Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento Acadêmico de Informática (DAINF)
Estrutura de Dados I
Professor: Rodrigo Minetto
Lista de exercícios (revisão)

Exercício 1 - Poscomp) Quais destes algoritmos de ordenação têm a classe de complexidade assintótica, no pior caso, em O(nlogn)

- Quick-Sort, Merge-Sort, e Heap-Sort
- Quick-Sort e Selection-Sort
- **Merge-Sort e Heap-Sort**
- Quick-Sort e Bubble-Sort
- Quick-Sort, Merge-Sort e Selection-Sort

Exercício 2 - Enade) Julgue os itens a seguir, acerca de algoritmos para ordenação.

I) O algoritmo de ordenação por inserção tem complexidade O(nlogn)

II) Um algoritmo de ordenação é dito estável caso ele não altere a posição relativa de elementos de mesmo valor.

III) No algoritmo quicksort, a escolha do elemento pivô influencia o desempenho do algoritmo.

IV) O bubble-sort e o algoritmo de ordenação por inserção fazem, em média, o mesmo número de comparações.

Estão certos apenas os itens

- I e II
- I e III
- II e IV
- I, III e IV
- **II, III e IV**

Exercício 3 - Cormen) Mostre que a complexidade do algoritmo busca binária é $\mathcal{O}(\log n)$. Dica: explore a complexidade do algoritmo através de um modelo de árvore binária.

Mostre, conforme feito várias vezes em aula, que altura da árvore é no máximo $\log n$, ou seja, resolva a equação $n/2^k = 1$, e que o trabalho em cada nível da árvore tem custo constante $\mathcal{O}(1)$, portanto, a soma de todos os custos (altura + níveis) tem custo no pior caso de $\mathcal{O}(\log n)$.

Exercício 4 / Questão de entrevistas) Dada uma permutação $p[1 \ldots n]$, escreva uma função em C para determinar o número de inversões em p com complexidade $\mathcal{O}(n^2)$. O par (i,j) é uma inversão quando $i < j$ e $a_i > a_j$. Por exemplo no vetor $p = \{2, 4, 1, 3, 5\}$, existem 3 inversões, já no vetor $p = \{2, 1, 3, 1, 2\}$ existem 4 inversões. Para qual vetor p podemos ter um número máximo de inversões? Existe uma maneira de resolver este problema em $\mathcal{O}(n \log n)$?

Uma solução ineficiente é usar o algoritmo Bubble-Sort, e a cada troca de elementos adjacentes é uma inversão a menos que o algoritmo remove. Existem soluções melhores com o algoritmo Merge-Sort.

```
void inv_count (int A[], int n) {
  int i, j;
  int inversoes = 0;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n-i-1; j++) {
      if (A[j] > A[j+1]) {
        swap (A, j, j+1);
        inversoes++;
      }
    }
  }
  printf("Inversões: %d.\n", inversoes);
}
```

Qual o número máximo de inversões? n(n-1)/2 (para um vetor em ordem decrescente).

Exercício 5) Dê um exemplo de uma entrada mínima que mostre a não estabilidade do algoritmo de ordenação por seleção.

Se a entrada consistir nos seguintes números: 1 1 0, então a ordenação é realizada da seguinte forma:

```
  (a) (b)
   1   1   0

      (b) (a)
   0   1   1
```

O que mostra a inversão da primera ocorrência do número um com a segunda ocorrência.

Exercício 6 - MIT) Discuta se é possível, para um hacker construir uma entrada com n números distintos para o Quick-Sort aleatorizado que obrigue-o a executar em $\mathcal{O}(n^2)$.

Não é possível. Nenhum adversário ou atacante tem controle sobre quais números aleatórios que o algoritmo usará. Portanto, embora seja verdade que o algoritmo Quick-Sort aleatorizado tenha tempo no pior caso de $\mathcal{O}(n^2)$, nenhum adversário pode forçar esse comportamento.

Exercício 7 - MIT) Qual a relação de recorrência do algoritmo Insertion-Sort supondo uma versão recursiva? E do algoritmo Merge-Sort?

Relação de recorrência do Insertion-Sort: $T(n) = T(n-1) + O(n) = O(n^2)$

Relação de recorrência do Merge-Sort: $T(n) = 2T(n/2) + O(n) = O(n \log n)$

Exercício 8 - USP) O algoritmo Insertion-Sort é codificado da seguinte forma:

```
void insertion_sort (int *A, int n) {
1.  int i;
2.  for (i = 1; i < n; i++) {
3.    int key = A[i];
4.    int j;
5.    for (j = i-1; (j >= 0) && (A[j] > key); j--) {
6.      A[j+1] = A[j];
7.    }
8.    A[j+1] = key;
9.  }
}
```

i) O que acontece se trocarmos "for (j = 1" por "for (j = 0" na linha 2?

ii) O que acontece se trocarmos "(A[j] > key)" por "(A[j] >= key)" na linha 5?

iii) O que acontece se trocarmos "A[j+1] = key" por "A[j] = key" na linha 8?

i) o algoritmo irá funcionar, mas realiza um teste desnecessário.

ii) o algoritmo irá funcionar, mas deixa de ser estável.

iii) o algoritmo deixa de ordenar corretamente, pense no que acontece quando o algoritmo desloca todos os elementos para direita para encaixar um menor elemento nesse conjunto; no final o índice j é positivo? Note que ele se torna -1 no final do for.

Exercício 9 - Google/Facebook/Amazon) O algoritmo Insertion-Sort tem uma etapa de busca da posição para inserir a chave na posição correta. No entanto, ele utiliza uma busca linear para esta tarefa — conforme pode ser visto nas linhas 5—7 do algoritmo mostrado no exercício 8. Pergunta: porque não utilizar uma **busca binária** para otimizar essa tarefa já que todos os elementos anteriores a chave estão ordenados? A complexidade do pior caso do Insertion-Sort pode ser melhorada como essa otimização?

A complexidade do pior caso do algoritmo Insertion-Sort, mesmo com esta otimização, continua sendo $\mathcal{O}(n^2)$. Isso ocorre porque a inserção de um dado em uma posição apropriada envolve duas etapas: 1) calcular a posição; 2) deslocar os dados da posição calculada no passo 1) um passo para a direita para criar uma lacuna onde os dados

serão inseridos. O uso da pesquisa binária reduz a complexidade de tempo na etapa 1 de $\mathcal{O}(n)$ para $\mathcal{O}(\log n)$, no pior caso. Mas, a complexidade de tempo na etapa 2) ainda permanece $\mathcal{O}(n)$.

Exercício 10 - Questão de entrevista) Dado um array com n intervalos de tempo para um conjunto de reuniões, tal que cada intervalo de tempo é definido através de um par com o início e fim, determine se um funcionário pode participar de todas as n reuniões. A saída deve ser um booleano tal que **true** indica que todas as reuniões não têm sobreposição de horário, e **false** caso contrário. Por exemplo:

```
Entrada:                                  Saída

Reunioes R[3] = {{15,20},{0,30},{5,10}};     False

Reunioes R[2] = {{7,10},{2,4}};              True
```

Desenvolva um algoritmo trivial, por força bruta, para resolver o problema acima em $\mathcal{O}(n^2)$, e uma versão otimizada com complexidade $\mathcal{O}(n \log n)$.

Um algoritmo trivial consiste em comparar todos os pares $\mathcal{O}(n^2)$. Uma versão otimizada consiste em ordenar o vetor pelos tempos de início e então percorrer o vetor de reuniões uma única vez comparando o tempo de término da reunião atual com o tempo de início da próxima. Veja o código reunioes.c para uma solução completa.