Ciência da Computação

Aula 6
Métodos de Ordenação

André Luiz Brun

unioeste
Universidade Estadual do Oeste do Paraná

```
void BubbleSort (int *Vet, int n)
{
1     int i, j, aux;
2     for (i=0;i<n;i++)
3   {
4         for (j=0;j<n-1;j++)
5       {
6           if (Vet[j] > Vet[j+1])
7           {
8                 aux = Vet[j];
9                 Vet[j] = Vet[j+1];
10                Vet[j+1] = aux;
11          }
12        }
13  }
}
```

```
void BubbleSort (int *Vet, int n)
{
1     int i, j, aux;
2     for (i=0;i<n;i++)
3   {
4         for (j=0;j<n-1;j++)
5       {
6           if (Vet[j] > Vet[j+1])
7           {
8                 aux = Vet[j];
9                 Vet[j] = Vet[j+1];
10                Vet[j+1] = aux;
11          }
12        }
13  }
}
```

```
void BubbleSort (int *Vet, int n)
{
1     int i, j, aux;
2     for (i=0;i<n;i++)
3   {
4         for (j=0;j<n-1;j++)
5       {
6           if (Vet[j] > Vet[j+1])
7           {
8                 aux = Vet[j];
9                 Vet[j] = Vet[j+1];
10                Vet[j+1] = aux;
11          }
12        }
13  }
}
```

unioeste
Universidade Estadual do Oeste do Paraná

```
void InsertionSort(int *Vet, int n)
{
1    int i, aux, j;
2    for (i=1;i<n;i++)
3    {
4        aux = Vet[i];
5        j = i-1;
6        while (j>=0 && Vet[j]>aux)
7        {
8            Vet[j+1] = Vet[j];
9            j = j-1;
10       }
11       Vet[j+1] = aux;
12   }
}
```

```
void InsertionSort(int *Vet, int n)
{
1    int i, aux, j;
2    for (i=1;i<n;i++)
3    {
4        aux = Vet[i];
5        j = i-1;
6        while (j>=0 && Vet[j]>aux)
7        {
8            Vet[j+1] = Vet[j];
9            j = j-1;
10       }
11       Vet[j+1] = aux;
12   }
}
```

```
void InsertionSort(int *Vet, int n)
{
1    int i, aux, j;
2    for (i=1;i<n;i++)
3    {
4        aux = Vet[i];
5        j = i-1;
6        while (j>=0 && Vet[j]>aux)
7        {
8            Vet[j+1] = Vet[j];
9            j = j-1;
10       }
11       Vet[j+1] = aux;
12   }
}
```

```
void SelectionSort(int *Vet, int n)
{
1   int i, j, min, aux;
2   for (i=0;i<n-1;i++)
3   {
4       min = i;
5       for (j=i+1;j<n;j++)
6           if (Vet[j] < Vet[min])
8               min = j;
9       if (Vet[i] != Vet[min])
10      {
11          aux = Vet[i];
12          Vet[i] = Vet[min];
13          Vet[min] = aux;
14      }
15  }
}
```

```
void SelectionSort(int *Vet, int n)
{
1   int i, j, min, aux;
2   for (i=0;i<n-1;i++)
3   {
4       min = i;
5       for (j=i+1;j<n;j++)
6           if (Vet[j] < Vet[min])
8               min = j;
9       if (Vet[i] != Vet[min])
10      {
11          aux = Vet[i];
12          Vet[i] = Vet[min];
13          Vet[min] = aux;
14      }
15  }
}
```

```
void SelectionSort(int *Vet, int n)
{
1   int i, j, min, aux;
2   for (i=0;i<n-1;i++)
3   {
4       min = i;
5       for (j=i+1;j<n;j++)
6           if (Vet[j] < Vet[min])
8               min = j;
9       if (Vet[i] != Vet[min])
10      {
11          aux = Vet[i];
12          Vet[i] = Vet[min];
13          Vet[min] = aux;
14      }
15  }
}
```

```
void MergeSort(int *Vet, int comeco, int fim)
{
1    if (comeco < fim)
2    {
3        int meio = (fim+comeco)/2;
4        MergeSort(Vet, comeco, meio);
5        MergeSort(Vet, meio+1, fim);
6        merge(Vet, comeco, meio, fim);
7    }
}
```

```
void merge(int *Vet, int comeco, int meio, int fim) {
1     int com1 = comeco, com2 = meio+1;
2     int comAux = 0, tam = fim-comeco+1;
3     int *vetAux;
4     vetAux = (int*)malloc(tam * sizeof(int));
5     while(com1 <= meio && com2 <= fim)
6     {
7       if(Vet[com1] < Vet[com2])
8       {
9           vetAux[comAux] = Vet[com1];
10          com1++;
11       }
12      else
13      {
14           vetAux[comAux] = Vet[com2];
15           com2++;
16       }
17       comAux++;
18  }
```

```
 //Caso ainda haja elementos na primeira metade
19  while(com1 <= meio)
20  {
21      vetAux[comAux] = Vet[com1];
22      comAux++;
23      com1++;
24  }
    //Caso ainda haja elementos na segunda metade
25  while(com2 <= fim)
26  {
27      vetAux[comAux] = Vet[com2];
28      comAux++;
29      com2++;
30 }
//Move os elementos de volta para o vetor original
31  for(comAux = comeco; comAux <= fim; comAux++)
32     Vet[comAux] = vetAux[comAux-comeco];
34  free(vetAux);
}
```

```
void QuickSort(int *Vet, int left, int right) {
1     int i, j, pivo, y;
2     i = left; j = right;
4     pivo = Vet[(left + right) / 2];
5     while (i <= j)
6     {
7        while (Vet[i] < pivo && i < right)
8             i++;
9        while (Vet[j] > pivo && j > left)
10            j--;
11       if (i <= j)
12       {
13           y = Vet[i];
14           Vet[i] = Vet[j];
15           Vet[j] = y;
16           i++;
17           j--;
18       }
19  }
```

```
20  if (j > left)
21  {
22      QuickSort(Vet, left, j);
23  }
24  if (i < right)
25  {
26      QuickSort(Vet, i, right);
27  }
}
```

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

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

BucketSort


11 9 21 8 17 19 13 1 24 12
Scatter
1
9 8
11 13 12
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Sort
1
8 9
11 12 13
17 19
21 24
0-5
5-10
10-15
15-20
20-25
Gather
1 8 9 11 12 13 17 19 21 24