Algoritmos II

MÉTODOS DE PESQUISA E ORDENAÇÃO

Métodos de pesquisa e ordenação

- O objetivo da pesquisa é encontrar um ou mais registros com chaves iguais a chave pesquisada
- Existe uma grande variedade de métodos de pesquisa, a escolha do método mais adequado depende de alguns fatores:
  - Quantidade de dados envolvidos
  - Frequência com que operações de inserção e retirada são efetuadas

Métodos de pesquisa e ordenação

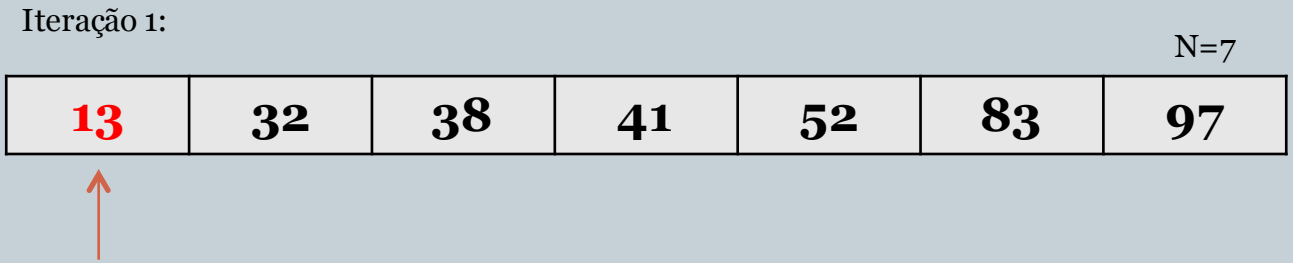
- Algumas das operações mais comuns incluem:
  - Inserção de um novo registro;
  - Pesquisa de um ou mais registros com uma determinada chave para torná-los disponíveis;
  - Remoção de um registro específico;
- Alguns dos principais métodos de pesquisa:
  - Pesquisa sequencial
  - Pesquisa binária

Pesquisa Sequencial

- Método de pesquisa mais simples
- A partir do primeiro registro, pesquisa sequencialmente até encontrar a chave procurada

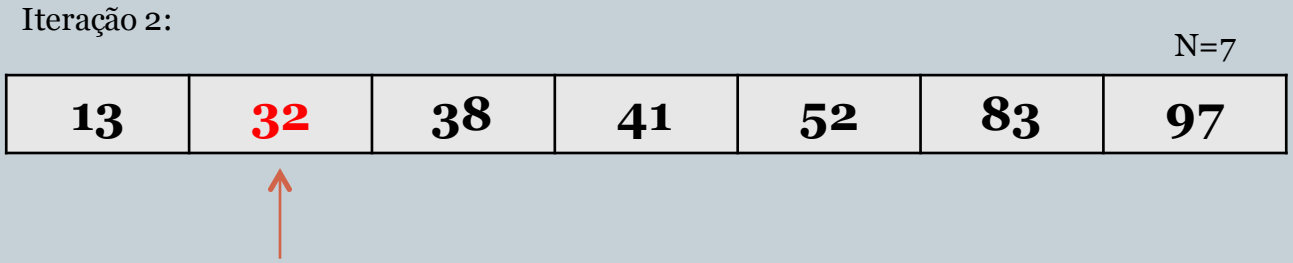
Pesquisa Sequencial

- Exemplo:
  - Procura pela chave 41



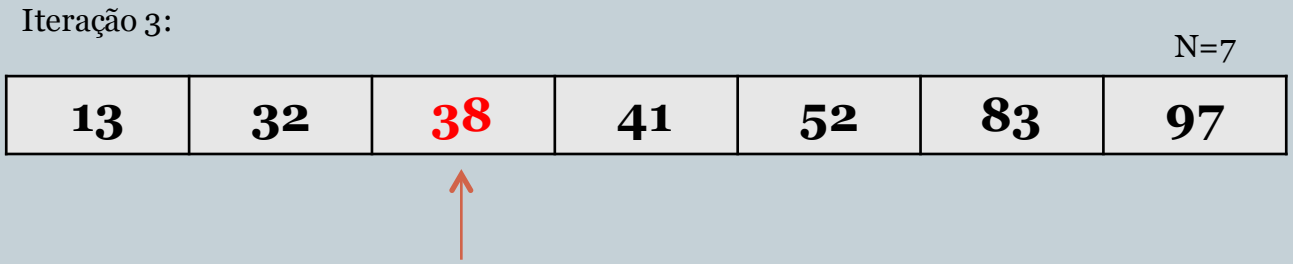
Pesquisa Sequencial

- Exemplo:
  - Procura pela chave 41



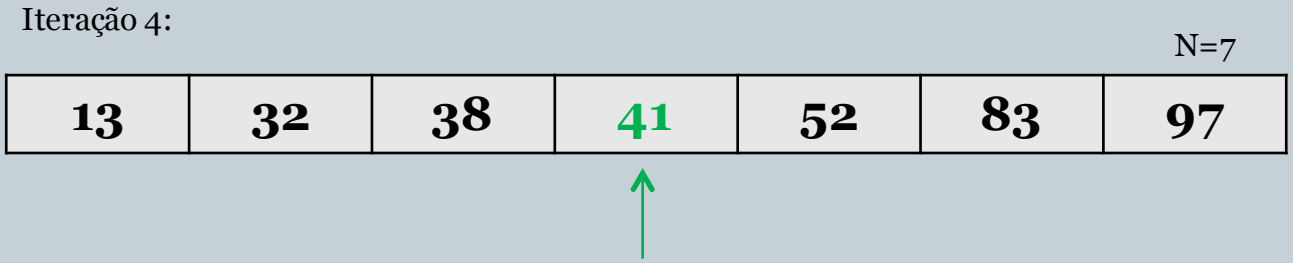
Pesquisa Sequencial

- Exemplo:
  - Procura pela chave 41



Pesquisa Sequencial

- Exemplo:
  - Procura pela chave 41



Pesquisa Sequencial

- Análise:
  - Pesquisa com sucesso
    - Melhor caso: C(n) = 1
    - Pior caso: C(n) = n
    - Caso médio: C(n) = (n+1) / 2
  - Pesquisa sem sucesso
    - C(n) = n + 1

Pesquisa Binária

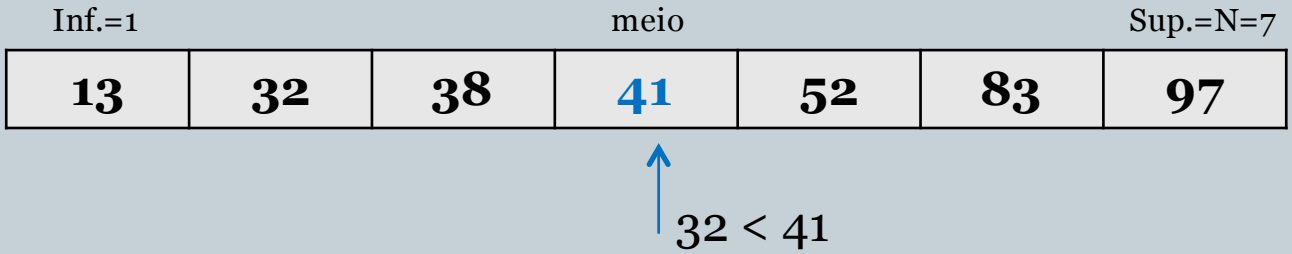
- A pesquisa pode ser muito eficiente se os registros forem mantidos em ordem.
- Reduz o tempo de busca dividindo o conjunto de dados (tabela) em duas partes, na sequência verifica em qual das partes o registro com a chave está localizado e concentra a busca naquela parte
- Para saber se uma determinada chave está presente na tabela compara-se a chave com o registro que está posicionado no meio da tabela

Pesquisa Binária

- Se a chave é menor, então o registro procurado está na primeira metade da tabela, se é maior, o registro está na segunda metade da tabela
- O processo é repetido até que a chave seja encontrada ou fique apenas um registro com uma chave diferente da procurada (pesquisa sem sucesso)

Pesquisa Binária

- Exemplo:
  - Procura pela chave 32

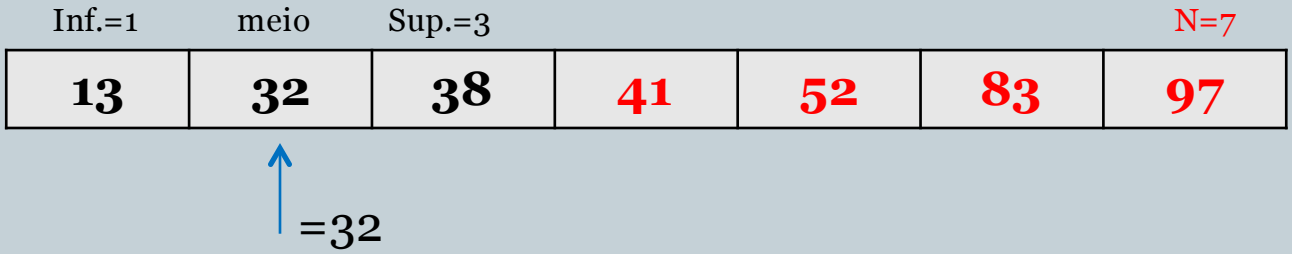


Pesquisa Binária

- Exemplo:
  - Procura pela chave 32

Pesquisa Binária

- Exemplo:
  - Procura pela chave 32



Pesquisa Binária

- Análise:
  - Implementação simples
  - Eficiente na busca
    - A cada iteração do algoritmo, o tamanho da tabela é dividido ao meio
    - Logo, o número de vezes que o tamanho da tabela é dividido ao meio é cerca de log(n)
  - Entretanto o custo para manter a tabela ordenada é alto
    - Inserção e remoção de elemento são ineficientes devido a necessidade de realocação dos elementos
    - Cada inserção na posição p da tabela implica no deslocamento dos registros a partir da posição p para as posições seguintes.
    - Consequentemente a pesquisa binária não é recomendada para uso em aplicações muito dinâmicas.

Métodos de ordenação

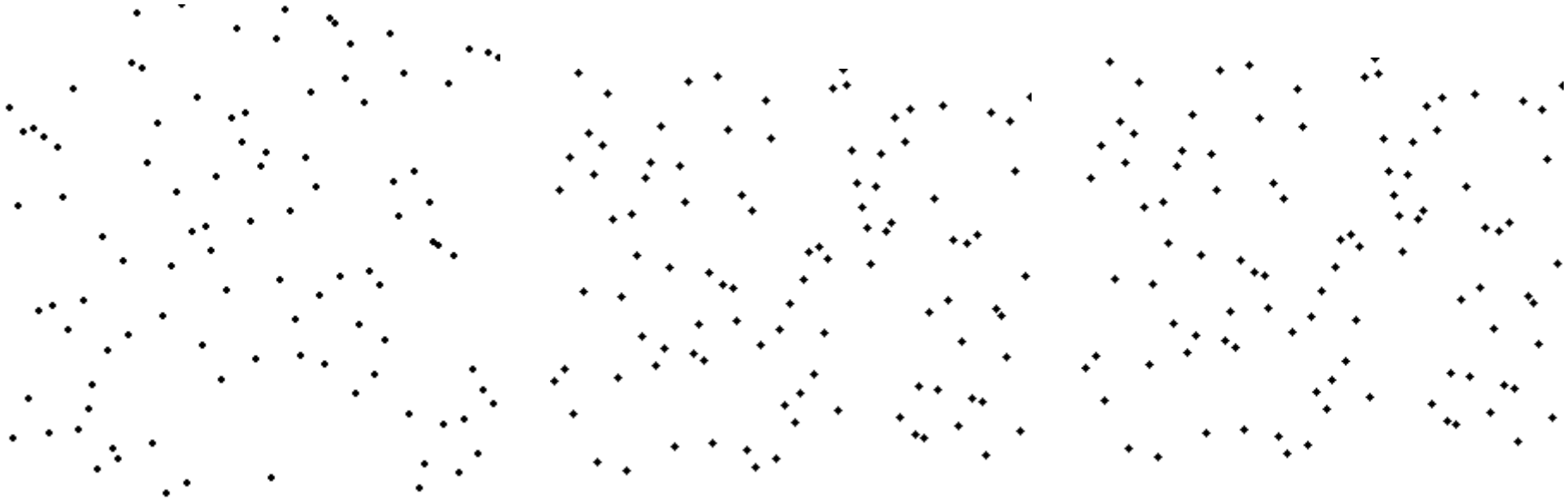
- Ordenar corresponde ao processo de rearranjar um conjunto de objetos em um ordem ascendente ou descendente.
- O objetivo principal da ordenação é facilitar a recuperação dos itens do conjunto ordenado posteriormente.
- A atividade de ordenação está presente na maioria das aplicações onde os objetos tem que ser pesquisados e recuperados

Métodos de ordenação

- Os métodos de ordenação são classificados em dois grandes grupos.
  - Ordenação interna: Conjunto de dados a ser ordenado cabe todo na memória principal.
  - Ordenação Externa: Conjunto de dados não cabe na memória principal, tendo que ser armazenado, por exemplo, em disco.
- O fator predominante na escolha do algoritmo de ordenação é o tempo.

Métodos de Ordenação

- Bolha (*BubbleSort*)
- Seleção (*SelectionSort*)
- Inserção (*InsertionSort*)



Método bolha

- Os elementos vão “borbulhando” a cada iteração do método até a posição correta para ordenação da lista.
- O método pode parar quando nenhum elemento borbulhar/trocar de posição.
- Como os elementos são trocados (borbulhados) frequentemente, há um alto custo de troca de elementos.

Exemplo do método bolha

Suponha que se deseja classificar em ordem crescente o seguinte vetor de chaves [28, 26, 30, 24, 25].

Primeira Varredura

28 26 30 24 25 compara par (28, 26): **troca**
26 28 30 24 25 compara par (28, 30): não troca
26 28 30 24 25 compara par (30, 24): **troca**
26 28 24 30 25 compara par (30, 25): **troca**
26 28 24 25 **30** Maior chave em sua posição definitiva

fim da primeira varredura

Exemplo do método bolha

Segunda Varredura

26 28 24 25 30 compara par (26, 28) : não troca
26 28 24 25 30 compara par (28, 24) : **troca**
26 24 28 25 30 compara par (28, 25) : **troca**
26 24 25 28 30 (não precisa comparar)

Terceira Varredura

26 24 25 28 30 compara par (26, 24) : **troca**
24 26 25 28 30 compara par (26, 25) : **troca**
24 25 26 28 30 (não precisa comparar)

Durante a quarta varredura, nenhuma troca ocorrerá e a execução do algoritmo terminará.

Análise de Desempenho

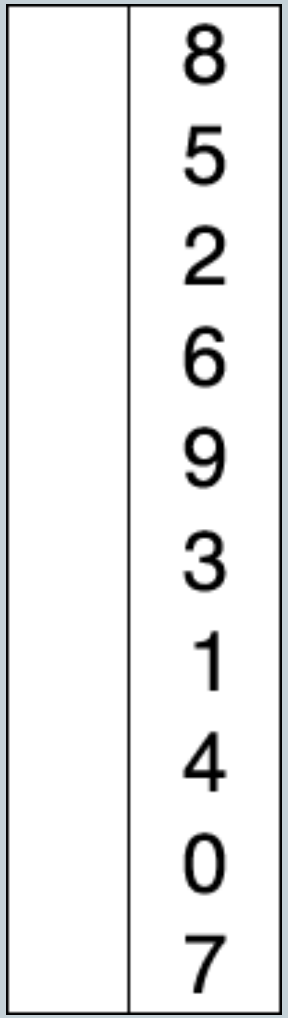
- Melhor caso
  - Quando o vetor já se encontra ordenado → nenhuma troca ocorre na primeira varredura.
  - Custo linear: n - 1 comparações
- Pior caso
  - Quando o vetor se encontra na ordem inversa a desejada.
  - A cada varredura apenas uma chave será colocada em sua posição definitiva.

Código em C

```
void bubblesort(int vet[], int n) {
 int i, j, cond, temp;
 cond = 1;
 for (i=n-1; (i >= 1) && (cond == 1); i--) {
     cond = 0;
     for (j=0; j < i ;j++) {
         if (vet[j+1] < vet[j]) {
             temp = vet[j];
             vet[j] = vet[j+1];
             vet[j+1] = temp;
             cond = 1;
         }
     }
 }
}
```

Método de Seleção

- O método de Seleção é um dos algoritmos mais simples de ordenação, cujo princípio de funcionamento é o seguinte:
- Selecione o menor (ou maior) item da lista em seguida troque-o com o que está na primeira posição da lista
- Repita esta operação com os n-1 itens restantes, depois com os n-2 ate que reste apenas um elemento



Análise de Desempenho

- Vantagens:
  - Custo linear no tamanho da entrada para o número de movimentos de registros.
  - É muito interessante para arquivos pequenos.
- Desvantagens:
  - O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
  - O algoritmo não é **estável.**

Código

```
void selection_sort(int num[], int tam) {
  int i, j, min;
  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) {
              int swap = num[i];
              num[i] = num[min];
              num[min] = swap;
       }
  }
}
```

Método de Inserção

- Algoritmo utilizado pelo jogador de cartas
  - As cartas são ordenadas da esquerda para direita uma por uma.
  - O jogador escolhe a segunda carta e verifica se ela deve ficar antes ou na posição que está.
  - Depois a terceira carta é classificada, deslocando-a até sua correta posição
  - O jogador realiza esse procedimento até ordenar todas as cartas

Análise de Desempenho

- Análise
  - Para arquivos já ordenados o algoritmo tem um custo de O(n).
    - Logo é um método recomendado quando a lista está parcialmente ordenada (pior caso, ordem reversa)
  - Bom método quando se deseja adicionar uns poucos itens a um arquivo já ordenado e depois obter uma lista ordenada (neste caso custo é linear)
  - Algoritmo quase tão simples quanto o algoritmo de ordenação por seleção

Código em C

```
void Insertion(int n, int vetor[]){
  int j,i,key;
  for(j = 1; j < n; j++) {
      key = vetor[j];
      i = j - 1;
      while(i >= 0 && vetor[i] > key) {
            vetor[i + 1] = vetor[i];
            i = i - 1;
      }
      vetor[i + 1] = key;
  }
}
```