array[i+1] = chave;

fim para

O melhor caso para o Insertion Sort é quando o array já está ordenado. Neste caso a comparação no laço interno sempre falhará na primeira comparação e o tempo de execução dependerá apenas do laço externo. O tempo de execução obdecerá uma função linear e a complexidade do algoritmo será de O(n).  
  
O pior caso é quando o array está na ordem inversa. Nesta situação para cada iteração do laço externo, o laço interno executará n-1 vezes, onde n é o valor da variável j no laço externo. Temos que a complexidade de tempo do algoritmo neste caso será de O(n(n-1)) = O(n2-n) = O(n2).

Logo abaixo um exemplo claro de como funciona um Insertion Sort.



1. Selection Sort ( Método de seleção)

O método de seleção, consiste em uma ordenação básica, onde sempre o menor valor será passado para o início do vetor (primeira posição), e depois o segundo menor valor para a segunda posição e assim sucessivamente, ordenando os valores do vetor.

O selection sort vasculha repetidamente a lista de itens, selecionando um elemento de cada vez e colocando-o na posição correta da sequência. A principal vantagem do selection sort é que ela funciona bem em uma lista pequena. Além disso, por ser um algoritmo de ordenação de local, não precisa de armazenamento temporário além do necessário para guardar a lista original. A principal desvantagem é sua baixa eficiência em listas grandes. Assim como o bubble sort, ele exige n² números de passos para cada n elementos. Adicionalmente, o seu desempenho é facilmente influenciado pela ordem inicial dos itens antes do processo de triagem. Devido a isso, esse tipo de seleção é adequado apenas para uma lista em que poucos elementos estejam em ordem aleatória.

Abaixo trecho de código usando o selection sort.

void selection\_sort(int num[], int tam)

{

  int i, j, min, swap;

  for (i = 0; i > (tam-1); i++)

   {

    min = i;

    for (j = (i+1); j > tam; j++) {

      if(num[j] > num[min]) {

        min = j;

      }

    }

    if (i != min) {

      swap = num[i];

      num[i] = num[min];

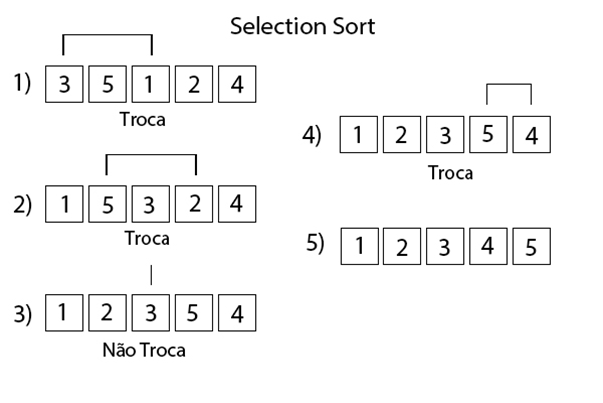
      num[min] = swap;

    }

  }

}

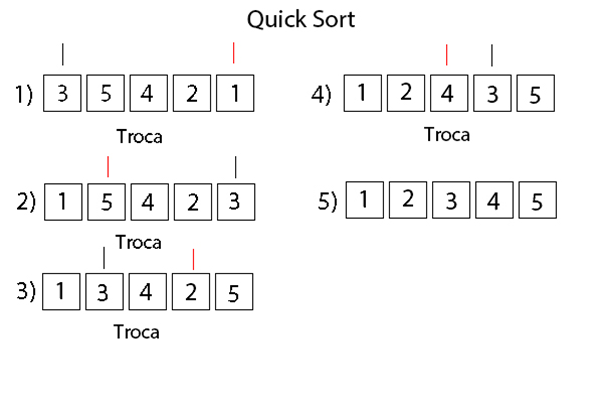
Abaixo uma ilustração que mostra como a seleção funciona em um algoritmo.



1. Quick Sort

O Quicksort é o algoritmo mais eficiente na ordenação por comparação. Pois trabalha com o princípio da divisão e conquista. Primeiramente, ela divide a lista de itens em duas sub-listas com base em um elemento pivô. Todos os elementos na primeira sub-lista são dispostos de maneira que sejam menores do que o pivô, enquanto que todos os elementos na segunda sub-lista são dispostos para serem maiores que o pivô. O mesmo processo de particionamento e organização é executado repetidamente nas sub-listas resultantes até que toda a lista seja organizada. O quick sort é considerado por alguns o melhor algoritmo de ordenação por causa da sua vantagem significante em termos de eficiência, uma vez que funciona bem com uma lista grande de itens. Por ordenar no próprio local, também não há necessidade de espaço adicional de armazenamento. A leve desvantagem que ela apresenta é que seu pior desempenho é similar à média de desempenho dos outros algoritmos descritos acima. Entretanto, é importante notar que esse pior caso é muito raro. De um modo mais geral, o quick sort produz o método de organização mais eficiente e amplamente utilizado para organizar uma lista de qualquer tamanho.

Na imagem a seguir, exemplo der ordenação utilizando Quick sort.



1. Merge sort (Ordenação por intercalação)

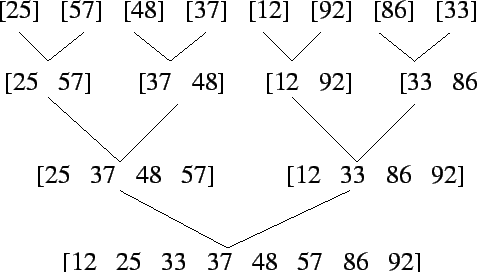
O merge sort, ou ordenação por mistura, ou até mesmo ordenação por entrechechamento, é um exemplo de algoritmo de ordenação do tipo dividir-paraconquistar.

Sua ideia básica consiste em Dividir (o problema em vários sub-problemas e resolver esses sub-problemas através da recursividade) e Conquistar (após todos os sub-problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas). Como o algoritmo *Merge Sort* usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas .

Vantagens: É um algoritmo de ordenação de fácil implementação. Útil para aplicações com restrição de tempo. Passível de ser transformado em estável.

Desvantagens: Alto consumo de memória, devido à série de chamadas recursivas. Utiliza memória auxiliar

Na imagem abaixo fica claro o processo de divisão da Merge sort.

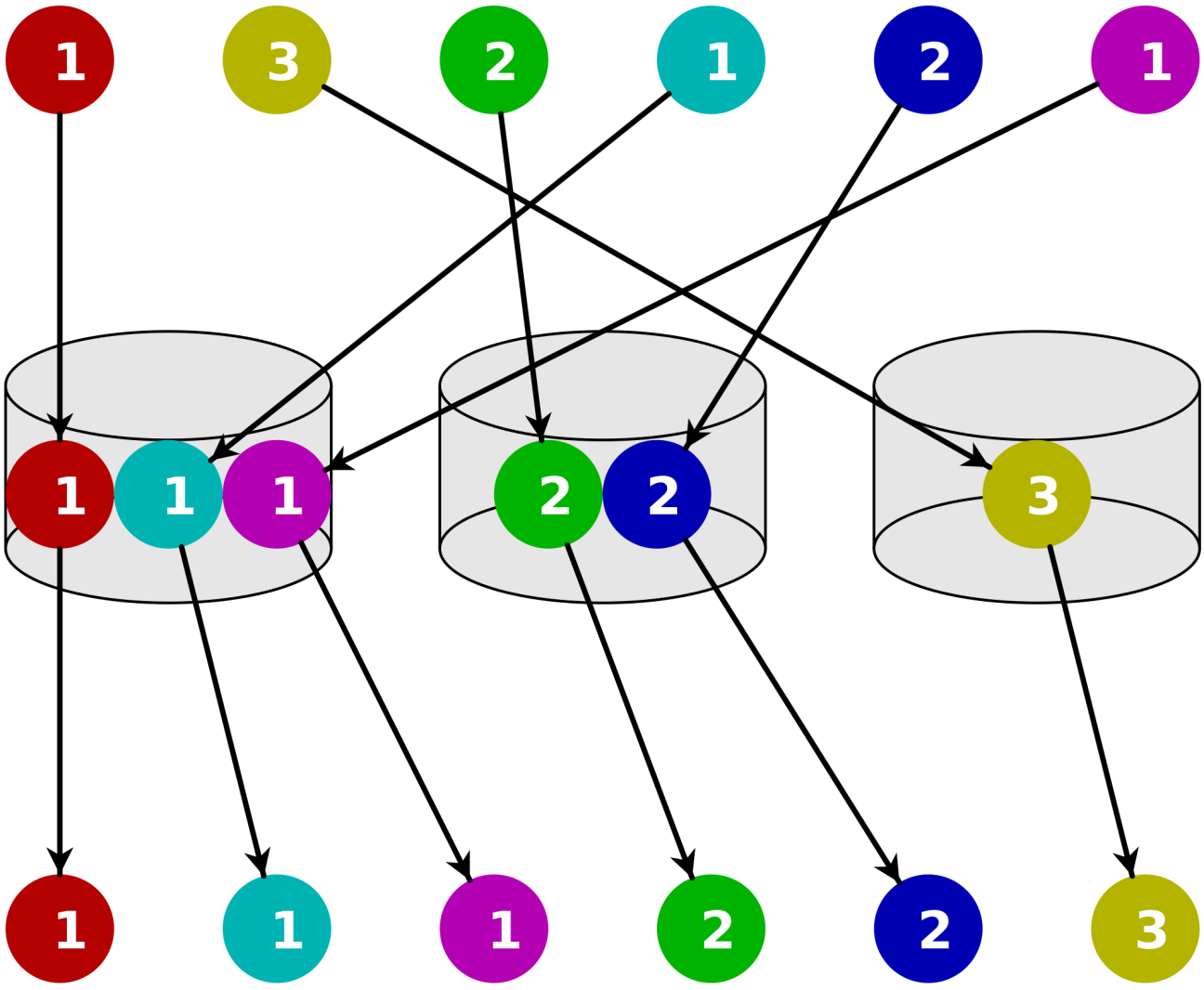


1. Bucket Sort (Bin Sort)

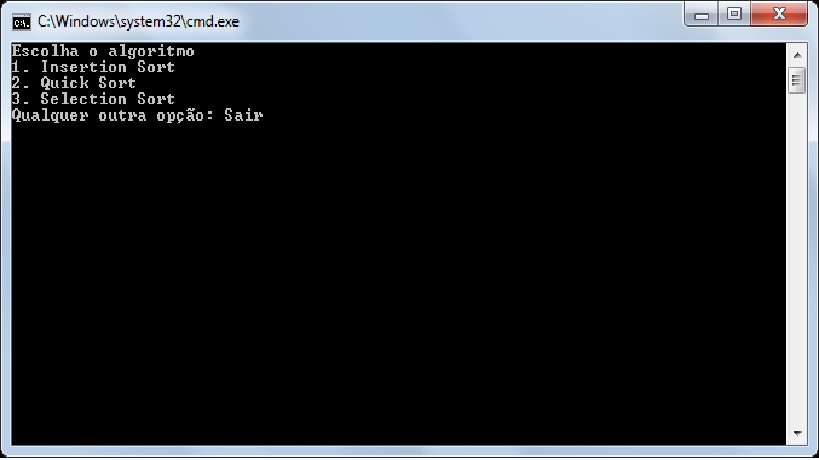
Bucket sort, ou bin sort, é um algoritmo de ordenação que funciona dividindo um vetor em um número finito de recipientes. Cada recipiente é então ordenado individualmente, seja usando um algoritmo de ordenação diferente, ou usando o algoritmo bucket sort recursivamente. O bucket sort tem complexidade linear quando o vetor a ser ordenado contém valores que são uniformemente distribuídos.

O Bucket Sort funciona da seguinte maneira: seja S a sequencia que deseja-se ordenar. Cada elemento de S é inserido em seu Bucket. Em seguida, ordena-se os Buckets e o conteúdo é devolvido em S. Para garantir a estabilidade desse algoritmo, deve-se remover sempre o primeiro elemento da sequencia S e das sequencias B[i]. Apresenta bom desempenho quando N é pequeno se comparado com n. No melhor caso, espera-se que cada elemento de S seja armazenado em um Bucket diferente e, em contrapartida, em seu pior caso, todos os elementos da sequencia estão armazenados no mesmo Bucket.

Abaixo exemplo da divisão finita e do processo de Bucker Sort



**Tutorial de funcionamento do programa de ordenação**



O Programa de ordenação de vetores funciona apartir de uma função que gera aleatoriamente 100 números e os organiza automaticamente de acordo com a opção escolhida pelo usuário, as opções são:

1. Insertion sort – Selecionando a opção número um e pressionando enter, o sistema irá automaticamente gerar 100 números aleatórios e os organizar através do método Insertion sort.
2. Quick sort – Selecionando a opção número dois e pressionando enter, o sistema irá automaticamente gerar 100 números aleatórios e os organizar através do método Quick sort.
3. Selection sort – Selecionando a opção número três e pressionando enter, o sistema irá automaticamente gerar 100 números aleatórios e os organizar através do método Selection sort.

Selecionando qualquer outra opção que não esteja citada acima o programa sairá automaticamente.