Nome: Bárbara Maria Sampaio Portes

- Somatórios

Perturbação

$$S_n + a_{n+1} = a_0 + \sum_{0 \leq i \leq n} a_{i+1}$$

Somatório Importantes:

- $\sum_{0 \leq i \leq n} i = \frac{n.(n+1)}{2}$ (Gauss)
- $\sum_{0 \leq i \leq n} i^2 = \frac{n(1+n)(1+2n)}{6}$ $\qquad \sum_{0 \leq i \leq n} i^3 = \frac{2n^3+3n^2+n}{6}$

Prova por indução

- **1º Passo (passo base)**: Provar que a fórmula é verdadeira para o primeiro valor (na equação substituir n pelo primeiro valor)
- **2º Passo (indução propriamente dita)**: Supondo que n > 0 e que a fórmula é válida quando trocamos n por (n-1)

$$S_n = S_{n-1} + a_n$$

S_{n-1} = é a equação substituindo n por (n-1)

a_n = n-ésimo termo da sequência

- Anotações da aula

➢ Custo total do algoritmo é igual a soma do custo de suas operações;
➢ A menos que dito ao contrário, consideramos o pior caso.
➢ Se um algoritmo for O(n), ele será O de qualquer coisa maior que (n); Se um algoritmo for Ω(n), ele será Ω de qualquer coisa menor que (n); Θ é o justo.

➔ Considerando o número de comparações entre registros da ordenação interna, temos que:

➢ Inserção e seleção fazem Θ (n²) comparações entre registros;
➢ O Quicksort, Mergesort e Heapsort fazem Θ (n*log(n));
➢ O limite inferior para o problema de ordenação é Θ (n*log(n));
➢ É impossível em uma situação normal que a ordenação faça menos que Θ (n*log(n))!
➢ O melhor caso do algoritmo de inserção acontece quando ordenamos um array ordenado de forma crescente. Neste caso, efetuamos Θ (n) comparações;
➢ O algoritmo de ordenação por inserção e a bala de prata para ordenarmos arrays ordenados ou praticamente ordenados, de forma crescente;
➢ O Countingsort realiza Θ (n) comparações para todos os casos para todos os casos, contudo, ele só funciona em situações específicas, inteiros e ele triplica o espaço de armazenamento;
➢ Análise de complexidade do Shellsort é um problema em aberto na computação;
➢ O seleção é o melhor algoritmo em termos de movimentações, ele realiza Θ (n) movimentações.

➔ Estrutura de dados

➢ A fila circular está vazia quando o primeiro == último;
➢ Em nossas estruturas de dados as funções recebem um elemento a ser inserido na estrutura e os de remover retornam o elemento a ser removido.

- Algoritmos de ordenação

1. Bolha:

➢ O problema dos algoritmos de seleção e da bolha é porque eles realizam várias comparações redundantes;
➢ Além disso, a bolha faz um número quadrático de movimentações;
➢ É um algoritmo estável;

```
class Bolha extends Geracao {
   public Bolha(){
      super();
   }
   public Bolha(int tamanho){
      super(tamanho);
   }
   //Algoritmo de ordenacao Bolha.
   @Override
   public void sort() {
        for (int i = (n - 1); i > 0; i--) {
            for (int j = 0; j < i; j++) {
                if (array[j] > array[j + 1]) {
                   swap(j, j+1);
                                }
   }
```

2. Inserção

Melhor caso:
○Efetuamos uma comparação em cada iteração do laço externo
○Repetimos o laço externo (n - 1) vezes
Pior caso:
○Efetuamos i comparações em cada iteração do laço interno
○Repetimos o laço externo (n - 1) vezes

➢ Cada iteração do laço externo tem as movimentações do interno mais duas
➢ $Mi(n) = Ci(n) + 1$

Sendo $Mi(n) = Ci(n) + 1$, no melhor caso, temos:

○$C(n) = (n - 1) = \Theta(n)$
○$M(n) = 2 + 2 + 2 + \ldots + 2$, n-1 vezes $= 2(n-1) = \Theta(n)$

Sendo $Mi(n) = Ci(n) + 1$, no pior caso, temos:

○$C(n) = 1 + 2 + 3 + \ldots + (n-1) = \sum_{0 \leq i \leq (n-1)} i = \frac{(n-1)*n}{2}$

○$M(n) = -1 + \sum_{0 \leq i \leq (n-1)} i = \frac{n(n+1)-2}{2} = \Theta(n^2)$

```
for (int i = 1; i < n; i++) {
      int tmp = array[i];
      int j = i - 1;
      while ( (j >= 0) && (array[j] > tmp) ){
            array[j + 1] = array[j]; // Deslocamento
            j--;
      }
      array[j + 1] = tmp;
}
```

3. Shellsort

A razão da eficiência do algoritmo ainda não é conhecida
Sua análise contém alguns problemas matemáticos difíceis, a começar pela própria sequência de incrementos
•O que se sabe é que cada incremento não deve ser múltiplo do anterior
Conjecturas para o número de comparações dado a seq. de Knuth:
•Conjetura 1: $C(n) = \Theta(n^{1,25})$
•Conjetura 2: $C(n) = \Theta(n(\ln n)^2)$

➢ Vantagens:
•Shellsort é uma ótima opção para arquivos de tamanho moderado
•Sua implementação é simples e requer pouco código

➢ Desvantagens:
•Seu tempo de execução é sensível à ordem inicial do arquivo
•Algoritmo não estável

```
void shellsort() {
   int h = 1;
   do { h = (h * 3) + 1; } while (h < n);
   do {
      h /= 3;
      for(int cor = 0; cor < h; cor++){
         insercaoPorCor(cor, h);
      }
   } while (h != 1);
}
void insercaoPorCor(int cor, int h){
   for (int i = (h + cor); i < n; i+=h) {
      int tmp = array[i];
      int j = i - h;
      while ((j >= 0) && (array[j] > tmp)) {
         array[j + h] = array[j];
         j -= h;
      }
      array[j + h] = tmp;
   }
}
```

```
void insercao(){
   for (int i = 1; i < n; i+= 1) {
      int tmp = array[i];
      int j = i - 1;
      while ((j >= 0) && (array[j] > tmp)) {
         array[j + 1] = array[j];
         j -= 1;
      }
      array[j + 1] = tmp;
   }
}
```

4. Quicksort

Divide o array em duas partes que serão independentemente ordenadas e a combinação de seus resultados produz a solução final
•A parte da esquerda terá elementos menores ou iguais a um pivô
•A parte da direita terá elementos maiores ou iguais a um pivô

Melhor caso :

$$C(n) = 2 * C(\frac{n}{2}) + n = n * lg(n) - n + 1$$

•Existem diversas técnicas para evitar o pior caso como, por exemplo, fazer com que o pivô seja a mediana de três elementos do array
No pior caso, há n/2 trocas em cada execução da função de partição
Nesse caso, o pivô está no meio do array e os elementos superiores estão sistematicamente no início da lista e; os inferiores, no fim
•Lembrando que em cada troca temos 3 movimentações

5. Mergesort

Ordenação por intercalação
•Algoritmo de ordenação do tipo dividir para conquistar
•Normalmente, implementado de forma recursiva e demandando um espaço adicional de memória (não é um algoritmo in-place)
•Dividir sistematicamente o array em subarrays até que os mesmos tenham tamanho um
•Conquistar através da intercalação (ordenada) sistemática de dois em dois subarrays
•Todos os casos:
○Em cada subarray (tamanho k), fazemos k - 1 comparações
○Supondo que o tamanho do array é uma potência de 2, fazemos lg(n) passos
{ C(1) = 0
C(n) = 2C(n/2) + Θ(n) Θ(n*lg(n))
○Movimentamos os elementos de cada subarray duas vezes
{M(1) = 0
M(n) = 2M(n/2) + Θ(n) Θ(n*lg(n))
•Método estável
•Normalmente, implementado de forma recursiva e demandando memória adicional
•Faz Θ(n*lg(n)) comparações nos três casos (melhor, médio e pior)

6. Heapsort

•O Heapsort é um algoritmo de seleção que encontra o maior elemento em uma lista, troca-o com o último e repete o processo
•Sua diferença em relação ao Algoritmo de Seleção é que o Heapsort utiliza um Heap Invertido para selecionar o maior elemento de forma eficiente
•As operações de inserção e remoção podem percorrer um ramo completo da árvore, com comparações e trocas em cada nó
•O pior caso para os números de comparações ou trocas depende da altura da árvore que será lg(n) (árvore balanceada)
•Assim, no pior caso, os números de comparações ou trocas serão Θ(lg(n))
•O número de movimentações é três vezes o de trocas mais as (n-1) movimentações correspondentes às remoções
•Como o número de trocas tem seu limite superior dado pelo de comparações, a complexidade, no pior caso, é Θ(n*lg(n))

7. Countsort

•Inicializar todas as posições do array de contagem com zero
•Para cada elemento do array de entrada, incrementá-lo no de contagem
•Fazer com que o array de contagem seja acumulativo de tal forma que cada posição i armazene o número de elementos menores ou iguais a i
•Sabendo o número de elementos menores ou iguais a i, preencher o array de saída

8. Conclusão

•A vantagem do algoritmo de seleção é seu número de movimentos de registros que é Θ(n)
•O algoritmos de Inserção é interessante para arrays ordenados (ou praticamente)
•Os métodos de Inserção e Countingsort são estáveis
•O Quicksort é o mais eficiente para uma grande variedade de situações
•O pior caso do Quicksort é Θ(n2)
•O pior caso do Mergesort e do Heapsort é Θ(n*lg(n))
•Uma desvantagem do Mergesort é o fato dele “duplicar” sistematicamente os vetores
•Uma vantagem do Quicksort é sua pilha de recursividade reduzida
•O Coutingsort é uma opção que sempre deve ser considerada para a ordenação de inteiros ou similares (e.g., números reais com um número fixo de casas decimais)

Pior caso:

$$C(n) = \Theta(n^2)$$

```
void quicksort(int esq, int dir) {
    int i = esq,   j = dir,   pivo = array[(esq+dir)/2];
    while (i <= j) {
        while (array[i] < pivo)
            i++;
        while (array[j] > pivo)
            j--;
        if (i <= j)
        {   swap(i, j);    i++;    j--;  }
    }
    if (esq < j)
        quicksort(esq, j);
    if (i < dir)
        quicksort(i, dir);
}
```

```
// Algoritmo de ordenacao Mergesort.
void mergesort(int esq, int dir) {
    if (esq < dir){
        int meio = (esq + dir) / 2;
        mergesort(esq, meio);
        mergesort(meio + 1, dir);
        intercalar(esq, meio, dir);
    }
}
public void intercalar(int esq, int meio, int dir){
    int n1, n2, i, j, k;
    //Definir tamanho dos dois subarrays
    n1 = meio-esq+1;
    n2 = dir - meio;
    int[] a1 = new int[n1+1];
    int[] a2 = new int[n2+1];
    //Inicializar primeiro subarray
    for(i = 0; i < n1; i++){
        a1[i] = array[esq+i];
    }
    //Inicializar segundo subarray
    for(j = 0; j < n2; j++){
        a2[j] = array[meio+j+1];
    }
    //Sentinela no final dos dois arrays
    a1[i] = a2[j] = 0x7FFFFFFF;

    //Intercalacao propriamente dita
    for(i = j = 0, k = esq; k <= dir; k++){
        array[k] = (a1[i] <= a2[j]) ? a1[i++] : a2[j++];
```

Ordenado de forma crescente

Algoritmo	500	5000	10000	30000
Inserção	11,3	87	161	-
Seleção	16,2	124	228	-
Shellsort	1,2	1,6	1,7	2
Quicksort	1	1	1	1
Heapsort	1,5	1,6	1,6	1,6

Ordenado de forma decrescente

Algoritmo	500	5000	10000	30000
Inserção	40,3	305	575	-
Seleção	29,3	221	417	-
Shellsort	1,5	1,5	1,6	1,6
Quicksort	1	1	1	1
Heapsort	2,5	2,7	2,7	2,9

Aleatório

Algoritmo	500	5000	10000	30000
Inserção	11,3	87	161	-
Seleção	16,2	124	228	-
Shellsort	1,2	1,6	1,7	2
Quicksort	1	1	1	1
Heapsort	1,5	1,6	1,6	1,6