CENTRO UNIVERSITÁRIO SALESIANO DE SÃO PAULO

ALISSON DE AQUINO SILVA, GIOVANO MARCONDES, JOÃO PEDRO MACIEL OLIVARES, NICOLAS RODRIGUES CAMPOS MARTINS E TALLES AMARAL

TRABALHO PRÁTICO 2

Lorena – São Paulo
2023

SUMÁRIO

MATRIZ DE CONTRIBUIÇÃO

Data	Exercícios	Aluno responsável
05/04/2023	5.2, 6.1, 6.3	Alisson de Aquino e Nicolas Rodrigues
12/04/2023	6.4, 6.5	João Pedro e Giovano Marcondes
25/04/2023	6.6, 7.1, 7.2	Talles Amaral e Nicolas Rodrigues

PLANILHA DE NOTAS

Lista	Exercícios	Nota	Observação
UA05	5.2 (1.0)		
UA06	6.1 (1.5), 6.3 (1.0), 6.4 (1.5), 6.5 (1.0), 6.6 (1.0)		
UA07	7.1 (1.5), 7.2 (1.5)		
UA08	Não considerada para avaliação		
Total	(10.0)		//

Exercício 5.2

Dadas duas versões do Algoritmo de Médias Pré-fixadas, apresentadas no material "**Análise de Agoritmos-4**":

a) Implementar os algoritmos **Medias1** e **Medias2**.
b) Rodar para um array com 1.000 elementos e medir tempo de execução;
c) Rodar novamente fazendo variações do tamanho da entrada (de 100.000 para 200.000 elementos;
d) Calcular os números de operações primitivas desses algoritmos;
e) Fazer as Análises Assintóticas dos Algoritmos (função de complexidade e notação Big Oh); e
f) Qual é o melhor? Argumentar com base na Análise Experimental e na Análise por Método Formal.

a)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define N 200000

void medial(float array[], int cont){
    int i,j;
    float a,A[cont];

    for(i=0;i<cont;i++){
        a=0;

        for(j=0;j<=i;j++){
            a += array[j];
        }

        A[i] = a / (i + 1);
    }

}

void media2(float array[], int cont){
    int i;
    float s=0, A[cont];

    for(i=0;i<cont;i++){
        s += array[i];
        A[i] = s / (i+1);
    }


}
```

```c
int main()
{
    int cont=0,i;
    float array[N];
    clock_t tempo1, tempo2;

    srand(time(NULL));
    for(i=0;i<N;i++){
        array[i] = rand()%N;
        cont++;
    }

    tempo1 = clock();

    printf("\n\nQuantidade de numeros: %d",cont);

    medial(array, cont);
    //media2(array,cont);

    tempo2 = clock();

    printf("\n\nTempo: %f segundos", ((double)(tempo2) - (double)(tempo1)) / CLOCKS_PER_SEC);

    return 0;
}
```

b)

Médias1:

```
Quantidade de numeros: 1000
Tempo: 0.000059 segundos
```

Médias2:

```
Quantidade de numeros: 1000
Tempo: 0.000037 segundos
```

c)

Médias1:

```
Quantidade de numeros: 100000
Tempo: 0.000080 segundos
```

```
Quantidade de numeros: 200000
Tempo: 0.000109 segundos
```

Médias2:

```
Quantidade de numeros: 100000
Tempo: 0.000044 segundos
```

```
Quantidade de numeros: 200000
Tempo: 0.000062 segundos
```

d)

Médias1:

```
42      float array[N]; //1

59      medial(array, cont); //3
```

```
 6  void medial(float array[], int cont){
 7      int i,j;
 8      float a,A[cont]; //1
 9
10      for(i=0;i<cont;i++){ // 1 + n+1
11          a=0; // i++ 3n
12
13          for(j=0;j<=i;j++){ // 1.n + n.(n+1)
14              a += array[j]; // j++ n.(3.n+2.n)
15          }
16
17          A[i] = a / (i + 1); // 4.n
18      }
19
20      /*for(i=0;i<cont;i++){
21          printf("Media[%d]: %.2f\n",i+1,A[i]);
22      }*/
23  }
```

1+3+1+1+n+1+n+n.(n+1)+n.(3.n+2.n)+4.n

7+n+n+n.(n+1)+n.(5n)+4n

$7+7n+n^2+n+5n^2$

$7+8n+6n^2$

Médias2:

```
42      float array[N]; //1

60      media2(array,cont); // 3
```

```
25  void media2(float array[], int cont){
26      int i;
27      float s=0, A[cont]; //2
28
29      for(i=0;i<cont;i++){ // 1 + n+1
30          s += array[i]; // i++ 3.n + 2.n
31          A[i] = s / (i+1); // 4n
32      }
33
34      /*for(i=0;i<cont;i++){
35          printf("Media[%d]: %.2f\n",i+1,A[i]);
36      }*/
37  }
```

1+3+2+1+n+1+3n+2n+4n

8+10n

e) Médias1: $O(n^2)$ Médias2: $O(n)$

f) O algoritmo Médias2 é melhor, por ser um O(n) é mais rápido que o Médias1, um $O(n^2)$

Exercício 6.1

Implementar uma Árvore Binária, em especial, os Métodos:

a) *insertTree* (Inserção de Node em Árvore);
b) *IsExternal* (Verificação se Node é Externo);
c) *Size* (Tamanho da Árvore);
d) *isInTree* (Verificação se Node existe na Árvore);
e) Testar as inserções para os valores (inserir nessa ordem): 12, 15, 10 e 13.

f) **Desafio:** Como você implementaria o método *isInternal* ?

a)

```
28  void insertTree(BTreeNode** t, int num)
29  {
30    if(*t == NULL)
31    {
32      *t = (BTreeNode*)malloc(sizeof(BTreeNode));
33      (*t)->left = NULL;
34      (*t)->right = NULL;
35      (*t)->numero = num;
36      Size++;
37    } else {
38      if(num <= (*t)->numero)
39      {
40        insertTree(&(*t)->left, num);
41      }
42      if(num > (*t)->numero)
43      {
44        insertTree(&(*t)->right, num);
45      }
46    }
47  }
```

b)

```
49  int isExternal(BTreeNode* t, int num) {
50    if(treeIsEmpty(t)) {
51      return 0;
52    }
53    return ((t->numero==num) && (t->left==NULL) && (t->right==NULL)) || isExternal(t->left, num) || isExternal(t->right, num);
54  }
```

c)

```
67  int size()
68  {
69      return Size;
70      //A variavel Size é incrementada sempre que a função insertTree é chamada
71  }
```

d)

```
73  int isInTree(BTreeNode* t, int num) {
74
75    if(treeIsEmpty(t)) {
76      return 0;
77    }
78    return t->numero==num || isInTree(t->left, num) || isInTree(t->right, num);
79  }
```

e)

```
 97  void main() {
 98
 99      BTreeNode* t = createTree();
100
101      insertTree(&t, 12);
102      insertTree(&t, 15);
103      insertTree(&t, 10);
104      insertTree(&t, 13);
105
106      if(treeIsEmpty(t))
107              printf("\n\nArvore vazia!!\n");
108      else    printf("\n\nArvore NAO vazia!!\n");
109
110      printf("\n");
111      printTree(t, t->numero);
112
113      printf("\n\nVerificacao de Existencia de Numero 15:\n");
114      if(isInTree(t, 15)) {
115      printf("\nO numero 15 pertence a arvore!\n");
116      } else {
117       printf("\nO numero 15 NAO pertence a arvore!\n");
118      }
119
120      printf("\n\nTamanho : %02d\n",size());
121
122      if(isExternal(t, 13))
123      printf("\n\nO Numero 13 eh Externo\n");
124      else
125      printf("\n\nO Numero 13 nao eh Externo\n");
126
127      if(isExternal(t, 12))
128      printf("\n\nO Numero 12 eh Externo\n");
129      else
130      printf("\n\nO Numero 12 nao eh Externo\n");
```

```
132         if(isExternal(t, 15))
133         printf("\n\nO Numero 15 eh Externo\n");
134         else
135         printf("\n\nO Numero 15 nao eh Externo\n");
136
137         if(isExternal(t, 10))
138         printf("\n\nO Numero 10 eh Externo\n");
139         else
140         printf("\n\nO Numero 10 nao eh Externo\n");
141
142         if(isInternal(t, 13))
143         printf("\n\nO Numero 13 eh Interno\n");
144         else
145         printf("\n\nO Numero 13 nao eh Interno\n");
146
147         if(isInternal(t, 12))
148         printf("\n\nO Numero 12 eh Interno\n");
149         else
150         printf("\n\nO Numero 12 nao eh Interno\n");
151
152         if(isInternal(t, 15))
153         printf("\n\nO Numero 15 eh Interno\n");
154         else
155         printf("\n\nO Numero 15 nao eh Interno\n");
156
157         if(isInternal(t, 10))
158         printf("\n\nO Numero 10 eh Interno\n");
159         else
160         printf("\n\nO Numero 10 nao eh Interno\n");
161
162         free(t);
163     }
```

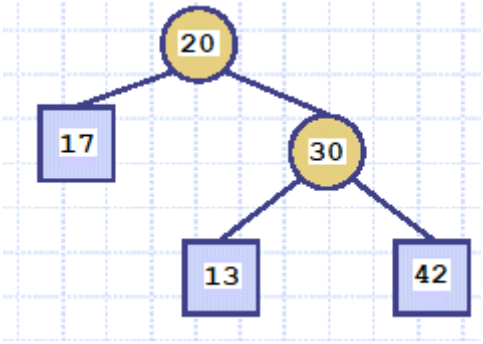
```
Arvore NAO vazia!!

12 10 15 13

Verificacao de Existencia de Numero 15:

O numero 15 pertence a arvore!


Tamanho : 04


O Numero 13 eh Externo


O Numero 12 nao eh Externo


O Numero 15 nao eh Externo


O Numero 10 eh Externo


O Numero 13 nao eh Interno


O Numero 12 eh Interno


O Numero 15 eh Interno


O Numero 10 nao eh Interno

Process returned 1 (0x1)   execution time : 0.038 s
Press any key to continue.
```

f)

```
56  int isInternal(BTreeNode* t, int num) {
57    if(treeIsEmpty(t)) {
58      return 0;
59    }
60
61    if(t->numero==num && t->left!=NULL || isInternal(t->left,num))
62      return 1;
63    else if (t->numero==num && t->right!=NULL || isInternal(t->right,num))
64      return 1;
65  }
```

Exercício 6.3

Observe o Árvore de exemplo dada a seguir:



a) Fazer as inserções, por código, que viabilizem a construção da árvore (nessa geometria);
b) Obter a altura e a profundidade dessa Árvore, com Testes de Mesa para os algoritmos Height e Depth.

a)

```
105     insertTree(&t, 20);
106     insertTree(&t, 17);
107     insertTree(&t, 30);
108     insertTree(&t->right, 13);
109     insertTree(&t->right, 42);
110
```

b)

```
69  int max(int a, int b) {
70      return (a > b) ? a : b;
71  }
72
73  int depth(BTreeNode* t, int num, int level) {
74      if (treeIsEmpty(t)) {
75          return -1;
76      }
77
78      if (t->numero == num) {
79          return level;
80      }
81
82      int leftDepth = depth(t->left, num, level + 1);
83      if (leftDepth != -1) {
84          return leftDepth;
85      }
86
87      int rightDepth = depth(t->right, num, level + 1);
88      return rightDepth;
89  }
90
91  int height(BTreeNode* t) {
92      if (treeIsEmpty(t)) {
93          return -1; //
94      }
95
96      int leftHeight = height(t->left);
97      int rightHeight = height(t->right);
98
99      return 1 + max(leftHeight, rightHeight);
100 }
```

```c
int main() {
    BTreeNode* t = createTree();

    insertTree(&t, 20);
    insertTree(&t, 17);
    insertTree(&t, 30);
    insertTree(&t->right, 13);
    insertTree(&t->right, 42);

    if(treeIsEmpty(t))
        printf("\n\nArvore vazia!!\n");
    else
        printf("\n\nArvore NAO vazia!!\n");

    printf("\n");
    printTree(t, t->numero);

    printf("\n\nVerificacao de Existencia de Numero 20:\n");
    if(isInTree(t, 20)) {
        printf("\nO numero 20 pertence a arvore!\n");
    } else {
        printf("\nO numero 20 NAO pertence a arvore!\n");
    }

    printf("\nProfundidade do numero 42: %d\n", depth(t, 42, 0));
    printf("Altura da arvore: %d\n", height(t));

    free(t);

    return 0;
}
```

```
Arvore NAO vazia!!

( 20 ) | 17 | ( 30 ) | 13 | | 42 |

Verificacao de Existencia de Numero 20:

O numero 20 pertence a arvore!

Profundidade do numero 42: 2
Altura da arvore: 2

Process returned 0 (0x0)   execution time : 0.013 s
Press any key to continue.
```

Exercício 6.4

Implementar para a Árvore Binária, agora os Métodos:

a) *Pre-order* (Caminhamento em Pré-ordem);
b) *Post-order* (Caminhamento em Pós-ordem);
c) *In-order* (Caminhamento em In-ordem);
d) Obter a sequência de nodes visitados da Árvore do **Exercício 6.3**, com Testes de Mesa para os algoritmos Pre-order, Post-Order e In-Order.

a)

```
87  void preOrder(BTreeNode *t)
88  {
89      if (t != NULL)
90      {
91          printf("%d ", t->numero);
92          preOrder(t->left);
93          preOrder(t->right);
94      }
95  }
```

b)

```
107 void postOrder(BTreeNode *t)
108 {
109     if (t != NULL)
110     {
111         postOrder(t->left);
112         postOrder(t->right);
113         printf("%d ", t->numero);
114     }
115 }
```

c)

```c
void inOrder(BTreeNode *t)
{
    if (t != NULL)
    {
        inOrder(t->left);
        printf("%d ", t->numero);
        inOrder(t->right);
    }
}
```

d)

```c
int main()
{
    BTreeNode *t = createTree();

    insertTree(&t, 20);
    insertTree(&t, 17);
    insertTree(&t, 30);
    insertTree(&t->right, 13);
    insertTree(&t->right, 42);

    if (treeIsEmpty(t))
        printf("Arvore vazia!\n");
    else
        printf("Arvore nao vazia!\n");

    printf("\nTravessia pre-order: ");
    preOrder(t);

    printf("\nTravessia in-Order: ");
    inOrder(t);

    printf("\nTravessia post-order: ");
    postOrder(t);

    printf("\n");

    return 0;
}
```

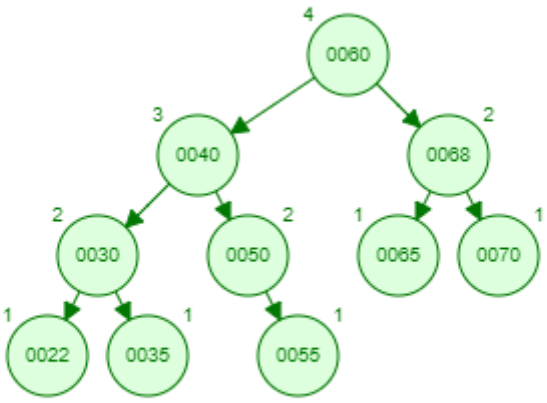
```
Arvore nao vazia!

Travessia pre-order: 20 17 30 13 42
Travessia in-Order: 17 20 13 30 42
Travessia post-order: 17 13 42 30 20

Process returned 0 (0x0)   execution time : 0.056 s
Press any key to continue.
```

Exercício 6.5

Simule visualmente a árvore AVL resultante da inserção dos elementos abaixo, na ordem dada: 50, 30, 22, 60, 70, 65, 68, 40, 55 e 35.

Para esse exercício, sugestão de uso dessa ferramenta:

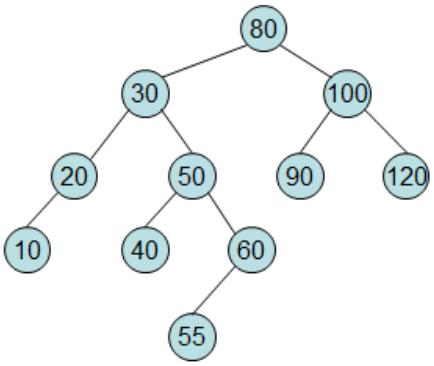
https://people.ok.ubc.ca/ylucet/DS/AVLtree.html

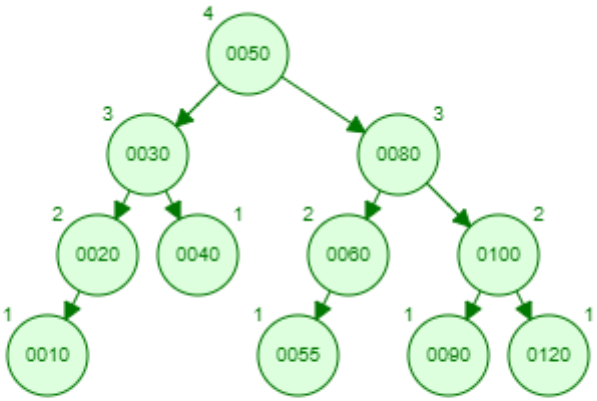


Exercícios 6.6

Abaixo é dada a configuração de uma árvore AVL imediatamente após a inserção do elemento 55 e antes do balanceamento.

Pede-se: identificar e realizar a rotação necessária ao balanceamento da árvore!





O nó desbalanceado era o 80, acontece uma rotação R que ajuda a manter a diferença de altura entre as subárvores em no máximo 1

Exercício 7.1

Com relação aos métodos de ordenação Insertion, Selection Sort, Bubble Sort e Merge Sort, fazer as seguintes implementações e experimentos iniciais:

a) Criar o Programa Ordenacao.c e implementar as seguintes funções: *Insertion Sort, Selection Sort, Bubble Sort e Merge Sort.*

b) Agora testar os métodos de ordenação usando a lista de elementos inteiros dada a seguir: 762, 52, 123, 45, 10, 2, 1544, 982, 565, 140, 265, 396.

c) Apresentar os prints dos testes com o antes e depois da lista, após aplicação de cada algoritmo.

a)

```
20  void insertionSort(int array[])
21  {
22      int i;
23      for (i = 1; i < TAM; i++)
24      {
25       int x = array[i];
26       int j;
27       for (j = i; j > 0 && (x < array[j-1]); j--)
28           { array[j] = array[j-1]; }
29       array[j] = x;
30      }
31  }
32
```

```
33      void selectionSort(int array[])
34  {
35          int j, k, maxIndex;
36          for(k=TAM-1; k>0; k--)
37      {
38              int maxIndex = k;
39              for(j = k-1;j>=0;j--)
40          {
41                  if (array[j] > array[maxIndex])
42                      maxIndex=j;
43          }
44              int temp = array[k];
45              array[k] = array[maxIndex];
46              array[maxIndex] = temp;
47      }
48  }
```

```
 6  void bubbleSort(int array[]){
 7      int i,j, aux;
 8
 9      for(i=0;i<TAM;i++){
10          for(j=0;j<TAM;j++){
11              if(array[i]<array[j]){
12                  aux = array[i];
13                  array[i] = array[j];
14                  array[j] = aux;
15              }
16          }
17      }
18  }
```

```
50  void merge(int arr[], int left[], int leftSize, int right[], int rightSize) {
51      int i = 0, j = 0, k = 0;
52
53      while (i < leftSize && j < rightSize) {
54          if (left[i] <= right[j]) {
55              arr[k] = left[i];
56              i++;
57          } else {
58              arr[k] = right[j];
59              j++;
60          }
61          k++;
62      }
63
64      while (i < leftSize) {
65          arr[k] = left[i];
66          i++;
67          k++;
68      }
69
70      while (j < rightSize) {
71          arr[k] = right[j];
72          j++;
73          k++;
74      }
75  }
```

```
77  void mergeSort(int arr[], int size) {
78      if (size < 2) {
79          return;
80      }
81
82      int mid = size / 2;
83      int left[mid];
84      int right[size - mid];
85
86      for (int i = 0; i < mid; i++) {
87          left[i] = arr[i];
88      }
89
90      for (int i = mid; i < size; i++) {
91          right[i - mid] = arr[i];
92      }
93
94      mergeSort(left, mid);
95      mergeSort(right, size - mid);
96
97      merge(arr, left, mid, right, size - mid);
98  }
```

b/c)

insertionSort:

```
Lista sem ordenacao:
762 52 123 45 10 2 1544 982 565 140 265

Lista com ordenacao:
2 10 45 52 123 140 265 565 762 982 1544
```

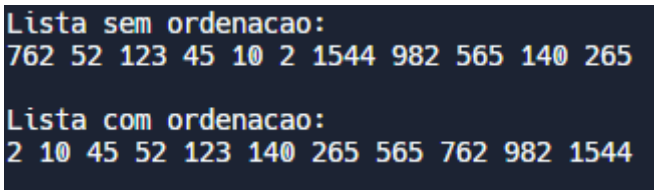
selectionSort:

```
Lista sem ordenacao:
762 52 123 45 10 2 1544 982 565 140 265

Lista com ordenacao:
2 10 45 52 123 140 265 565 762 982 1544
```

bubbleSort:

```
Lista sem ordenacao:
762 52 123 45 10 2 1544 982 565 140 265

Lista com ordenacao:
2 10 45 52 123 140 265 565 762 982 1544
```

mergeSort:


Lista sem ordenacao:
762 52 123 45 10 2 1544 982 565 140 265
Lista com ordenacao:
2 10 45 52 123 140 265 565 762 982 1544

Exercício 7.2

Para os mesmos métodos de ordenação acima, fazer os seguintes experimentos e comparação de Desempenho:

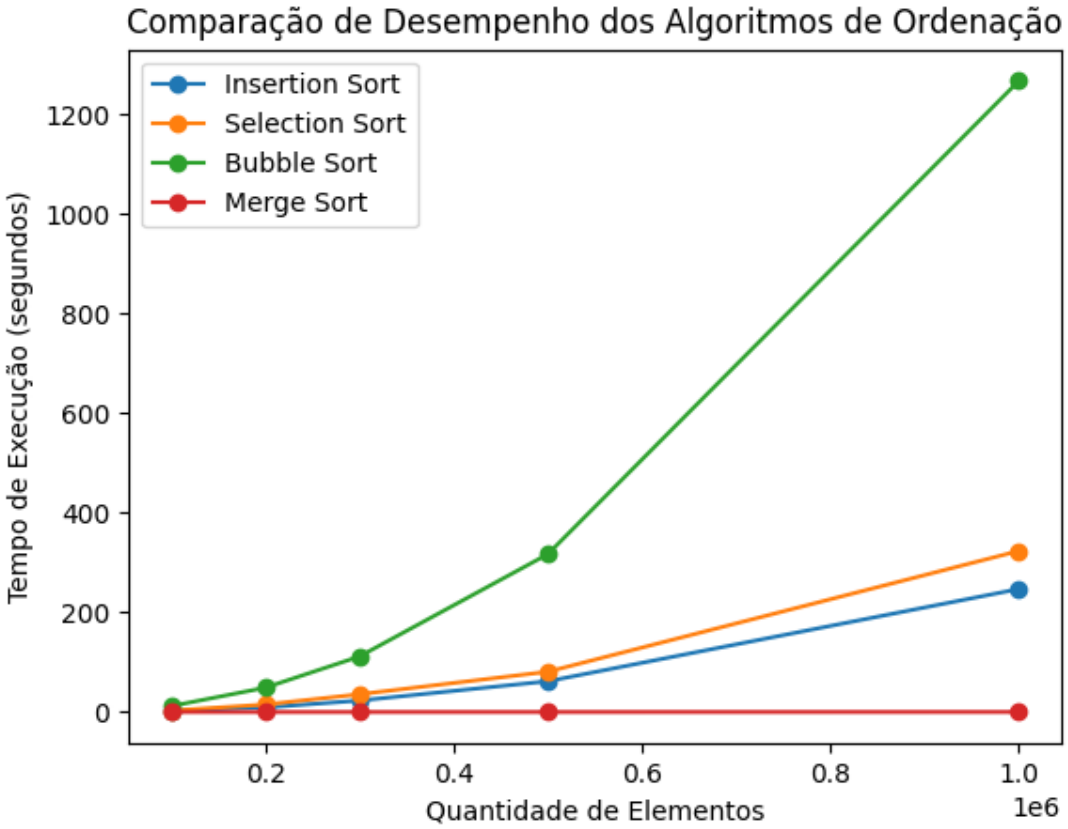
a) No mesmo Programa Ordenacao.c anterior, testar os métodos de ordenação para as seguintes quantidades de elementos aleatoriamente gerados: 100000, 200000, 300000, 500000 e 1000000.

b) Medir o tempo de execução de cada método e criar uma tabela e gráfico comparativo.

c) Procure analisar o desempenho empírico de cada um e concluir o porquê de cada desempenho.

b)

Qtd. Elementos	100000	200000	300000	500000	1000000
insertionSort	2,4671 s	9,0850 s	22,9728 s	61,7800 s	247,12 s
selectionSort	2,8419 s	14,9698 s	35,6379 s	80,9242 s	323,6968 s
bubbleSort	11,7897 s	48,7358 s	111,4506s	317,0743s	1268,2972s
mergeSort	0,0118 s	0,0246 s	0,0367 s	0,06230 s	0,1246 s



c)

Os algoritmos insertionSort, selectionSort e o bubbleSort tendem a ser menos eficientes por possuírem complexidade de tempo quadrática. Já o mergeSort, com complexidade de Tempo O(n log n), oferece um desempenho muito superior.