Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Sistemas e Computação
Graduação em Ciência da Computação

Algoritmos de ordenação básicos

1. Qual a principal característica básica (lógica/sistemática) de cada algoritmo a seguir: bubble sort, selection sort e insertion sort?
2. Considerando os trechos de código abaixo, qual seriam esses algoritmos de ordenação? Por quê?

```
procedure XXXXXX( A : list of sortable items )
  n = length(A)
  for each i in 1 to n-1 inclusive {
   for each j in n-1 to i inclusive{
      if A[j-1] > A[j] {
       swap(A[j-1], A[j])
      }
   }
  }
end procedure
```

```
XXXXXXXXXX-SORT(A)
  n = length(A)
  for i ← n downto 2 {
      max ← i
      for j ← 1 to i {
         if A[j] > A[max] {
            max ← j
         }
      }
      swap (A[i], A[max])
   }
}
```

3. Considere o algoritmo do selection sort, cujo loop interno faz a seleção do menor elemento. Depois que o elemento é selecionado, ele é trazido para sua posição definitiva em apenas uma troca. Considerando a entrada A = [1, 2, 4, 3, 7, 5, 6], quantas trocas seriam feitas pelo algoritmo do selection sort aplicado a essa entrada?
4. O bubble sort recursivo pode ser pensado com o seguinte design: o array de entrada tem tamanho N (indexado de 0 a N-1). Caso o "pedaço" do array a ser ordenado tenha apenas 1 elemento então não faz nada. Caso contrário, o algoritmo "empurra" o maior elemento do "pedaço" considerado para o final e chama recursivamente considerando um pedaço menor (excluindo o último elemento).
    a. Escreva o algoritmo
    b. Qual seria a relação de recorrência desse algoritmo?
    c. Qual o tempo do algoritmo segundo sua relação de recorrência.

5. O algoritmo odd-even sort é uma variação do bubble sort. Ele funciona através da comparação de todos os pares indexados com ímpar de elementos adjacentes. Se um par está na ordem errada (A[i] A[i+1] estão na ordem errada), os elementos são trocados. O próximo passo repete isso para os pares indexados com par de elementos adjacentes. E o algoritmo continua até que a entrada esteja ordenada. Você pode pensar no algoritmo como se a cada iteração um bubblesort é aplicado considerando índices ímpares e outro considerando índices pares (as trocas são feitas entre elementos A[i] e A[i+1]). Quando não houver mais trocas significa que o array está ordenado. Implemente o odd-even sort e calcule seu tempo de execução.

Algoritmos de ordenação baseados em dividir para conquistar

1. O tempo do merge sort depende da ordem dos valores do vetor a ser ordenado? Explique detalhadamente sua resposta.
2. O que mais afeta a performance do quicksort?
3. O quicksort mediana de três é uma variação do quicksort que tenta minimizar o problema de pivôs ruins. Seu princípio é o seguinte:
   a. Ordene os elementos A[inicio], A[meio], A[fim] do array
   b. Escolha o A[meio] como pivô, traga ele para a primeira posição
   c. Aplique o quicksort normalmente ao array (de inicio até fim)

   Essa variação degrada a performance do quicksort? Por quê?
4. O Stooge sort é um algoritmo de ordenação que utiliza a estratégia de dividir para conquistar. Seu código é mostrado abaixo:

```
STOOGE-SORT(A, i, j)
1  if A[i] > A[j]
2     then exchange A[i] ↔ A[j]
3  if i + 1 ≥ j
4     then return
5  k ← ⌊(j − i + 1)/3⌋          ▷ Round down.
6  STOOGE-SORT(A, i, j − k)     ▷ First two-thirds.
7  STOOGE-SORT(A, i + k, j)     ▷ Last two-thirds.
8  STOOGE-SORT(A, i, j − k)     ▷ First two-thirds again.
```

   a. Argumente porque o stooge sort realmente ordena a entrada
   b. Encontra a relação de recorrência do algoritmo
   c. Compare a performance do stooge sort com os algoritmos quicksort e mergesort.

Ordenação em tempo linear

1. Ilustre a execucao do radix sort considerando a entrada: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR,TAR, DIG, BIG, TEA, NOW, FOX.
2. Ilustre a execução do bucket sort com 5 baldes considerando a entrada: 79, 13, 16, 64,39, 20,89, 53, 71, 42
3. [POSCOMP] Considerando seu conhecimento sobre os algoritmos de ordenação, julgue as alternativas a seguir:
   a. Seja A[1,n] um vetor não ordenado de inteiros com um número constante k de valores distintos. Então existe um algoritmo de ordenação por contagem que ordena A em tempo linear.
   b. Seja A[1,n] um vetor não ordenado de inteiros com um número constante k de valores distintos, então o limite inferior para um algoritmo de ordenação por comparações para ordenar A é de O(n lg n).
   c. Seja A[1, n] um vetor não ordenado de inteiros, cada inteiro com no máximo d dígitos, onde cada dígito assume um valor entre um número constante k de valores distintos. Então o problema de ordenar A tem limite inferior O(n).
   d. Seja A[1, n] um vetor não ordenado de inteiros, cada inteiro com no máximo d dígitos, onde cada dígito assume um valor entre O(n) valores distintos. Então o problema de ordenar A tem limite inferior O(n lg n).

e. Seja A[1,n] um vetor não ordenado de inteiros com um número constante k de valores distintos, então um um algoritmo de ordenação por comparação ótimo para ordenar A tem complexidade O(n lg n)

4. Dado um array com números variando entre 1 até n^6 (n é o tamanho do array). É possível ordenar o array em tempo linear? Justifique sua resposta.

Busca binária e estatística de ordem

1. Tem-se um array de elementos possivelmente desordenados e distintos. Deseja-se encontrar a k-esima estatística de ordem desse array. Comente como você resolveria esse problema fazendo uso das ideias usadas no algoritmo do quicksort. Procure dar detalhes suficientes para deixar sua ideia completamente clara e detalhar o tempo de seu algoritmo (melhor e pior caso)
2. O floor de um número x é o número menor que mais se aproxima de x. Considere um array ordenado e um número x (podendo pertencer ou não ao array). Implemente um algoritmo que busca o floor de x no array usando a estratégia de busca binária. O algoritmo deve ter a seguinte assinatura: `int floor (int[] array, int x).`
3. Implemente um algoritmo baseado em busca binária que calcula a raiz quadrada de um número x. Note que, se o algoritmo encontrar a raiz quadrada exata então deve parar. Caso contrário, o algoritmo continua o cálculo até que um erro mínimo seja atingido. A assinatura do algoritmo deve ser: `double raizQuadrada(double x, double erro).`