Introdução a Análise de Algoritmos
2º semestre de 2017 - Turma 04
Lista de exercícios 3

1. Em quais situações uma busca sequencial pode ser mais vantajosa do que técnicas de busca mais sofisticadas (busca binária, tabelas de espalhamento)?

2. Que condições devem ser satisfeitas para que, em uma tabela de espalhamento em que colisões são tratadas por encadeamento, a operação de busca tenha complexidade $\Theta(1)$ na média? Justifique.

3. O que caracteriza uma boa função de *hashing*?

4. Simule a inserção dos valores $\{279, 139, 43, 75, 27, 219, 407, 103, 123, 183, 235, 283, 231, 327, 539, 71\}$ em uma tabela de espalhamento na qual as colisões são tratadas por encadeamento. Considere que a estrutura de lista ligada usada é simplesmente ligada, que apenas é mantido o ponteiro para o primeiro elemento da lista e que inserções são sempre feitas no início das lista. Considere ainda que a função de hashing é definida por $h(k) = k \ \% \ q$, onde q (a quantidade de posições da tabela) é igual a 8.

5. A partir da tabela obtida no exercício anterior, descreva o que acontece durante a busca pelos seguintes valores: $279, 139, 103, 327, 71$.

6. Podemos afirmar que a função de *hashing* usada nos exercícios 4 e 5 é uma boa função? Justifique.

7. Explique o princípio de funcionamento do algoritmo de ordenação *insertion sort*. Mostre que sua complexidade de tempo de execução é $\Omega(n)$ e $O(n^2)$.

8. Explique o princípio de funcionamento do algoritmo de ordenação *selection sort*. Mostre que sua complexidade de tempo de execução é $\Theta(n^2)$.

9. Explique o princípio de funcionamento do algoritmo de ordenação *heap sort*. Mostre que sua complexidade de tempo de execução é $O(nlgn)$.

10. Explique o princípio de funcionamento do algoritmo de ordenação *merge sort*. Mostre que sua complexidade de tempo de execução é $\Theta(nlgn)$.

11. O que podemos observar ao comparar os algoritmos de ordenação *selection sort* e *heap sort*?

12. Simule a execução dos algoritmos de ordenação *insertion sort*, *selection sort*, *heap sort* e *merge sort* para os seguintes valores: $7, 5, 2, 3, 1, 6, 8, 4$.

13. Considere a seguinte variação do algoritmo de ordenação *merge sort*:

```
public static void merge_sort(int [] a, int ini, int fim){

  if(ini < fim){

    int q1 = ini + (fim - ini) / 4;

    merge_sort(a, ini, q1);
    merge_sort(a, q1 + 1, fim);
    merge(a, ini, q1, fim);
  }
}
```

a) O desempenho do método/função *merge* é afetado devido às modificações desta versão? Justifique.

b) E quanto ao desempenho do *merge sort* como um todo, há alguma alteração? Justifique.