Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Algoritmos e Estruturas de Dados
Semestre de Inverno 2017/18
Primeira série de problemas
Parte 1

Observações:

- Data de entrega: **16 de outubro de 2017**.
- No contexto desta série, o triplo *(v, l, r)* representa o *subarray* do *array* v, compreendido entre os índices l e r inclusivé.

1 Algoritmos Elementares

1. Considere o método

   ```
   public static int printEachThreeElementsThatSumTo(int[] v, int l, int r, int s)
   ```

   que dado o *subarray* (v, l, s) e o inteiro s, apresenta na consola todos os triplos de elementos distintos desse *array* tal que a sua soma seja igual a s. O método retorna o número de triplos encontrados.

   Sendo n a dimensão do *subarray* e, se possível,

   1.1. Realize uma implementação usando um algoritmo com custo assimptótico $O(n^3)$.

   1.2. Realize uma implementação usando um algoritmo com custo assimptótico $O(n^2 \log n)$.

   1.3. Realize uma implementação usando um algoritmo com custo assimptótico $O(n^2)$.

   Avalie experimentalmente as implementações anteriores. Apresente os resultados graficamente, utilizando uma escala adequada.

2. Realize o método

   ```
   public static int removeIndexes(int[] v, int l, int r, int[] vi, int li, int ri)
   ```

   que retira da sequência representada pelos inteiros do *subarray* (v, l, r) os elementos cujos índices originais estão presentes no *subarray* (vi, li, ri), que está ordenado de forma crescente. A sequência resultante fica contida de forma contígua nas primeiras posições do *subarray* (v, l, r) mantendo a ordem relativa dos elementos. O valor retornado pelo método é a dimensão da sequência resultante. Valorizam-se as soluções que não criem *arrays* auxiliares e que tenham complexidade $O(n)$, com $n = r - l + 1$.

3. Realize o método

   ```
   public static String greaterCommonPrefix(String[] v, int l, int r, String word)
   ```

   que, dado o *subarray* (v, l, r), ordenado de modo crescente e a palavra *word*, retorna a palavra presente no *subarray* (v, l, r) que contenha o maior prefixo que seja comum a um prefixo da palavra *word*. Em caso de empate, retorna a *string* que se encontra na posição de maior valor no *subarray* (v, l, r). Por exemplo, considerando o *subarray* $(v, 0, 2)$, em que [“agendar”, “dia”, “teste”] e a palavra “agendas” , o maior prefixo comum é “agenda” e portanto o método retorna a palavra “agendar”. No entanto, considerando o mesmo *subarray* e a palavra “caneta”, o maior prefixo comum é a *string* vazia, e portanto o método retorna a palavra “teste”. Caso o *subarray* (v, l, r), não contenha palavras, o método deverá retornar `null`.

4. Realize o metodo

   ```
   public static int sumGivesN(int n)
   ```

   que, dado um número inteiro positivo n, escreve no *standard output* todas as sequências de números consecutivos cuja soma é n. Este método retorna o número de resultados encontrados. Por exemplo, quando n é 24, existem duas sequências: a sequência 7, 8 e 9, e a sequência 24. Valorizam-se as soluções com tempo $O(n)$.

5. Realize o método estático,

   ```
   public static int deleteMin(int[] maxHeap, int sizeHeap)
   ```

   que, recebendo um *max-heap* de elementos distintos e de dimensão `sizeHeap`, remove se possível, o menor elemento presente no *heap*. O método retorna a nova dimensão do *heap*. O *array* `maxHeap` poderá ser modificado. Indique justificando, a complexidade do algoritmo.

2 Análise de desempenho

1. Considere os seguintes algoritmos:

   - (Algoritmo 1): O algoritmo 1 resolve um problema de dimensão N, ao dividir recursivamente o problema em dois subproblemas de dimensão N/2 e combina os resultados em tempo linear.
   - (Algoritmo 2): O algoritmo 2 resolve um problema de dimensão N, recursivamente, através da resolução de um subproblema de dimensão N/2 e realiza algum processamento em tempo constante.

   1.1. Para cada um destes algoritmos, indique a equação de recorrência.

   1.2. Dê um exemplo de algoritmo estudado nas aulas semelhante a cada um destes

2. Considere o algoritmo *xpto*.

   ```
   public static long xpto(long x, long n){
     if (n == 0) return 1;
     if (n % 2 == 0) return xpto(x, n/2) * xpto(x, n/2);
     return xpto(x, n/2) * xpto(x, n/2) * x;
   }
   ```

   2.1. Indique a complexidade do mesmo, no pior caso, em função de n. Indique a equação de recorrência e resolva-a.

   2.2. Será possível diminuir a complexidade deste método, mantendo a mesma funcionalidade? Justifique.

3. Considere os algoritmos de ordenação estudados.

   3.1. Indique, justificando, um algoritmo cujo custo, quaisquer que sejam os valores presentes na sequência a ordenar, pertence a $O(n^2)$ mas não pertence a $\Theta(n^2)$

   3.2. Indique, justificando, um algoritmo cujo custo, quaisquer que sejam os valores presentes na sequência a ordenar, pertence a $O(n \log n)$ mas não pertence a $\Theta(n \log n)$.

4. Um algoritmo de ordenação é *parsimonioso* se nenhum par de elementos é comparado mais do que uma vez. Considerando as implementações das aulas e do livro recomendado da disciplina, quais dos seguintes algoritmos de ordenação (`insertion-sort`, `selection-sort`, `heap-sort` e `merge-sort`) são parsimoniosos? Justifique.