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

Resposta Pergunta 1: (b) bubble sort, select sort, insertion sort, heap sort, merge sort, quick sort.

Resposta Pergunta 2: A menor profundidade possível para uma árvore de decisão com n elementos é n. Que corresponde ao ramo da árvore onde todos os elementos estão em ordem crescente ou decrescente.

Resposta Pergunta 3: A maior profundidade possível para uma árvore de decisão com n elementos é $n \cdot lg\ n$. Que corresponde ao ramo da árvore com o pior embaralhamento possível dos nós.

Resposta Pergunta 4: Seja h a maior altura da árvore de decisão. Seja n a quantidade de elementos da árvore. A quantidade de permutações da entrada n é dada por $n!$. Como os nós folhas são dados pelas permutações da entrada então a árvore de decisão deve ter no mínimo $n!$ elementos. Por se tratar de uma árvore binária (desbalanceada) logo a quantidade de nós folha deve ser no máximo 2^h elementos. Portanto, a quantidade de nós folha l é dado pela seguinte desigualdade $n! \leq l \leq 2^h$.

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

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

$A = 7, 1, 3, 1, 2, 4, 5, 7, 2, 4, 3$
$C = 0, 2, 2, 2, 2, 1, 0, 2$
$C = 0, 2, 4, 6, 8, 9, 9, 11$
$B = x, x, x, x, x, 3, x, x, x, x, x$
$C = 0, 2, 4, 5, 8, 9, 9, 11$
$B = x, x, x, x, x, 3, x, 4, x, x, x$
$C = 0, 2, 4, 5, 7, 9, 9, 11$
$B = x, x, x, 2, x, 3, x, 4, x, x, x$
$C = 0, 2, 3, 5, 7, 9, 9, 11$
$B = x, x, x, 2, x, 3, x, 4, x, x, 7$
$C = 0, 2, 3, 5, 7, 9, 9, 10$
$B = x, x, x, 2, x, 3, x, 4, 5, x, 7$
$C = 0, 2, 3, 5, 7, 8, 9, 10$
$B = x, x, x, 2, x, 3, 4, 4, 5, x, 7$
$C = 0, 2, 3, 5, 6, 8, 9, 10$
$B = x, x, 2, 2, x, 3, 4, 4, 5, x, 7$
$C = 0, 2, 2, 5, 6, 8, 9, 10$

$B = x, 1, 2, 2, x, 3, 4, 4, 5, x, 7$
$C = 0, 1, 2, 5, 6, 8, 9, 10$
$B = x, 1, 2, 2, 3, 3, 4, 4, 5, x, 7$
$C = 0, 1, 2, 4, 6, 8, 9, 10$
$B = 1, 1, 2, 2, 3, 3, 4, 4, 5, x, 7$
$C = 0, 0, 2, 4, 6, 8, 9, 10$
$B = 1, 1, 2, 2, 3, 3, 4, 4, 5, 7, 7$
$C = 0, 0, 2, 4, 6, 8, 9, 9$
$B = 1, 1, 2, 2, 3, 3, 4, 4, 5, 7, 7$
$C = 0, 0, 2, 4, 6, 8, 9, 9$
Vetor ordenado:
$B = 1, 1, 2, 2, 3, 3, 4, 4, 5, 7, 7$

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

$A = 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2$
$C = 0, 0, 0, 0, 0, 0, 0$
$C =, 2, 2, 2, 2, 1, 0, 2$
$C = 2, 4, 6, 8, 9, 9, 11$
$B = x, x, x, x, x, 2, x, x, x, x, x$
$C = 2, 4, 5, 8, 9, 9, 11$
$B = x, x, x, x, x, 2, x, 3, x, x, x$
$C = 2, 4, 5, 7, 9, 9, 11$
$B = x, x, x, 1, x, 2, x, 3, x, x, x$
$C = 2, 3, 5, 7, 9, 9, 11$
$B = x, x, x, 1, x, 2, x, 3, x, x, 6$
$C = 2, 3, 5, 7, 9, 9, 10$
$B = x, x, x, 1, x, 2, x, 3, 4, x, 6$
$C = 2, 3, 5, 7, 8, 9, 10$
$B = x, x, x, 1, x, 2, 3, 3, 4, x, 6$
$C = 2, 3, 5, 6, 8, 9, 10$
$B = x, x, 1, 1, x, 2, 3, 3, 4, x, 6$
$C = 2, 2, 5, 6, 8, 9, 10$
$B = x, 0, 1, 1, x, 2, 3, 3, 4, x, 6$
$C = 1, 2, 5, 6, 8, 9, 10$
$B = x, 0, 1, 1, 2, 2, 3, 3, 4, x, 6$
$C = 1, 2, 4, 6, 8, 9, 10$
$B = 0, 0, 1, 1, 2, 2, 3, 3, 4, x, 6$
$C = 0, 2, 4, 6, 8, 9, 10$
$B = 0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6$
$C = 0, 2, 4, 6, 8, 9, 9$
$B = 0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6$
$C = 0, 2, 4, 6, 8, 9, 9$
Vetor ordenado:

$B = 0, 0, 1, 1, 2, 2, 3, 3, 4, 6, 6$

Resposta Pergunta 7:

- $A = [713, 131, 312, 124, 245, 457, 572, 724, 243, 437]$.

  Seja d o digito que estou ordenando, onde $d = 1$ indica o digito mais a esquerda.

  $d = 1$ então $131, 312, 572, 713, 243, 124, 724, 245, 457, 437$
  $d = 2$ então $312, 713, 124, 724, 131, 437, 243, 245, 457, 572$
  $d = 3$ então $124, 131, 243, 245, 312, 437, 457, 572, 713, 724$

- $A = [COW, DOG, SEA, NOW, ROW, FOX, BIG, BOX, TAB, BAR]$.

  Seja d o digito que estou ordenando, onde $d = 1$ indica o digito mais a esquerda.

  $d = 1$ então $SEA, TAB, DOG, BIG, BAR, COW, NOW, ROW, FOX, BOX$
  $d = 2$ então $TAB, BAR, SEA, BIG, DOG, COW, NOW, ROW, FOX, BOX$
  $d = 3$ então $BAR, BIG, BOX, COW, DOG, FOX, NOW, ROW, SEA, TAB$

Resposta Pergunta 9:

```c
void counting_sort(int A[], int n, int k){
 int C[k];
 for (int i = 0; i < k; i++){
  C[i] = 0;
 }
 for (int j = 0; j < n; j++){
  C[A[j]] = C[A[j]] + 1;
 }
 for (int i = 1; i < k; i++){
  C[i] = C[i] + C[i-1];
 }
 int B[n];
 for (int j = n-1; j >= 0; j--){
  B[C[A[j]]-1] = A[j];
  C[A[j]] = C[A[j]] - 1;
 }
 for (int j = 0; j < n; j++){
  A[j] = B[j];
 }
}
```

Listing 1: Resposta do exercício 9 codificado na linguagem C

Análise de Complexidade:
Pior Caso: $\Theta(n+k)$
Caso Médio: $\Theta(n+k)$
Melhor Caso: $\Theta(n+k)$

Resposta Pergunta 10:

```c
#include <stdio.h>

void print_vector(int vetor[], int n);
void radix_sort(int A[], int n, int d);
void counting_sortRadix(int A[], int n, int k, int d);
int getDigit(int v, int d);

int main(){
 int vetor[] = {329, 457, 657, 839, 436, 720, 355};
 int n = 7;
 int d = 3;
 printf("====radix_sort====\n");
 printf("vetor desordenado\n");
 print_vector(vetor, n);
 radix_sort(vetor, n, d);
 printf("vetor ordenado\n");
 print_vector(vetor, n);
 return 0;
}

void radix_sort(int A[], int n, int d){
 for (int i = 0; i < d; i++){
  counting_sortRadix(A, n, 10, i);
 }
}

void counting_sortRadix(int A[], int n, int k, int d){
 int C[k];
 int B[n];
 for (int i = 0; i < k; i++){
  C[i] = 0;
 }
 for (int j = 0; j < n; j++){
  int p = getDigit(A[j], d);
  C[p] = C[p] + 1;
 }
 for (int i = 1; i < k; i++){
  C[i] = C[i] + C[i-1];
 }
 for (int j = n-1; j >= 0; j--){
  int p = getDigit(A[j], d);
  B[C[p]-1] = A[j];
  C[p] = C[p] - 1;
 }
 for (int j = 0; j < n; j++){
  A[j] = B[j];
```

```
  }
}

int getDigit(int v, int d){
 if (d == 0){
  return v%10;
 }else if (d == 1){
  return (v/10)%10;
 }else if (d == 2){
  return (v/100)%10;
 }else if (d == 3){
  return (v/1000)%10;
 }else if (d == 4){
  return (v/10000)%10;
 }else if (d == 5){
  return (v/100000)%10;
 }else if (d == 6){
  return (v/1000000)%10;
 }else if (d == 7){
  return (v/10000000)%10;
 }else if (d == 8){
  return (v/100000000)%10;
 }else if (d == 9){
  return (v/1000000000)%10;
 }
 return -1;
}

void print_vector(int vetor[], int n){
 for (int i = 0; i < n; i++){
  printf("%d ", vetor[i]);
 }
 printf("\n");
}
```

Listing 2: Resposta do exercício 10 codificado na linguagem C

Análise de Complexidade:
Pior Caso: $\Theta(d(n+k))$
Caso Médio: $\Theta(d(n+k))$
Melhor Caso: $\Theta(d(n+k))$

Resposta Pergunta 11:

```
#define tam_bucket 100
#define num_bucket 10

typedef struct {
  int topo;
  int balde[tam_bucket];
}bucket;

void bucket_sort(int vetor[],int n){
 bucket b[num_bucket];
```

```
  for(int i = 0; i <num_bucket; i++){
12   b[i].topo = 0;
  }
14  for(int i = 0; i < n; i++){
    int j = (num_bucket)-1;
16   while(1){
      if(j < 0){
18     break;
      }
20    if(vetor[i] >= j*10){
       b[j].balde[b[j].topo] = vetor[i];
22     (b[j].topo)++;
       break;
24    }
      j--;
26   }
  }
28  for(int i = 0; i < num_bucket; i++){
    if(b[i].topo){
30   insertion_sort(b[i].balde, b[i].topo);
    }
32  }
  int i = 0;
34  for(int j = 0; j < num_bucket; j++){
    for(int k = 0; k < b[j].topo; k++){
36   vetor[i]=b[j].balde[k];
     i++;
38  }
  }
40 }

42 void insertion_sort(int vetor[], int n){
  for (int j = 1; j < n; j++){
44   int chave = vetor[j];
    int i = j-1;
46  while (i >= 0 && vetor[i] > chave){
     vetor[i+1] = vetor[i];
48   i--;
    }
50  vetor[i+1] = chave;
  }
52 }
```

Listing 3: Resposta do exercício 11 codificado na linguagem C

Análise de Complexidade:
Pior Caso: $O(n^2)$
Caso Médio: $\Theta(n+k)$
Melhor Caso: $\Omega(n+k)$

Resposta Pergunta 12: O counting sort pode ser aplicado sobre números inteiros, o ideal é que o intervalo desses números inteiros seja pequeno. Para a sua utilização sobre os números reais adaptações devem ser feitas. Para a sua utilização sobre caracteres al-

fanuméricos também são necessárias adaptações.

Resposta Pergunta 13: Para ordenar números não inteiros de 0 a 1 com três casas decimais o primeiro passo é fazer a multiplicação dos números da entrada por 1000. Dessa forma, teremos números inteiros de de 0 a 1000. O passo seguinte é aplicar o algoritmo counting sort sobre tais números fazendo $k = 1000$ (ou então pegar k igual ao maior número nesse novo intervalo) e por fim basta percorrer o vetor ordenado e dividir os números por 1000.

Resposta Pergunta 14:

```c
#include <stdio.h>

void counting_sort(int A[], int n, int k){
 int C[k];
 for (int i = 0; i < k; i++){
  C[i] = 0;
 }
 for (int j = 0; j < n; j++){
  C[A[j]] = C[A[j]] + 1;
 }
 for (int i = 1; i < k; i++){
  C[i] = C[i] + C[i-1];
 }
 int B[n];
 for (int j = n-1; j >= 0; j--){
  B[C[A[j]]-1] = A[j];
  C[A[j]] = C[A[j]] - 1;
 }
 for (int j = 0; j < n; j++){
  A[j] = B[j];
 }
}

void print_vector_int(int vetor[], int n){
 for (int i = 0; i < n; i++){
  printf("%d ", vetor[i]);
 }
 printf("\n");
}

void print_vector_float(float vetor[], int n){
 for (int i = 0; i < n; i++){
  printf("%.3f ", vetor[i]);
 }
 printf("\n");
}

void vector_int_to_float(int vetInt[], float vetFloat[], int n){
 for (int i = 0; i < n; i++){
  vetInt[i] = 1000*vetFloat[i];
 }
```

```
}

void vector_float_to_int(float vetFloat[], int vetInt[], int n){
 for (int i = 0; i < n; i++){
  vetFloat[i] = vetInt[i]/1000.0;
 }
}

int main(){
 float vetorFloat[] = {0.253, 0.521, 0.378, 0.216, 0.278, 0.324, 0.013,
   0.399};
 int n = 8;
 int k = 1000;
 int vetorInt[n];
 vector_int_to_float(vetorInt, vetorFloat, n);
 printf("====counting_sort====\n");
 printf("vetor desordenado\n");
 print_vector_float(vetorFloat, n);
 print_vector_int(vetorInt, n);
 counting_sort(vetorInt, n, k);
 printf("vetor ordenado\n");
 print_vector_int(vetorInt, n);
 vector_float_to_int(vetorFloat, vetorInt, n);
 print_vector_float(vetorFloat, n);
 return 0;
}
```

Listing 4: Resposta do exercício 14 codificado na linguagem C

Resposta Pergunta 15: O algoritmo bucket sort pode ser aplicado sobre valores aleatórios, mas que seguem uma distribuição uniforme. O ideal é que os valores estejam no intervalo de 0 a 1, porém transformações podem ser feitas sobre os dados de forma a aplicar o algoritmo.

Resposta Pergunta 16: Para ordenar números inteiros no intervalo de 0 a 1000 basta fazer uma transformação dos números inteiros de 0 a 1000 em números reais de 0 a 1. Isso pode ser feito dividindo os números por 1000. Então aplica-se o método bucket sort e por fim nos dados ordenados volta-se os valores para o intervalo apropriado multiplicando por 1000.

Resposta Pergunta 19: Algoritmo utiliza basicamente as linhas do counting sort que constrói o vetor C. Para contar o número de inteiros em [a..b], fazemos C[b] - C[a-1] com C[-1] = 0.

Resposta Pergunta 21: Algoritmos estáveis: Insertion sort, merge sort. Algoritmos não estáveis: Heapsort, quicksort. Podemos tornar qualquer algoritmo estável através do mapeamento do vetor em um vetor de pares, onde o primeiro elemento em cada par é o elemento original e o segundo elemento é o seu índice. Assim, podemos ordenar lexicograficamente. Esse esquema exige um espaço adicional na ordem $\Theta(n)$.

Resposta Pergunta 22: Hipótese de indução (loop invariante): "No início do laço for, o

vetor é ordenado nos último i-1 dígitos." Inicialização (k=0): O vetor é trivialmente ordenado no último dígito 0. Manutenção (k=i): Vamos assumir que o vetor está ordenado nos últimos i-1 dígitos. Após ordenarmos no i-ésimo dígito, o vetor estará ordenado nos últimos i dígitos. Elementos com dígitos diferentes na i-ésima posição são ordenados apropriadamente. No caso de mesmo dígitos, a ordem correta é mantida já que um algoritmo de ordenação estável é aplicado e os elementos já estão ordenados nos i-1 dígitos. Terminação (k=n): O laço termina quando i=d+1. Desde que o laço invariante ocorre, teremos ordenados os d dígitos.

Resposta Pergunta 23: Usaremos radix sort. Se tivermos um número de 2 dígitos na base n, radix-sort levará $\Theta(2(n+n)) = \Theta(4n) = \Theta(n)$.

Resposta Pergunta 24: Pior caso: todas as chaves estão no mesmo subintervalo (lista) e aparecem em ordem inversa. Precisamos ordenar uma única lista com n elementos em ordem reversa usando insertion sort. Logo, levaremos $\Theta(n^2)$. Podemos utilizar merge ou heap sort para melhorar a pior caso. Insertion sort funciona bem em listas ligadas. Se utilizamos outros algoritmos de ordenação, precisaremos converter cada lista para vetores o que pode levar a perda de performance na prática.