Módulo 8

Métodos de Ordenação

Algoritmos e Estruturas de Dados II

C++

(Rone Ilídio)

Métodos de Ordenação

- Inserção *
- Seleção *
- Bolha
- Quicksort *
- Heapsort
- Mergesort *

Animações

http://math.hws.edu/TMCM/java/xSortLab/index.html

Inserção

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

Inserção

8-3-5-4-2-1

8- -5-4-2-1 temp=3

-8-5-4-2-1 temp=3

3-8-5-4-2-1 insere o 3

3-8-5-4-2-1 temp = 5

3-8- -4-2-1 temp = 5

3- -8-4-2-1 temp = 5

3-5-8-4-2-1 insere o 5

3-5-8-4-2-1 temp=4

3-5-8- -2-1 temp=4

3-5- -8-2-1 temp=4

3- -5-8-2-1 temp=4

3-4-5-8-2-1 temp=4

...

Inserção

- Complexidade de tempo
  - Pior caso = vetor ordenado em ordem inversa
- Em relação ao número de comparações v[i]>x :
  - 1 + 2 + ... + (j-1)

  →PA: n = j-1, a1=1, na=j-1
  - Sn = (n (a1+na)) / 2

  →f(n) = ((j-1)(1 + j-1))/2

f(n) = (j(j-1))/2 ==> $O(j^2)$ ou $O(n^2)$

Inserção

- Melhor caso → Vetor ordenado
  - Executará j-1 comparações
  - O for interior só executará uma vez
  - O for exterior executará n-1 vezes
  - f(n) = n-1 → O(n)

Seleção

- Mais simples
- Na primeira iteração o menor elemento é colocado na posição 0
- Na segunda, o segundo maior é colocado na posição 1
- ...
- O maior elemento é colocado na posição n-1

Seleção

```
void selecao(int n, int v[]){
  for (int a=0; a<n-1; a++)
      for(int b=0; b<n; b++)
          if(v[a] > v[b]){
              int aux = v[a];
              v[a] = v[b];
              v[b] = aux;
          }
  }
}
```

Seleção

Complexidade de tempo

- Pior caso = Melhor caso

Em relação ao número de comparações v[a] > v[b] :

1 + 2 + ... + (n-1)

PA: n = j-1, a1=1, na=j-1

Sn = (n (a1+na)) / 2

f(n) = ((n-1)(1 + n-1))/2

f(n) = (n(n-1))/2 ==> $O(n^2)$

Quicksort

- Considerado o mais rápido
- Método
  - Escolhe um elemento como pivô
  - Elementos menores que o pivô vão para a esquerda
  - Elementos maiores que o pivô vão para a direita
  - Posiciona o pivô entre esses conjuntos de elementos
  - Repete a operação duas vezes: para os elementos menores que o pivô e para o os elementos maiores que o pivô

Quicksort

- Necessita de duas funções
  - Separa:
    - Escolhe um pivô
    - Os menores que o pivô vão para a esquerda
    - Os maiores vão para a direita
    - O pivô fica entre esses dois grupos
    - Retorna a posição (j) onde encontra-se o pivô
  - Quicksort
    - Chama *separa* para o vetor inteiro e obtém j
    - Chama recursivamente quicksort de *0* a*(j-1)* e de *(j+1)* a *n-1*

Quicksort

```
int separa(int p, int r, int v[]){
    int c, j, k, t;
    c = v[r];     j=p;
    for(k=p; k<r; k++){
        if(v[k] <= c){
            t=v[j];  v[j]=v[k];  v[k]=t;
            j++;
        }
    }
    v[r] = v[j];   v[j]=c;
    return j;
}
```

Quicksort

```
void quicksort(int p, int r, int v[]){
    int j;
    if (p<r){
        j = separa(p,r,v);
        quicksort(p,j-1,v);
        quicksort(j+1,r,v);
    }
}
```

- Obs: primeira chamada --> quicksort(0, n-1, v)

Quicksort

- Bom exemplo: 8 2 7 4 5
- Complexidade: depende da escolha do pivô
  - Pior caso: n^2 - vetor ordenado
  - Melhor caso: n ($\log_2$ n) – sempre o elemento do meio
  - Caso médio- n ($\log_2$ n)

Mergesort

- Dividir para conquistar
- Ideia básica:
  - fácil criar uma sequência ordenada a partir de duas outras também ordenadas
- Funcionamento
  - divide a sequência original em duas parte
  - agrupa em sequências de quatro elementos
  - executa até que as sequencias tenham dois elementos → ordena
  - Une as sequencias duas a duas até formar um único vetor

```
void intercala(int p, int q, int r, int v[]){
        int i, j, k, *w;
        w = new int[r-p]; //Vetor temporário
        i = p; j= q; k = 0;
        while(i<q && j<r){
                if(v[i] <= v[j]) w[k++] = v[i++];
                else w[k++] = v[j++];
        }
        while (i<q) w[k++] = v[i++];
        while (j<r) w[k++] = v[j++];
        for(i = p; i<r; i++) v[i] = w[i-p];
}
```

```
void mergesort(int p, int r, int v[]){
        if(p<r-1){
                int q = (p+r)/2;
                mergesort(p,q,v);
                mergesort(q,r,v);
                intercala(p,q,r,v);
        }
}
```

Obs: chamada mergesort(0,tam,v);

onde tam é o tamanho de v

Mergesort

8 2 7 1 3 5 6 4 - inicial

8 2 7 1 3 5 6 4 - 2 partes

8 2 7 1 3 5 6 4 - 4 partes

8 2 7 1 3 5 6 4 - 8 partes

ordena cada uma das partes

2 8 1 7 3 5 4 6 -

1 2 7 8 3 4 5 6 - intercala

1 2 3 5 6 7 8 - intercala

Mergesort

- Exemplo:Para um vetor de 8 elementos

Mergesort(0,4) Mergesort(4,8)

Mergesort(0,2) Mergesort(2,4) Mergesort(4,6) Mergesort(6,8)

- Intercala(0,2,4) Intercala(4,6,8)
- Intercala(0,4,8)

Mergesort – Intercalação de vetores

- Definição do problema:
  - Dado um vetor v[p .. r-1]
  - Os sub-vetores são ordenados: v[p..q-1] e v[q..r-1]
- Arranjar o vetor v tal que ele fique totalmente ordenado

p					q-1	q				r-1
1	3	5	5	7	9	9	2	4	7	8

Mergesort

- Complexidade
  - n log n comparações para todos os casos
- Obs: na prática ele é um pouco mais lento que o Quicksort pois ele utiliza um vetor auxiliar e o acesso a ele deixa-o mais lento