# BC1424
# Algoritmos e Estruturas de Dados I

## Aula 15:

## Comparação empírica de métodos de ordenação baseada em comparações

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

1Q-2015

# Ordenação

**Algoritmos tratados em aula**

- Selection sort
- Insertion sort
- Bubble sort

- Merge sort
- Heap sort
- Quick sort

Complexidade computacional $\Omega(n \log(n))$

# Algoritmos (review)

```c
void BubbleSort (int v[], int n) {
    int i, j, aux;

    for (i=n-1; i>=1; i--) {
        for (j=0; j<i; j++) {
            if (v[j]>v[j+1]) {
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
        }
    }
}
```

```c
void SelectionSort (int v[], int n) {
    int i, j, iMin, aux;

    for (i=0; i<n-1; i++) {
        iMin = i;

        for (j=i+1; j<n; j++) {
            if (v[iMin]>v[j])
                iMin = j;
        }

        if (iMin!=i) {
            aux = v[iMin];
            v[iMin] = v[i];
            v[i] = aux;
        }
    }
}
```

```c
void InsertionSort (int v[], int n) {
    int i, j, aux;

    for (i=1; i<n; i++) {
        aux = v[i];
        for (j=i-1; j>=0 && v[j]>aux ; j--)
                v[j+1] = v[j];
        v[j+1] = aux;
    }
}
```

```c
void Intercala(int p, int q, int r, int v[]) {
    int i, j, k;
    int *w = (int *)malloc( (r-p+1)*sizeof(int) );
    i = p;
    j = q;
    k = 0;
    while (i<q && j<r) {
        if (v[i]<v[j])
            w[k++] = v[i++];
        else
            w[k++] = v[j++];
    }
    while(i<q)
        w[k++] = v[i++];
    while(j<r)
        w[k++] = v[j++];
    for (i=p; i<r; i++)
        v[i] = w[i-p];
    free(w);
}

void MergeSort (int p, int r, int v[]) {
    if (p<r-1) {
        int q = (p+r)/2;
        MergeSort(p, q, v);
        MergeSort(q, r, v);
        Intercala(p, q, r, v);
    }
}
```

```c
void MaxHeapify (int A[], int m, int i) {
    int e, d, maior, aux;
    e = 2*i;
    d = 2*i+1;
    if (e<=m && A[e]>A[i])
        maior = e;
    else
        maior = i;
    if (d<=m && A[d]>A[maior])
        maior = d;
    if (maior!=i) {
        aux = A[maior];
        A[maior] = A[i];
        A[i] = aux;
        MaxHeapify(A, m, maior);
    }
}

void BuildMaxHeap(int A[], int n) {
    int i;
    for (i=n/2; i>=1; i--)
        MaxHeapify (A, n, i);
}

void HeapSort(int A[], int n) {
    int i, m, aux;
    BuildMaxHeap(A, n);
    m = n;
    for (i=n; i>=2; i--) {
        aux = A[i];
        A[i] = A[1];
        A[1] = aux;
        m = m-1;
        MaxHeapify (A, m, 1);
    }
}
```

```c
int Particione(int A[], int p, int r) {
    int i, j, x, aux;

    x = A[r];
    i = p-1;

    for (j=p; j<=r-1; j++) {
        if (A[j]<=x) {
            i = i+1;
            aux = A[i];
            A[i] = A[j];
            A[j] = aux;
        }
    }

    aux = A[i+1];
    A[i+1] = A[r];
    A[r] = aux;

    return i+1;
}

void QuickSort(int A[], int p, int r) {
    if (p<r) {
        int q = Particione(A, p, r);
        QuickSort(A, p, q-1);
        QuickSort(A, q+1, r);
    }
}
```

```c
int Particione(int A[], int p, int r) {
    int i, j, x, aux;

    srand(time(NULL));
    i = rand()%(r-p)+p;

    aux = A[r];
    A[r] = A[i];
    A[i] = aux;

    x = A[r];
    i = p-1;

    for (j=p; j<=r-1; j++) {
        if (A[j]<=x) {
            i = i+1;
            aux = A[i];
            A[i] = A[j];
            A[j] = aux;
        }
    }

    aux = A[i+1];
    A[i+1] = A[r];
    A[r] = aux;

    return i+1;
}
```

Versão que escolhe o pivô de forma aleatória

# Benchmark

# Vetor aleatório: `gerarVetorAleatorio.c`

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    srand(time(NULL));
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[i] = rand();
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

Na linha de comandos (terminal)

```
gcc gerarVetorAleatorio.c -o gerarVetorAleatorio
./gerarVetorAleatorio 100 > vetorAleatorio100.dat
```

# gerarVetorCrescente.c

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[i] = i;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

# gerarVetorDecrescente.c

```c
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    long int i;
    long int n = atoi(argv[1]);
    long int vetor[n];

    for (i=0; i<n; i++) {
        vetor[n-i-1] = i;
    }

    for (i=0; i<n-1; i++) {
        printf("%ld\n", vetor[i]);
    }
    printf("%ld", vetor[i]);
}
```

# gerarVetorParcialmenteCrescente.c

```
 5 void main(int argc, char *argv[])
 6 {
 7     srand(time(NULL));
 8
 9     long int i, j, aux;
10     long int n = atoi(argv[1]);
11     long int vetor[n];
12
13     // vetor ordenado
14     for (i=0; i<n; i++) {
15         vetor[i] = i;
16     }
17
18     // vetor quase ordenado
19     int k = 50;
20
21     for (i=0; i<=n-k; i++) {
22         j = i+rand()%k;
23         aux = vetor[i];
24         vetor[i] = vetor[j];
25         vetor[j] = aux;
26     }
27
28     for (i=0; i<n-1; i++) {
29         printf("%ld\n", vetor[i]);
30     }
31     printf("%ld", vetor[i]);
32 }
```

# gerarVetorParcialmenteDecrescente.c

```
 5 void main(int argc, char *argv[])
 6 {
 7     srand(time(NULL));
 8
 9     long int i, j, aux;
10     long int n = atoi(argv[1]);
11     long int vetor[n];
12
13     // vetor ordenado
14     for (i=0; i<n; i++) {
15         vetor[n-i-1] = i;
16     }
17
18     // vetor quase ordenado
19     int k = 50;
20
21     for (i=0; i<=n-k; i++) {
22         j = i+rand()%k;
23         aux = vetor[i];
24         vetor[i] = vetor[j];
25         vetor[j] = aux;
26     }
27
28     for (i=0; i<n-1; i++) {
29         printf("%ld\n", vetor[i]);
30     }
31     printf("%ld", vetor[i]);
32 }
```

# Scripts

Todos os arquivos necessários para essa aula de laborório estão disponíveis no Tidia e na seguinte página em:
http://professor.ufabc.edu.br/~jesus.mena/courses/bc1424-1q-2015/

**Procedimento:**

- Baixar do tidia os arquivos de teste: `aed1-15.tar.gz` (repositório)
- Subir o arquivo `aed1-15.tar.gz` a sua área no cloud9
- Descomprimir o arquivo no cloud9
  (no terminal execute: `tar xvf aed1-15.tar.gz` )

- Relembre a estrategia adotada nos algoritmos de ordenação

- Executar o arquivo de geração de vetores para comparação:

```
sh -v benchmark.sh
```

- Executar o arquivo de teste:

```
sh -v benchmark-comparar.sh
```

A execução pode demorar alguns minutos

**Gerando o benchmark**
No terminal digite: `sh -v benchmark.sh`

```
# vetores aleatorios
gcc gerarVetorAleatorio.c -o gerarVetorAleatorio
./gerarVetorAleatorio 100 > vetorAleatorio100.dat
./gerarVetorAleatorio 1000 > vetorAleatorio1000.dat
./gerarVetorAleatorio 10000 > vetorAleatorio10000.dat
./gerarVetorAleatorio 100000 > vetorAleatorio100000.dat
./gerarVetorAleatorio 1000000 > vetorAleatorio1000000.dat

# vetores crescente
gcc gerarVetorCrescente.c -o gerarVetorCrescente
./gerarVetorCrescente 100 > vetorCrescente100.dat
./gerarVetorCrescente 1000 > vetorCrescente1000.dat
./gerarVetorCrescente 10000 > vetorCrescente10000.dat
./gerarVetorCrescente 100000 > vetorCrescente100000.dat
./gerarVetorCrescente 1000000 > vetorCrescente1000000.dat

# vetores decrescente
gcc gerarVetorDecrescente.c -o gerarVetorDecrescente
./gerarVetorDecrescente 100 > vetorDecrescente100.dat
./gerarVetorDecrescente 1000 > vetorDecrescente1000.dat
./gerarVetorDecrescente 10000 > vetorDecrescente10000.dat
./gerarVetorDecrescente 100000 > vetorDecrescente100000.dat
./gerarVetorDecrescente 1000000 > vetorDecrescente1000000.dat

# vetores parcialmente ordenados de forma crescente
gcc gerarVetorParcialmenteOrdenadoCrescente.c -o gerarVetorParcialmenteOrdenadoCrescente
./gerarVetorParcialmenteOrdenadoCrescente 100 > vetorPCrescente100.dat
./gerarVetorParcialmenteOrdenadoCrescente 1000 > vetorPCrescente1000.dat
./gerarVetorParcialmenteOrdenadoCrescente 10000 > vetorPCrescente10000.dat
./gerarVetorParcialmenteOrdenadoCrescente 100000 > vetorPCrescente100000.dat
./gerarVetorParcialmenteOrdenadoCrescente 1000000 > vetorPCrescente1000000.dat

# vetores parcialmente ordenados de forma decrescente
gcc gerarVetorParcialmenteOrdenadoDecrescente.c -o gerarVetorParcialmenteOrdenadoDecrescente
./gerarVetorParcialmenteOrdenadoDecrescente 100 > vetorPDecrescente100.dat
./gerarVetorParcialmenteOrdenadoDecrescente 1000 > vetorPDecrescente1000.dat
./gerarVetorParcialmenteOrdenadoDecrescente 10000 > vetorPDecrescente10000.dat
./gerarVetorParcialmenteOrdenadoDecrescente 100000 > vetorPDecrescente100000.dat
./gerarVetorParcialmenteOrdenadoDecrescente 1000000 > vetorPDecrescente1000000.dat
```

O caractere '>' indica redirecionamento. O vetor com números será salvo em formato txt e terá o nome vetorAleatorio1000000.dat (pode ter qualquer extensão)

**Testando os algoritmos com o bechmark**
No terminal digite: `sh -v benchmar-comparar.sh`

```
# BUBBLE SORT
gcc bubbleSort.c -o bubbleSort
./bubbleSort 100 < vetorAleatorio100.dat > bubbleSort-vetorAleatorio.txt
./bubbleSort 1000 < vetorAleatorio1000.dat >> bubbleSort-vetorAleatorio.txt
./bubbleSort 10000 < vetorAleatorio10000.dat >> bubbleSort-vetorAleatorio.txt
./bubbleSort 100000 < vetorAleatorio100000.dat >> bubbleSort-vetorAleatorio.txt
#./bubbleSort 1000000 < vetorAleatorio1000000.dat >> bubbleSort-vetorAleatorio.txt

./bubbleSort 100 < vetorCrescente100.dat > bubbleSort-vetorCrescente.txt
./bubbleSort 1000 < vetorCrescente1000.dat >> bubbleSort-vetorCrescente.txt
./bubbleSort 10000 < vetorCrescente10000.dat >> bubbleSort-vetorCrescente.txt
./bubbleSort 100000 < vetorCrescente100000.dat >> bubbleSort-vetorCrescente.txt
#./bubbleSort 1000000 < vetorCrescente1000000.dat >> bubbleSort-vetorCrescente.txt

./bubbleSort 100 < vetorDecrescente100.dat > bubbleSort-vetorDecrescente.txt
./bubbleSort 1000 < vetorDecrescente1000.dat >> bubbleSort-vetorDecrescente.txt
./bubbleSort 10000 < vetorDecrescente10000.dat >> bubbleSort-vetorDecrescente.txt
./bubbleSort 100000 < vetorDecrescente100000.dat >> bubbleSort-vetorDecrescente.txt
#./bubbleSort 1000000 < vetorDecrescente1000000.dat >> bubbleSort-vetorDecrescente.txt

./bubbleSort 100 < vetorPCrescente100.dat > bubbleSort-vetorPCrescente.txt
./bubbleSort 1000 < vetorPCrescente1000.dat >> bubbleSort-vetorPCrescente.txt
./bubbleSort 10000 < vetorPCrescente10000.dat >> bubbleSort-vetorPCrescente.txt
./bubbleSort 100000 < vetorPCrescente100000.dat >> bubbleSort-vetorPCrescente.txt
#./bubbleSort 1000000 < vetorPCrescente1000000.dat >> bubbleSort-vetorPCrescente.txt

./bubbleSort 100 < vetorPDecrescente100.dat > bubbleSort-vetorPDecrescente.txt
./bubbleSort 1000 < vetorPDecrescente1000.dat >> bubbleSort-vetorPDecrescente.txt
./bubbleSort 10000 < vetorPDecrescente10000.dat >> bubbleSort-vetorPDecrescente.txt
./bubbleSort 100000 < vetorPDecrescente100000.dat >> bubbleSort-vetorPDecrescente.txt
#./bubbleSort 1000000 < vetorPDecrescente1000000.dat >> bubbleSort-vetorPDecrescente.txt
```

O caractere '>>' permite adicionar (append) os tempos de execução no arquivo txt.

Veja linha a linha todo o arquivo .sh

```
# QUICK SORT (versao randomizada)
gcc quickSort2.c -o quickSort2
./quickSort2 100 < vetorAleatorio100.dat > quickSort2-vetorAleatorio.txt
./quickSort2 1000 < vetorAleatorio1000.dat >> quickSort2-vetorAleatorio.txt
./quickSort2 10000 < vetorAleatorio10000.dat >> quickSort2-vetorAleatorio.txt
./quickSort2 100000 < vetorAleatorio100000.dat >> quickSort2-vetorAleatorio.txt
./quickSort2 1000000 < vetorAleatorio1000000.dat >> quickSort2-vetorAleatorio.txt

./quickSort2 100 < vetorCrescente100.dat > quickSort2-vetorCrescente.txt
./quickSort2 1000 < vetorCrescente1000.dat >> quickSort2-vetorCrescente.txt
./quickSort2 10000 < vetorCrescente10000.dat >> quickSort2-vetorCrescente.txt
./quickSort2 100000 < vetorCrescente100000.dat >> quickSort2-vetorCrescente.txt
./quickSort2 1000000 < vetorCrescente1000000.dat >> quickSort2-vetorCrescente.txt

./quickSort2 100 < vetorDecrescente100.dat > quickSort2-vetorDecrescente.txt
./quickSort2 1000 < vetorDecrescente1000.dat >> quickSort2-vetorDecrescente.txt
./quickSort2 10000 < vetorDecrescente10000.dat >> quickSort2-vetorDecrescente.txt
./quickSort2 100000 < vetorDecrescente100000.dat >> quickSort2-vetorDecrescente.txt
./quickSort2 1000000 < vetorDecrescente1000000.dat >> quickSort2-vetorDecrescente.txt

./quickSort2 100 < vetorPCrescente100.dat > quickSort2-vetorPCrescente.txt
./quickSort2 1000 < vetorPCrescente1000.dat >> quickSort2-vetorPCrescente.txt
./quickSort2 10000 < vetorPCrescente10000.dat >> quickSort2-vetorPCrescente.txt
./quickSort2 100000 < vetorPCrescente100000.dat >> quickSort2-vetorPCrescente.txt
./quickSort2 1000000 < vetorPCrescente1000000.dat >> quickSort2-vetorPCrescente.txt

./quickSort2 100 < vetorPDecrescente100.dat > quickSort2-vetorPDecrescente.txt
./quickSort2 1000 < vetorPDecrescente1000.dat >> quickSort2-vetorPDecrescente.txt
./quickSort2 10000 < vetorPDecrescente10000.dat >> quickSort2-vetorPDecrescente.txt
./quickSort2 100000 < vetorPDecrescente100000.dat >> quickSort2-vetorPDecrescente.txt
./quickSort2 1000000 < vetorPDecrescente1000000.dat >> quickSort2-vetorPDecrescente.txt
```

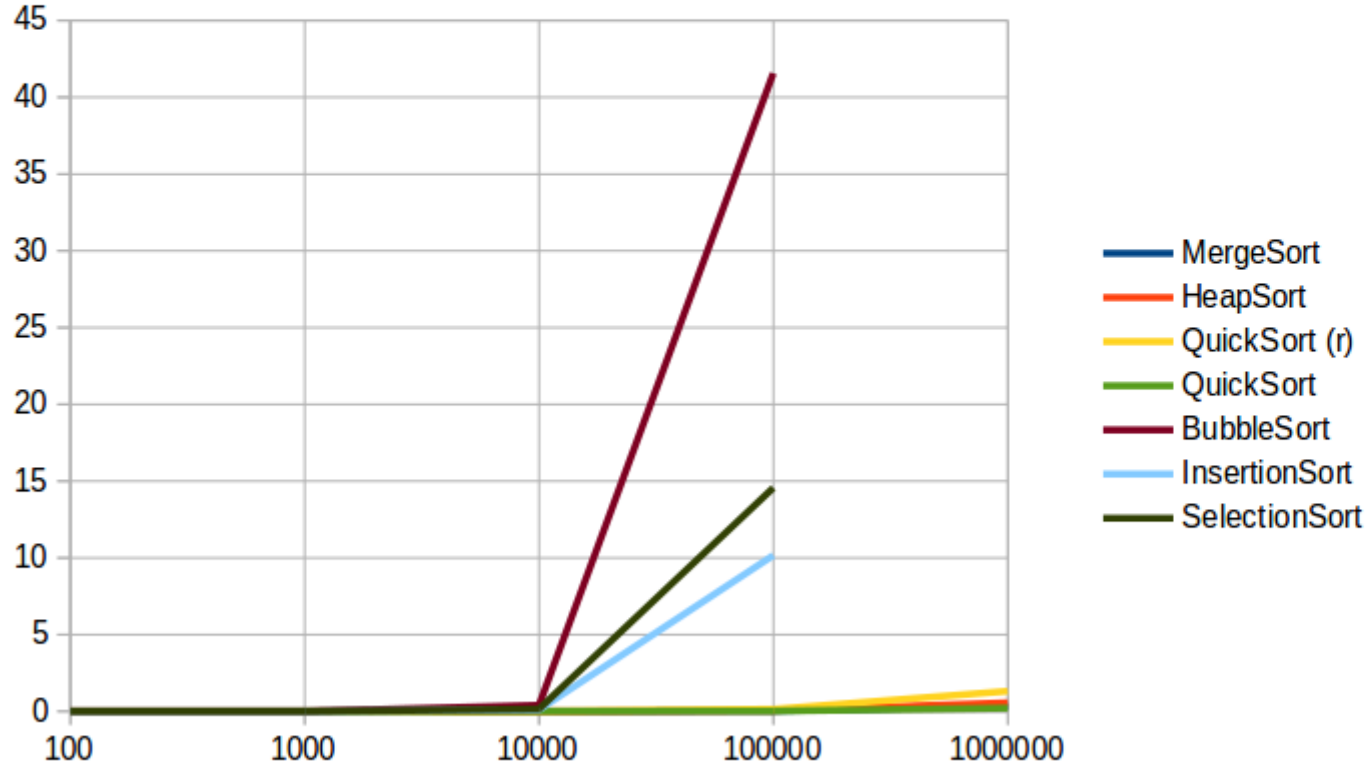
```
# Tabelas
paste mergeSort-vetorAleatorio.txt heapSort-vetorAleatorio.txt quickSort2-vetorAleatorio.txt quickSort1-vetorAleatorio.txt bubbleSort-vetorAleatorio.txt

paste mergeSort-vetorCrescente.txt heapSort-vetorCrescente.txt quickSort2-vetorCrescente.txt quickSort1-vetorCrescente.txt bubbleSort-vetorCrescente.txt
paste mergeSort-vetorDecrescente.txt heapSort-vetorDecrescente.txt quickSort2-vetorDecrescente.txt quickSort1-vetorDecrescente.txt bubbleSort-vetorDecre

paste mergeSort-vetorPCrescente.txt heapSort-vetorPCrescente.txt quickSort2-vetorPCrescente.txt quickSort1-vetorPCrescente.txt bubbleSort-vetorPCrescent
paste mergeSort-vetorPDecrescente.txt heapSort-vetorPDecrescente.txt quickSort2-vetorPDecrescente.txt quickSort1-vetorPDecrescente.txt bubbleSort-vetorP
```

# Alguns resultados (empíricos)

```
jmenac@aed1:~/workspace/aed1-15 $ paste mergeSort-vetorAleatorio.txt heapSort-vetorAleatorio.txt quickSort2-veto
onSort-vetorAleatorio.txt selectionSort-vetorAleatorio.txt
0.000054        0.000016        0.000096        0.000009        0.000041        0.000011        0.000019
0.000210        0.000183        0.001218        0.000100        0.003771        0.000919        0.001611
0.002980        0.002519        0.011807        0.001319        0.402068        0.098685        0.150272
0.026580        0.041515        0.142947        0.021974        41.568623       10.166478       14.558414
0.331140        0.564232        1.315364        0.209602
jmenac@aed1:~/workspace/aed1-15 $ paste mergeSort-vetorCrescente.txt heapSort-vetorCrescente.txt quickSort2-veto
onSort-vetorCrescente.txt selectionSort-vetorCrescente.txt
0.000056        0.000017        0.000072        0.000026        0.000017        0.000001        0.000019
0.000168        0.000200        0.001063        0.002532        0.001592        0.000005        0.001754
0.001385        0.002453        0.008418        0.269848        0.167226        0.000045        0.143163
0.017252        0.027179        0.089772        27.568003       16.613380       0.000702        14.891460
0.200360        0.370314        1.108426
jmenac@aed1:~/workspace/aed1-15 $
jmenac@aed1:~/workspace/aed1-15 $ paste mergeSort-vetorDecrescente.txt heapSort-vetorDecrescente.txt quickSort2-
xt insertionSort-vetorDecrescente.txt selectionSort-vetorDecrescente.txt
0.000058        0.000012        0.000172        0.000031        0.000026        0.000024        0.000021
0.000172        0.000166        0.001021        0.002068        0.002955        0.002046        0.001861
0.001954        0.002694        0.014862        0.233625        0.320085        0.215879        0.157988
0.016143        0.025495        0.130316        20.441996       28.810044       19.649737       15.045489
0.195667        0.341225        1.189645
jmenac@aed1:~/workspace/aed1-15 $ paste mergeSort-vetorPCrescente.txt heapSort-vetorPCrescente.txt quickSort2-ve
sertionSort-vetorPCrescente.txt selectionSort-vetorPCrescente.txt
0.000054        0.000013        0.000123        0.000021        0.000036        0.000007        0.000017
0.000255        0.000222        0.001787        0.000170        0.002026        0.000119        0.001486
0.001970        0.002448        0.011948        0.004596        0.171512        0.001081        0.142058
0.020855        0.030925        0.127878        0.533658        16.602426       0.011091        14.661533
0.243014        0.402200        1.193165
jmenac@aed1:~/workspace/aed1-15 $ paste mergeSort-vetorPDecrescente.txt heapSort-vetorPDecrescente.txt quickSort
nte.txt insertionSort-vetorPDecrescente.txt selectionSort-vetorPDecrescente.txt
0.000066        0.000017        0.000103        0.000008        0.000031        0.000014        0.000020
0.000192        0.000214        0.001044        0.000225        0.002768        0.002735        0.001931
0.002033        0.002231        0.010504        0.005262        0.288105        0.193196        0.188448
0.025322        0.027600        0.117659        0.448359        29.329207       19.539461       20.409276
0.240897        0.379186        1.195488
```

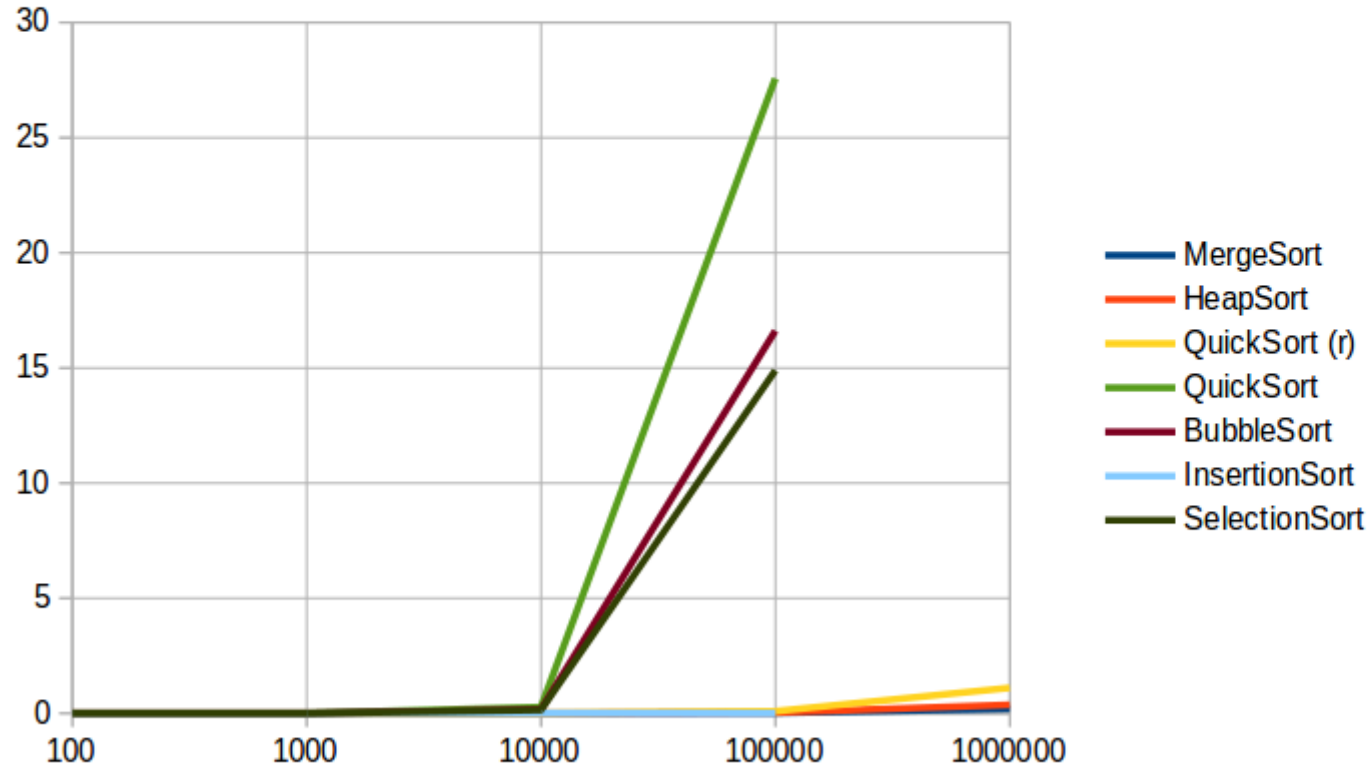
Vetor Aletorio

| n | MergeSort | HeapSort | QuickSort (r) | QuickSort | BubbleSort | InsertionSort | SelectionSort |
|---|---|---|---|---|---|---|---|
| **100** | 0,000054 | 0,000016 | 0,000096 | 0,000009 | 0,000041 | 0,000011 | 0,000019 |
| **1000** | 0,00021 | 0,000183 | 0,001218 | 0,0001 | 0,003771 | 0,000919 | 0,001611 |
| **10000** | 0,00298 | 0,002519 | 0,011807 | 0,001319 | 0,402068 | 0,098685 | 0,150272 |
| **100000** | 0,02658 | 0,041515 | 0,142947 | 0,021974 | 41,568623 | 10,166478 | 14,558414 |
| **1000000** | 0,33114 | 0,564232 | 1,315364 | 0,209602 | - | - | - |



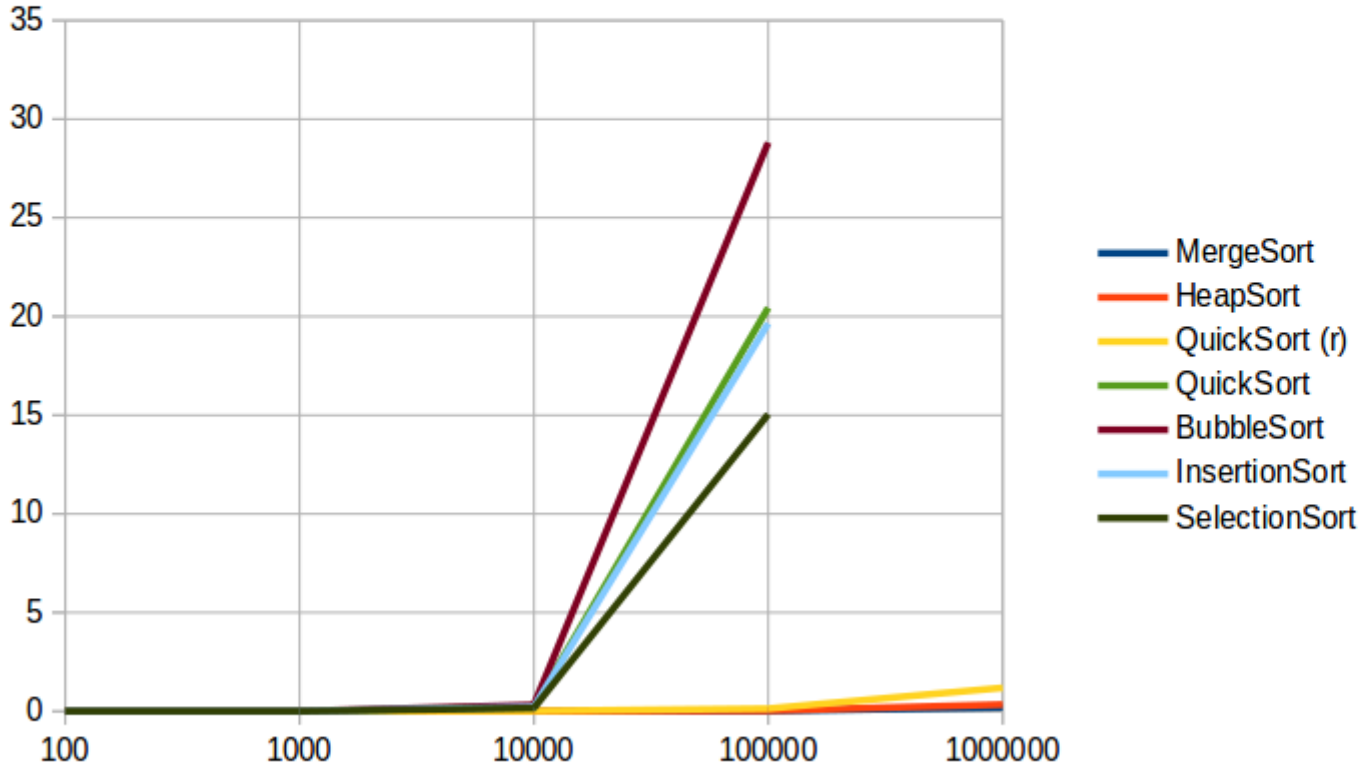
Vetor Crescente

| n | MergeSort | HeapSort | QuickSort (r) | QuickSort | BubbleSort | InsertionSort | SelectionSort |
|---|---|---|---|---|---|---|---|
| **100** | 0,000056 | 0,000017 | 0,000072 | 0,000026 | 0,000017 | 0,000001 | 0,000019 |
| **1000** | 0,000168 | 0,0002 | 0,001063 | 0,002532 | 0,001592 | 0,000005 | 0,001754 |
| **10000** | 0,001385 | 0,002453 | 0,008418 | 0,269848 | 0,167226 | 0,000045 | 0,143163 |
| **100000** | 0,017252 | 0,027179 | 0,089772 | 27,568003 | 16,61338 | 0,000702 | 14,89146 |
| **1000000** | 0,20036 | 0,370314 | 1,108426 | | | | |



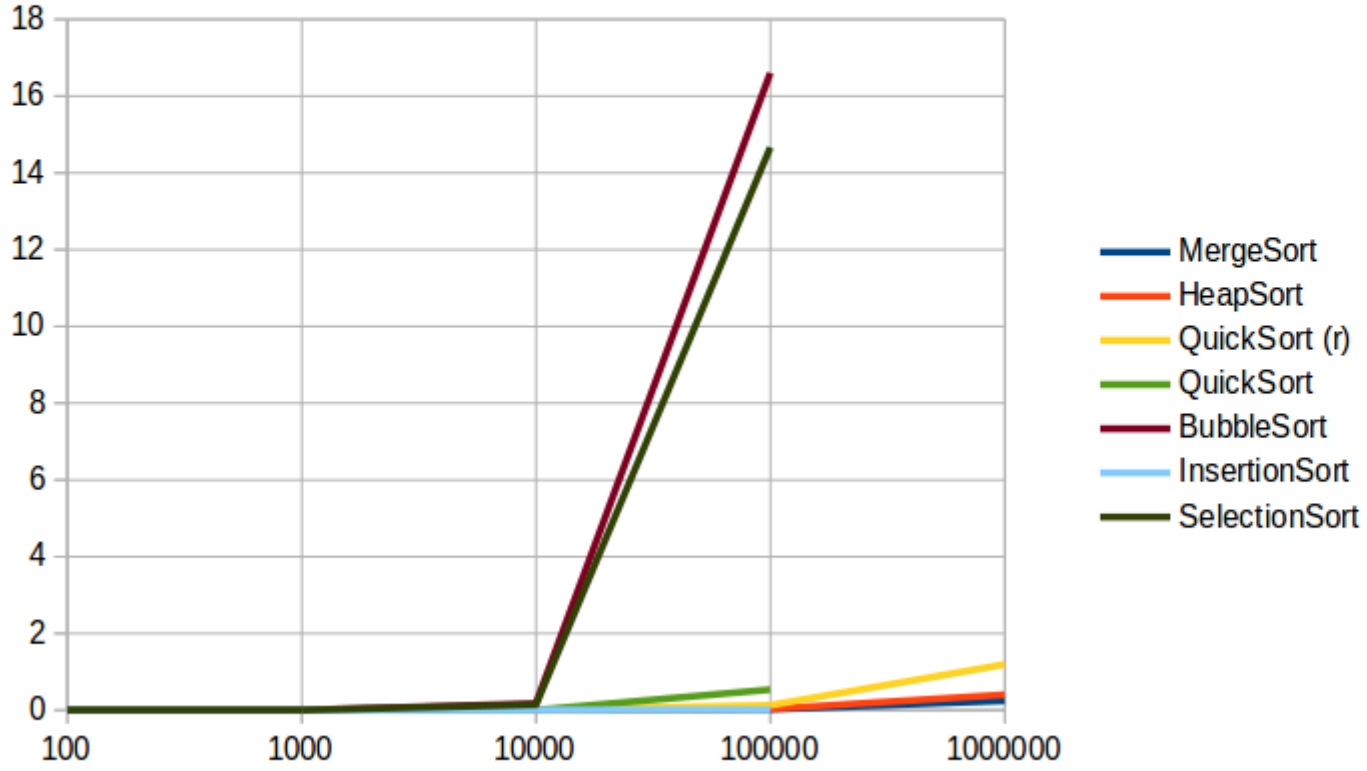
Vetor Decrescente

| n | MergeSort | HeapSort | QuickSort (r) | QuickSort | BubbleSort | InsertionSort | SelectionSort |
|---|---|---|---|---|---|---|---|
| **100** | 0,000058 | 0,000012 | 0,000172 | 0,000031 | 0,000026 | 0,000024 | 0,000021 |
| **1000** | 0,000172 | 0,000166 | 0,001021 | 0,002068 | 0,002955 | 0,002046 | 0,001861 |
| **10000** | 0,001954 | 0,002694 | 0,014862 | 0,233625 | 0,320085 | 0,215879 | 0,157988 |
| **100000** | 0,016143 | 0,025495 | 0,130316 | 20,441996 | 28,810044 | 19,649737 | 15,045489 |
| **1000000** | 0,195667 | 0,341225 | 1,189645 | | | | |



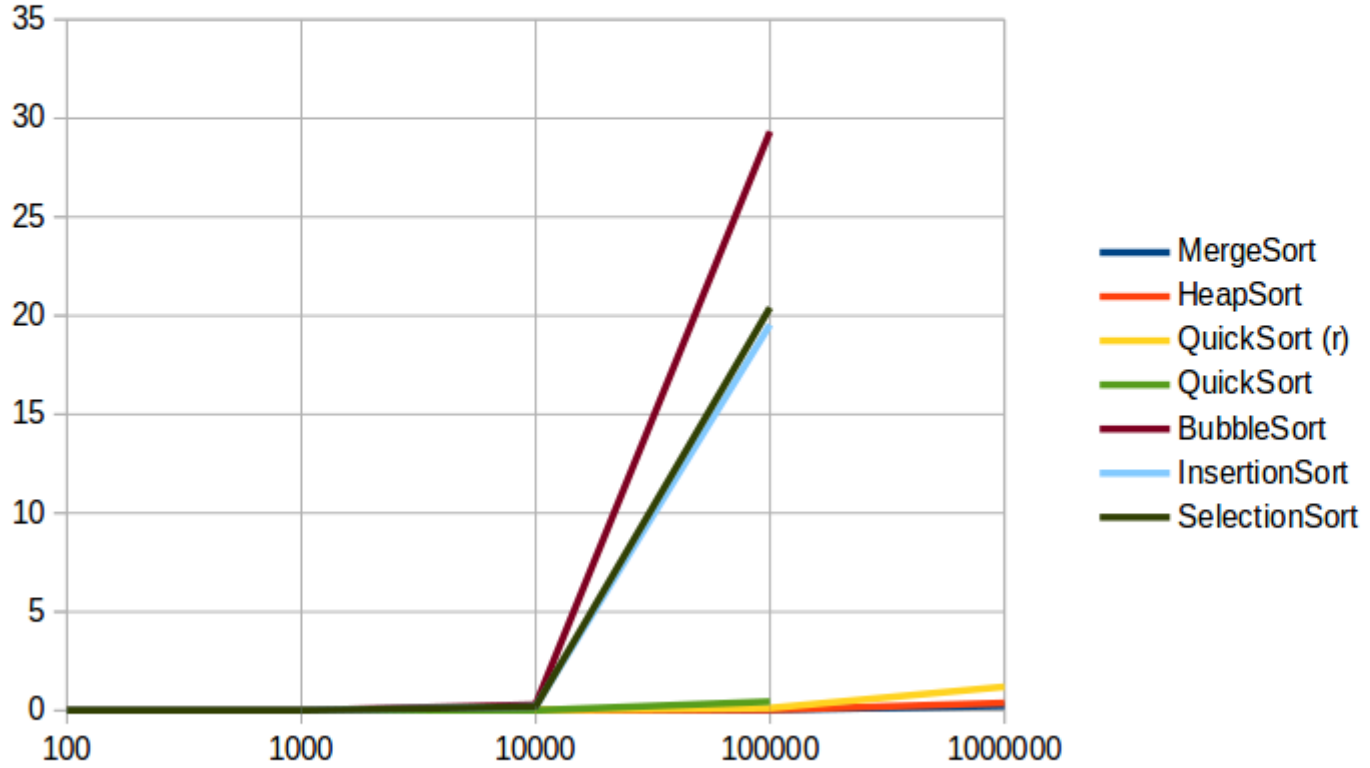
Vetor Parcialmente Crescente

| n | MergeSort | HeapSort | QuickSort (r) | QuickSort | BubbleSort | InsertionSort | SelectionSort |
|---|---|---|---|---|---|---|---|
| **100** | 0,000054 | 0,000013 | 0,000123 | 0,000021 | 0,000036 | 0,000007 | 0,000017 |
| **1000** | 0,000255 | 0,000222 | 0,001787 | 0,00017 | 0,002026 | 0,000119 | 0,001486 |
| **10000** | 0,00197 | 0,002448 | 0,011948 | 0,004596 | 0,171512 | 0,001081 | 0,142058 |
| **100000** | 0,020855 | 0,030925 | 0,127878 | 0,533658 | 16,602426 | 0,011091 | 14,661533 |
| **1000000** | 0,243014 | 0,4022 | 1,193165 | | | | |



Vetor Parcialmente Decrescente

| n | MergeSort | HeapSort | QuickSort (r) | QuickSort | BubbleSort | InsertionSort | SelectionSort |
|---|---|---|---|---|---|---|---|
| **100** | 0,000066 | 0,000017 | 0,000103 | 0,000008 | 0,000031 | 0,000014 | 0,00002 |
| **1000** | 0,000192 | 0,000214 | 0,001044 | 0,000225 | 0,002768 | 0,002735 | 0,001931 |
| **10000** | 0,002033 | 0,002231 | 0,010504 | 0,005262 | 0,288105 | 0,193196 | 0,188448 |
| **100000** | 0,025322 | 0,0276 | 0,117659 | 0,448359 | 29,329207 | 19,539461 | 20,409276 |
| **1000000** | 0,240897 | 0,379186 | 1,195488 | | | | |



# Lista 06 (última lista)

**Sorting**

- **Quicksort 1 - Partition**
- **Quicksort 2 – Sorting**
- **Running Time of Quicksort**

Será utilizado um programa de deteção de plágio em todas as submissões!
Plágio → reprovação

**Data:** **19**/Abril (domingo) até às 23h50.

**Envio:** Através do Tidia.

**Arquivos:**

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA_nomeDoProblema.pdf

Exemplo: 10123456_solveMeFirst**.c**
10123456_solveMeFirst**.pdf**

# Desafio 3: 25/04 (Opcional)

**IDEIA:**
O tempo de execução do QuickSort pode ser melhorado na prática considerando a vantagem do tempo de execução do insertionSort quando a entrada esta quase ordenada (veja resultados empíricos desta aula).

**SOLUÇÃO:**
A) Utilize o QuickSort (versão que escolhe o pivô de forma aletória) para tamanhos de vetor maiores do que `k`. Se o tamanho for menor ou igual a `k`, devolva o vetor sem ordenar. Esse procedimento permitirá ter um vetor quase ordenado.
B) Utilize o InsertionSort para ordenar o vetor inteiro.
O tempo de execução total, desta abordagem, será: $O(nk + n \log(n/k))$

**QUESTÕES A RESPONDER (DE FORMA EMPÍRICA) NO DESAFIO:**
- Na prática essa nova abordagem é melhor do que o QuickSort (versao aleatória)?
- Para que valores de `k` a abordagem é o mais apropriada?
- Como devem ser escolhidos os valores de K?

OBS: Quanto mais evidências (e.g. testes, tabelas, graficos) fundamentarem sua resposta, melhor!
Faça uma apresentação de, no máximo, 6 minutos (sorteio de apresentador).

**SOBRE A AVALIAÇÃO:**
- **Um ponto na media final**.
- Pode ser elaborado por até 3 alunos.
- **É obrigatória a apresentação no último de aula (Sab. 25/abril). Pode preparar slides para a apresentação. Não é necessária a entrega de relatório.**
- No Tidia envie apenas um arquivo .zip contendo o programa e arquivos necessários para replicar os testes.