SSC0503 - Introdução à Ciência de Computação II
6ª Lista - Tópico 5
Professor: Claudio Fabiano Motta Toledo (`claudio@icmc.usp.br`)
Estagiário PAE: Jesimar da Silva Arantes (`jesimar.arantes@usp.br`)

---

1. Com base em suas experiências marque a alternativa correta que descreve de forma geral (caso médio) a eficiência dos métodos de ordenação (do menos eficiente ao mais eficiente) para entradas grandes.

   - (a) heap sort, merge sort, quick sort, bubble sort, select sort, insertion sort.
   - (b) bubble sort, select sort, insertion sort, heap sort, merge sort, quick sort.
   - (c) bubble sort, select sort, insertion sort, merge sort, heap sort, quick sort.
   - (d) bubble sort, select sort, insertion sort, merge sort, quick sort, heap sort.
   - (e) bubble sort, select sort, merge sort, insertion sort, quick sort, heap sort.

2. Qual é a menor profundidade possível de uma folha em uma árvore de decisão para uma ordenação por comparação?

3. Qual é a maior profundidade possível de uma folha em uma árvore de decisão para uma ordenação por comparação?

4. Qual a quantidade de nós folhas em uma árvore de decisão com n entradas?

5. Acesse a página e veja o funcionamento dos algoritmos counting sort, radix sort e bucket sort, passo a posso.

   - *https://www.cs.usfca.edu/∼ galles/visualization/CountingSort.html*
   - *https://www.cs.usfca.edu/∼ galles/visualization/RadixSort.html*
   - *https://www.cs.usfca.edu/∼ galles/visualization/BucketSort.html*

   Obs: ao copiar e colar o link o símbolo ∼ costuma dar problema, digite-o você mesmo.

6. Simule a execução do counting sort usando como entrada os seguintes vetores:

   - $A = [7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3]$.
   - $A = [6, 0, 2, O, 1, 3, 4, 6, 1, 3, 2]$.

7. Simule a execução do radix sort usando como entrada os seguintes vetores:

   - $A = [713, 131, 312, 124, 245, 457, 572, 724, 243, 437]$.
   - $A = [COW, DOG, SEA, NOW, ROW, FOX, BIG, BOX, TAB, BAR]$.

8. Simule a execução do bucket sort usando como entrada os seguintes vetores:

   - $A = [0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42]$.

- $A = [79, 13, 16, 64, 39, 20, 89, 53, 71, 42]$.

9. Desenvolva um programa em C que faça a ordenação através do método counting sort sobre um vetor de tamanho N. Em seguida, diga qual a análise de complexidade no melhor caso, pior caso e caso médio.

10. Desenvolva um programa em C que faça a ordenação através do método radix sort sobre um vetor de tamanho N. Em seguida, diga qual a análise de complexidade no melhor caso, pior caso e caso médio.

11. Desenvolva um programa em C que faça a ordenação através do método bucket sort sobre um vetor de tamanho N. Em seguida, diga qual a análise de complexidade no melhor caso, pior caso e caso médio.

12. Descreva sobre quais condições o algoritmo counting sort pode ser aplicado (sem modificações).

13. Descreva em alto nível como adaptar o counting sort para ordenar números não inteiros no intervalo de 0 a 1 com três casas decimais.

14. Desenvolva um programa em C que implemente as adaptações do exercício anterior (counting sort para números não inteiros com três casas decimais).

15. Descreva sobre quais condições o algoritmo bucket sort pode ser aplicado (sem modificações).

16. Descreva em alto nível como adaptar o bucket sort para ordenar números fora do intervalo $[0, 1)$, por exemplo $[0, 1000]$. Observação: os números ainda devem seguir uma distribuição uniforme.

17. Implementar os algoritmos counting sort, radix sort e bucket sort, realizando experimentos que avaliem a quantidade de operações (comparações) e o tempo de execução para:

    - Um vetor com 1.000, 10.000, 100.000 e 1.000.000 de números inteiros entre 0 e 99.999 gerados aleatoriamente.
    - Comparar estes algoritmos com o quickSort para os mesmos vetores.

18. Reescreva a linha 10 do algoritmos counting-sourt (como descrito em Cormen) como: *10 for j D 1 to A:length*. Demonstre que o algoritmo ainda funciona perfeitamente. O algoritmo continua estável?

19. Escreva um algoritmo que, dado n inteiros entre 0 e k, pré-processe sua entrada e seja capaz de retornar quantos dos n elementos estão no intervalo $[a \cdots b]$ em $O(1)$. Qual a complexidade do seu algoritmo? Não pode ser pior que $\Theta(n + k)$.

20. Baseado na Figura 8.3, Cormen 3a ed., ilustre o funcionamento do algoritmo radix-sort usando a seguinte lista de palavras em inglês: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

21. Quais dos seguintes algoritmos de ordenação são estáveis: insertion sort, merge sort, heapsort, and quicksort? Proponha um esquema simples capaz de tornar qualquer algoritmo de ordenação estável. Quanto tempo e espaço adicional seu esquema necessitará?

22. Prove por indução que radix sort funciona.

23. Mostre como ordenar n inteiros no intervalo de 0 até $n^3 - 1$ em tempo $O(n)$.

24. Explique porquê o pior caso do algoritmo bin-sort é $\Theta(n^2)$. Qual mudança simples no algoritmo preservará suas performance média linear e torna seu pior caso $O(n \cdot lg\ n)$.