Gabriel Carvalho Silva 11932438
Paulo Roberto Domingues dos Santos 11838721

Trabalho Final de Introdução à Computação II

Brasil

Dezembro 2020

Gabriel Carvalho Silva 11932438
Paulo Roberto Domingues dos Santos 11838721

Trabalho Final de Introdução à Computação II

Universidade de São Paulo

Professor Renato Tinós

Brasil
Dezembro 2020

Resumo

Este texto compõe o Trabalho Final da disciplina de Introdução à Computação II, ministrada no 2º Semestre de 2020 pelo professor Renato Tinós, na Universidade de São Paulo. O trabalho propõe a implementação de algoritmos aprendidos em aula e análise de seus respectivos comportamentos.

Os vetores utilizados foram gerados utilizando a plataforma https://www.random.org/integer-sets/ e podem ser encontrados no arquivo de texto que acompanha o trabalho.

1 Implementações

A seguir, apresentamos os algoritmos de cada método de ordenação:

1. Inserção direta

```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;

void insDir(int arr[], int n)
{
        int i, j;

        for(i = 2; i <= n; i++)
        {
                mov = mov + 2;

                int x = arr[i];
                arr[0] = x;
                j = i;

                comp = comp + 1;
                while(x < arr[j-1])
                {
                        mov = mov + 1;

                        arr[j] = arr[j-1];
                        j--;

                        comp = comp + 1;
                }

                mov = mov + 1;
                arr[j] = x;

        }

}
int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n + 1];
```

```
        for(int i = 1; i < n + 1; i++)
                std::cin >> array[i];

        insDir(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << " ";

        std::cout << "\nComparacoes: " << comp
            << "\nMovimentacoes: " << mov << std::endl;

        return 0;
}
```

2. Inserção binária

```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;

void swap(int *a, int*b)
{
        int aux;
        aux = *a;
        *a = *b;
        *b = aux;
}

void binInsertionSort(int *array, int tamArray)
{
        int i, j, R, L, x, m;
        for(i = 1; i < tamArray; i++)
        {
                x = array[i];
                mov = mov + 1;
                L = 0;
                R = i;
                while(L < R)
                {
                        m = (L + R) / 2;
                        comp = comp + 1;
                        if(array[m] <= x)
                                L = m + 1;
                        else
                                R = m;
```

```
                }
                j = i;
                while(j > R)
                {
                        array[j] = array[j-1];
                        mov = mov + 1;
                        j--;
                }
                array[R] = x;
                mov = mov + 1;

        }

}

int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n];

        for(int i = 0; i < n; i++)
                std::cin >> array[i];

        binInsertionSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << "␣";

        std::cout << "\nComparacoes:␣" << comp
            << "\nMovimentacoes:␣" << mov << std::endl;

        return 0;
}
```

3. Seleção

```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;

void swap(int *a, int *b)
{
        int aux;
```

```
        aux = *a;
        *a = *b;
        *b = aux;
}

void selSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n-1; i++)
    {
        int menor;

        menor = i;

        for (j = i+1; j < n; j++)
        {
                    comp = comp + 1;

            if (arr[j] < arr[menor])
            {
                menor = j;
            }
        }
        swap(&arr[i], &arr[menor]);

        mov = mov + 3;
    }
}

int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n];

        for(int i = 0; i < n; i++)
                std::cin >> array[i];

        selSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << "␣";

        std::cout << "\nComparacoes:␣" << comp
```

```
            << "\nMovimentacoes: " << mov << std::endl;

        return 0;
}
```

4. BubbleSort

```
#include <stdlib.h>
#include <iostream>

int mov = 0, comp = 0;

void swap(int *a, int*b)
{
        int aux;
        aux = *a;
        *a = *b;
        *b = aux;
}

void bubSort(int arr[], int n)
{
        int i, j;
        for(i = 1; i < n; i++)
        {
                for(j = n-1; j > i-1; j--)
                {

                    if(arr[j-1] > arr[j])
                    {
                                mov = mov + 3;
                        swap(&arr[j-1], &arr[j]);

                        }
                        comp = comp + 1;
                }
        }
}


int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n];
```

```
        for(int i = 0; i < n; i++)
                std::cin >> array[i];

        bubSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << " ";

        std::cout << "\nComparacoes: " << comp
            << "\nMovimentacoes: " << mov << std::endl;

        return 0;
}
```

5. HeapSort

```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;


void swap(int *a, int*b)
{
        int aux;
        aux = *a;
        *a = *b;
        *b = aux;
}

void heapify(int *array, int L, int R)
{
        int i, j, x;

        i = L;
        j = 2*L;
        x = array[L];
        mov++;

        comp++;
        if((j < R) && (array[j] < array[j+1]))
                j++;

        comp++;
        while((j <= R) && (x < array[j]))
        {
```

```
                array[i] = array[j]; mov++;
                i = j;
                j = 2*j;

                comp++;
                if((j < R) && (array[j] < array[j+1]))
                        j++;

        }
        array[i] = x; mov++;

}

void heapSort(int *array, int tamArray)
{
        for(int L = tamArray/2; L > 0; L--)
        {
                heapify(array, L, tamArray);
        }
        for( int R = tamArray; R > 1; R--)
        {
                mov += 3;
                swap(&array[1], &array[R]);

                heapify(array, 1, R-1);

        }
}


int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n+1];

        for(int i = 1; i < n + 1; i++)
                std::cin >> array[i];

        heapSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << "␣";
```

```
        std::cout << "\nComparacoes: " << comp
            << "\nMovimentacoes: " << mov << std::endl;

        return 0;
}
```

6. Fusão

```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;

void merge(int *a, int L, int h, int R, int *c)
{

    int i = L, j = h + 1, k = L - 1;

    while(i <= h && j <= R)
    {
        k++;
        comp++;
        if(a[i] < a[j])
        {
            mov++;
            c[k] = a[i];
            i++;
        } else {
            mov++;
            c[k] = a[j];
            j++;
        }
    }

    while(j <= R)
    {
        k++;
        mov++;
        c[k] = a[j];
        j++;
    }

    while(i <= h)
    {
        k++;
        mov++;
        c[k] = a[i];
```

```
        i++;
    }

}

void mpass(int *a, int n, int p, int *c)
{
    int i = 1;
    while(i <= n-2*p+1)
    {
        merge(a, i, i+p-1,i+2*p-1, c);
        i = i + 2*p;
    }
    if(i+p-1 < n)
        merge(a, i, i+p-1, n, c);
    else
        for(int j = i; j <= n; j++)
        {
            mov++;
            c[j] = a[j];
        }
}

void mergeSort(int *array, int n)
{
    int p = 1;
    int *auxArray = new int[n+1];

    while(p < n)
    {
        mpass(array, n, p, auxArray);
        p = 2*p;
        mpass(auxArray, n, p, array);
        p = 2*p;
    }
}


int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n+1];
```

```
        for(int i = 1; i < n + 1; i++)
                std::cin >> array[i];

        mergeSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << " ";

        std::cout << "\nComparacoes: " << comp
            << "\nMovimentacoes: " << mov << std::endl;

        return 0;
}
```

7. QuickSort

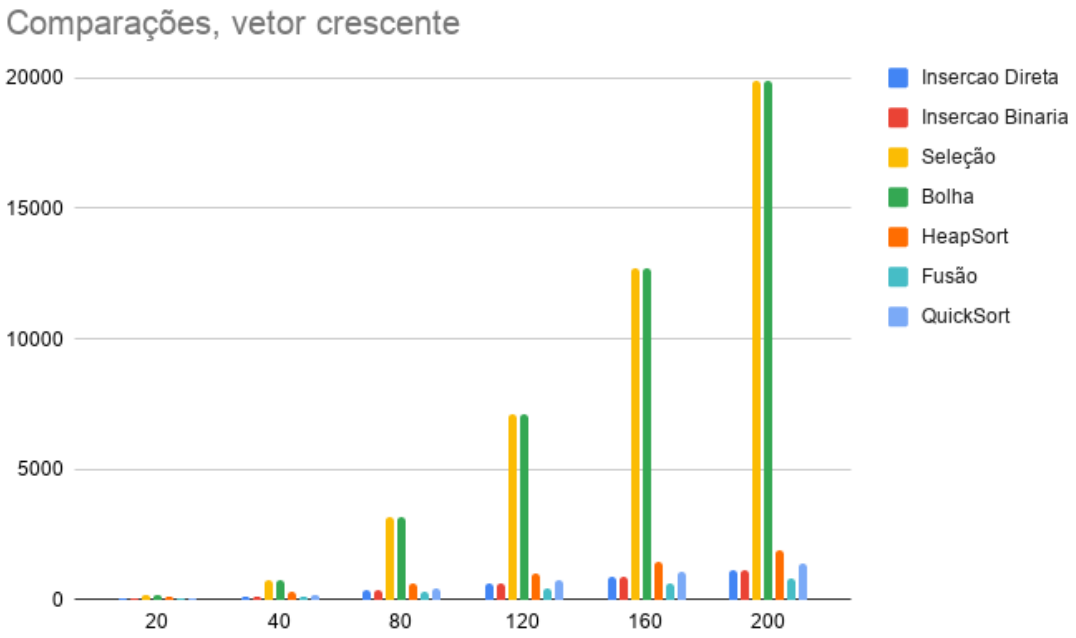
```
#include <iostream>
#include <stdlib.h>

int mov = 0, comp = 0;

void swap(int *a, int*b)
{
        int aux;
        aux = *a;
        *a = *b;
        *b = aux;
}

//caso recorrente
void qSort(int array[], int L, int R)
{
        int i, j, x;

        i = L;
        j = R;
        x = array[(L+R)/2];
        mov = mov + 1;

        do
        {
                comp = comp + 1;
                while(array[i] < x)
                {
                        i++;
                        comp = comp + 1;
                }
```

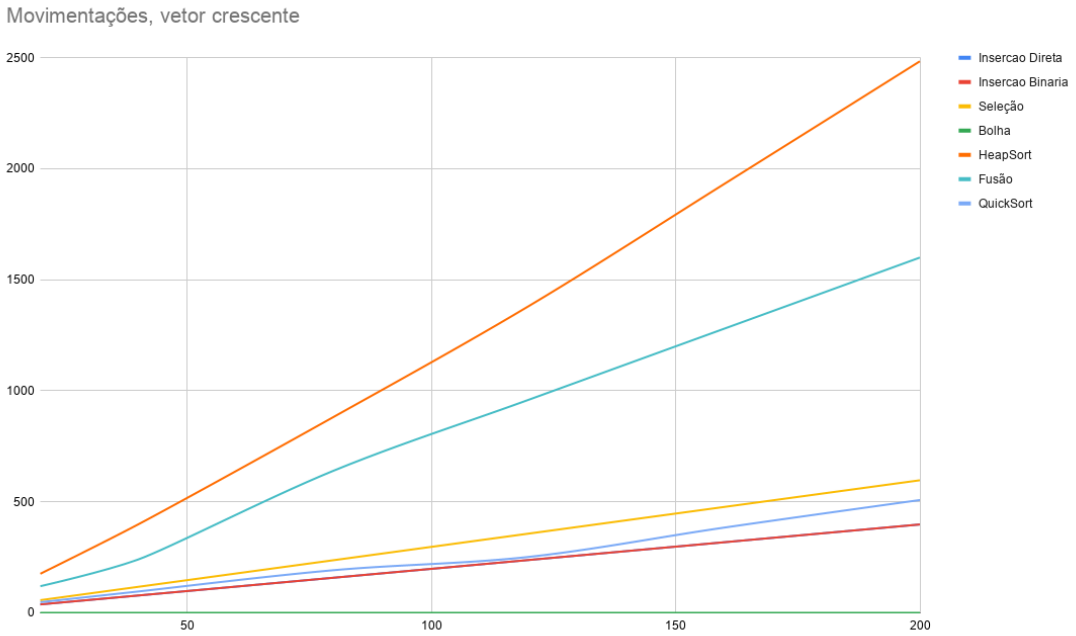
```
                comp = comp + 1;
                while(array[j] > x)
                {
                        j--;
                        comp = comp + 1;
                }

                if(i <= j)
                {
                        mov = mov + 3;
                        swap(&array[i], &array[j]);

                        i++;
                        j--;
                }

        }while(i <= j);

        if(L < j) qSort(array, L, j);

        if(R > i) qSort(array, i, R);

}
//caso inicial
void quickSort(int array[], int tamArray)
{
        qSort(array, 1, tamArray);
}

int main()
{
        int n;

        std::cin >> n;

        int *array = new int [n + 1];

        for(int i = 1; i < n+1; i++)
                std::cin >> array[i];

        quickSort(array, n);

        for(int i = 0; i < n; i++)
                std::cout << array[i] << "␣";
```

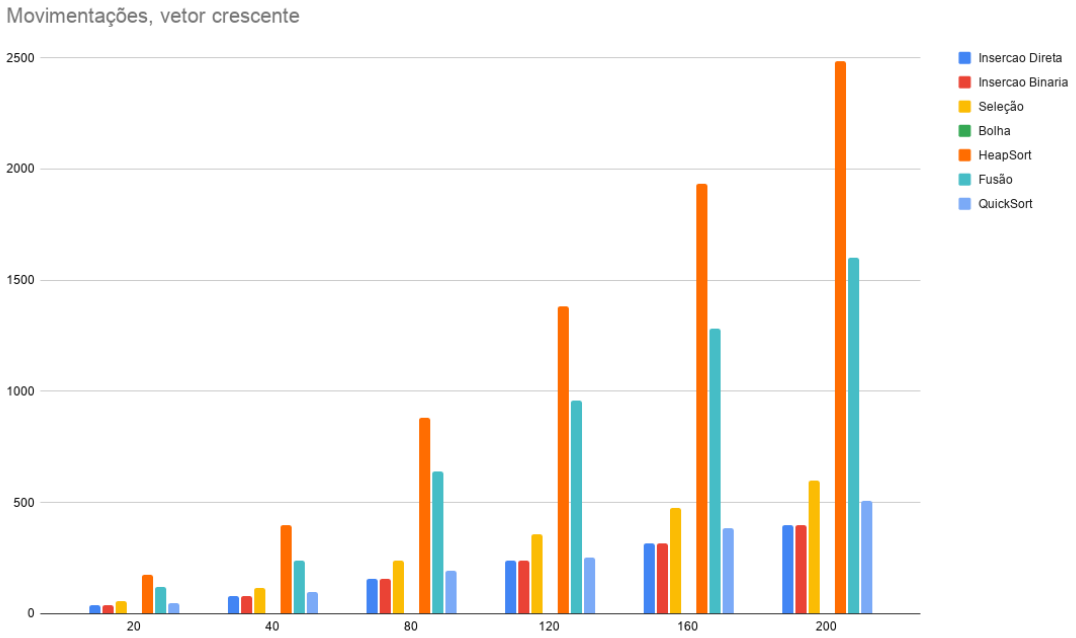
```
        std::cout << "\nComparacoes:␣" << comp
            << "\nMovimentacoes:␣" << mov << std::endl;

        return 0;
}
```

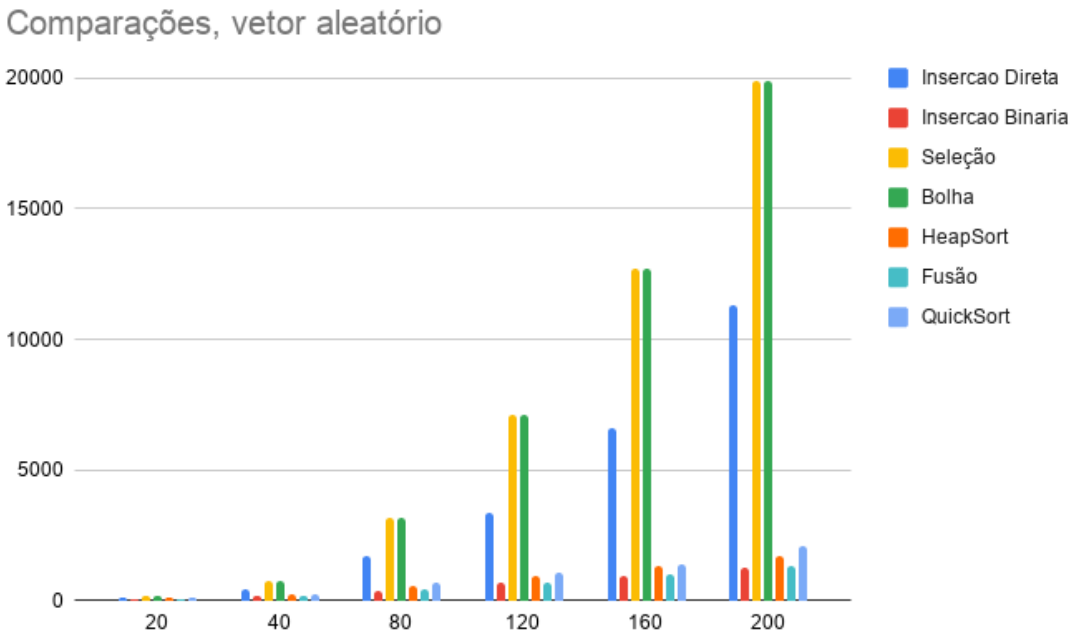
2 Gráficos

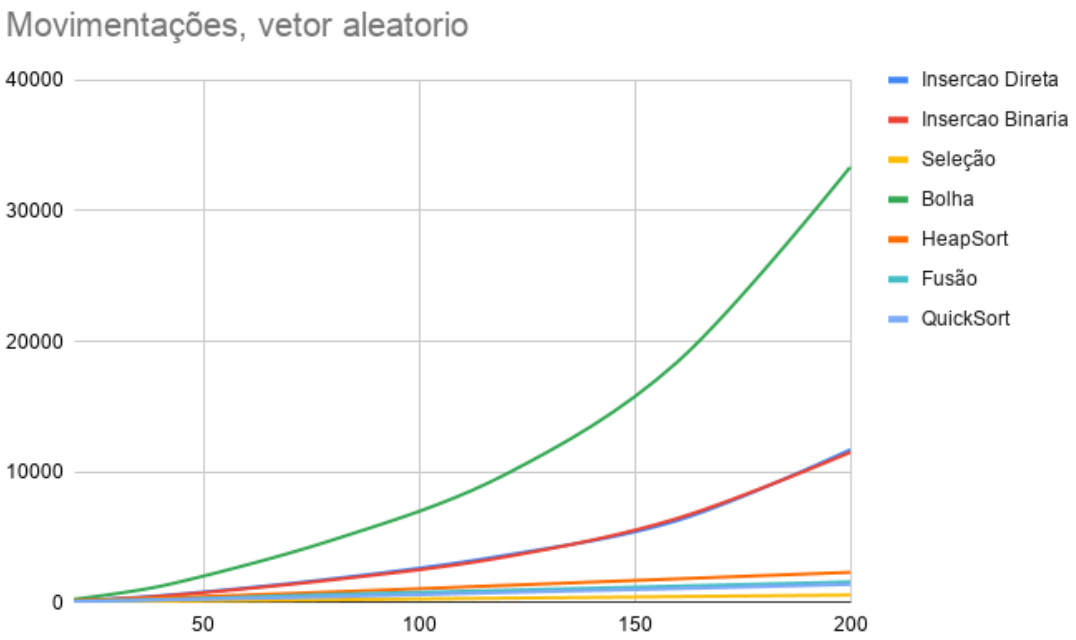
A seguir são apresentados os gráficos que serão utilizados para a análise.

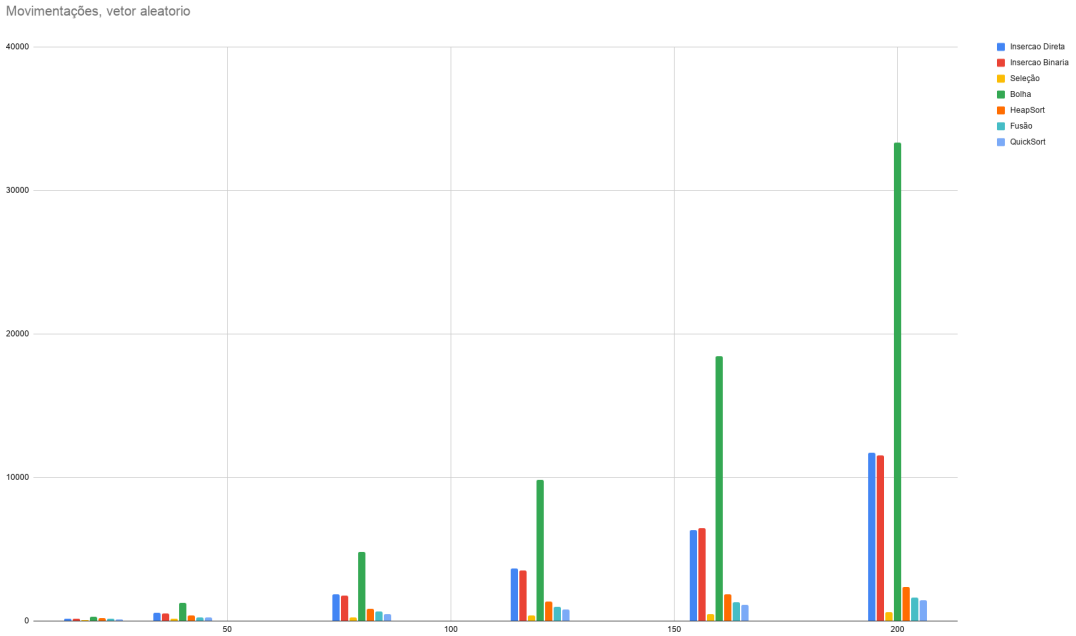


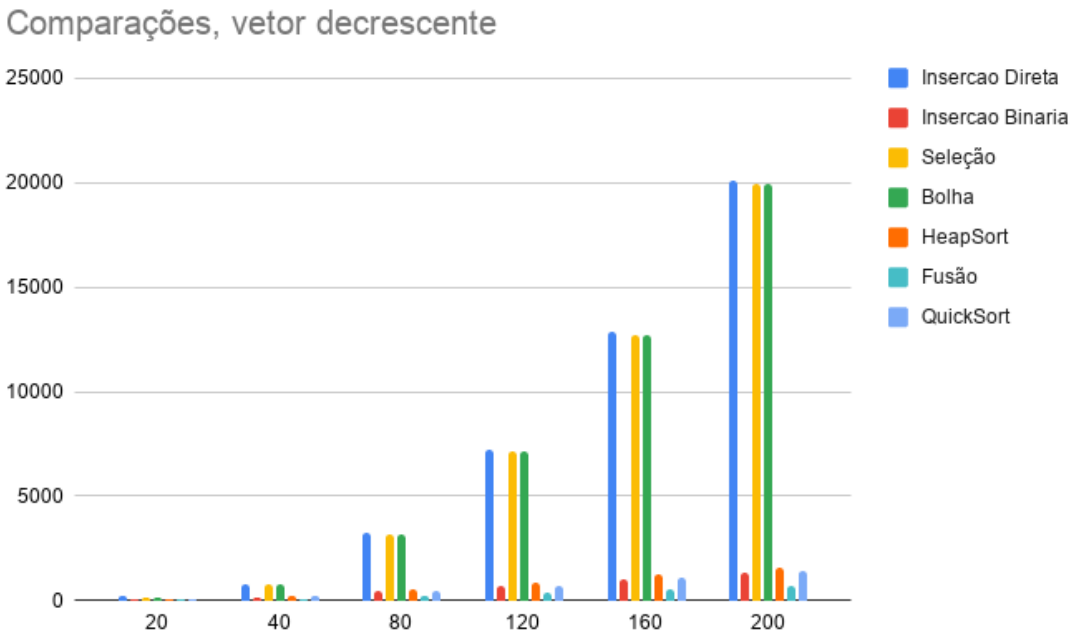


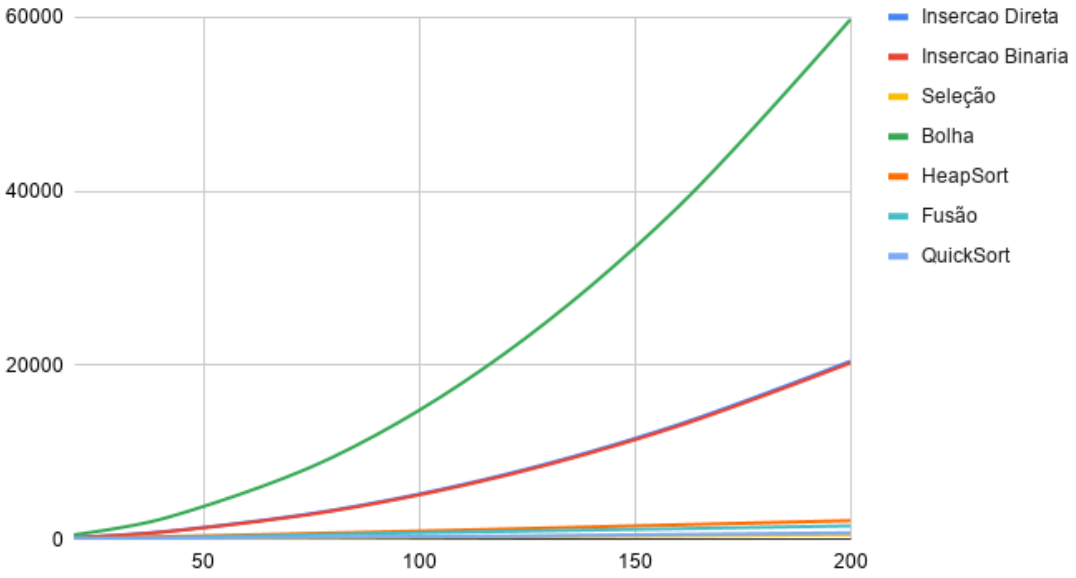

Movimentações, vetor crescente
2500
2000
1500
1000
500
0
20
40
80
120
160
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort

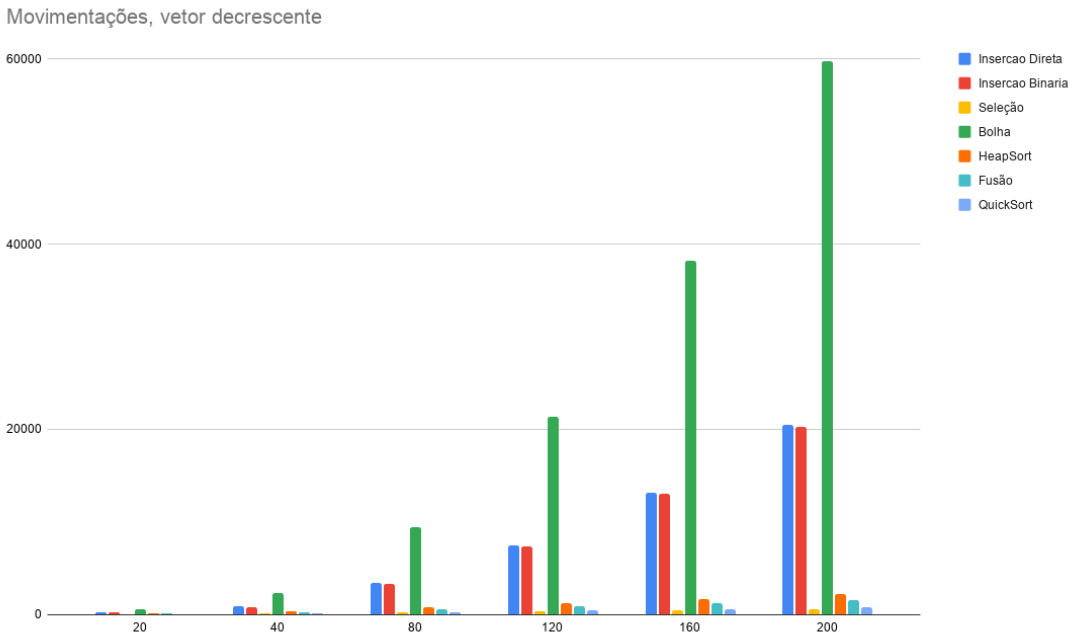

Comparações, vetor aleatório
20000
15000
10000
5000
0
20
40
80
120
160
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort


Movimentações, vetor aleatorio
40000
30000
20000
10000
0
50
100
150
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort


Movimentações, vetor aleatorio
40000
30000
20000
10000
0
50
100
150
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort


Comparações, vetor decrescente
25000
20000
15000
10000
5000
0
20
40
80
120
160
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort


Movimentações, vetor decrescente
60000
40000
20000
0
50
100
150
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort


Movimentações, vetor decrescente
60000
40000
20000
0
20
40
80
120
160
200
Insercao Direta
Insercao Binaria
Seleção
Bolha
HeapSort
Fusão
QuickSort

3 Análise dos algoritmos

Os gráficos apresentados anteriormente foram construídos a partir dos seguintes dados:

Movimentacoes								
Crescente		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	38	38	57	0	176	120	48
	40	78	78	117	0	398	240	96
	80	158	158	237	0	882	640	192
	120	238	238	357	0	1382	960	252
	160	318	318	477	0	1932	1280	384
	200	398	398	597	0	2484	1600	508
Aleatorio		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	147	128	57	270	170	120	99
	40	539	500	117	1266	372	240	219
	80	1840	1761	237	4809	833	640	466
	120	3629	3510	357	9816	1332	960	790
	160	6309	6468	477	18450	1836	1280	1087
	200	11712	11513	597	33345	2335	1600	1411
Decrescente		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	247	228	57	570	158	120	79
	40	897	858	117	2340	350	240	157
	80	3397	3318	237	9480	782	640	313
	120	7497	7378	357	21420	1240	960	429
	160	13197	13038	477	38160	1724	1280	625
	200	20497	20298	597	59700	2202	1600	805

Comparações								
Crescente		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	54	54	190	190	119	52	78
	40	143	143	780	780	281	124	191
	80	360	360	3160	3160	645	288	456
	120	600	600	7140	7140	1025	428	726
	160	873	873	12720	12720	1445	656	1065
	200	1153	1153	19900	19900	1887	844	1407
Aleatorio		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	109	62	190	190	113	65	118
	40	461	160	780	780	255	172	268
	80	1682	397	3160	3160	596	416	689
	120	3391	668	7140	7140	975	687	1105
	160	6627	950	12720	12720	1359	1025	1393
	200	11314	1257	19900	19900	1738	1318	2098
Decrescente		Insercao Direta	Insercao Binaria	Seleção	Bolha	HeapSort	Fusão	QuickSort
	20	209	69	190	190	101	40	82
	40	819	177	780	780	233	100	196
	80	3239	433	3160	3160	545	240	462
	120	7259	713	7140	7140	883	404	730
	160	12879	1025	12720	12720	1247	560	1072
	200	20099	1345	19900	19900	1605	732	1412

A partir deles, podemos notar o comportamento de crescimento do número de comparações e de movimentações para cada método à medida em que o tamanho do vetor utilizado aumenta. Foram testados vetores cujos elementos estão em ordem crescente, para representar o melhor caso; vetores com elementos em ordem decrescente, para o pior caso; e finalmente vetores com elementos aleatórios para contemplar o caso médio.

Com essas informações podemos perceber o impacto que um método não tão eficiente pode gerar na performance de um programa, além de permitir concluirmos que métodos priorizar quando não se pode evitar ter vetores já ordenados, mas com os quais a solução não deveria perder muito tempo, por exemplo.

Os gráficos e a tabela confirmam as análises assintóticas feitas em sala e isso nos permite tirar algumas conclusões.

4 Conclusões

É claro que a escolha de um método específico depende de diversos fatores, como a organização dos dados, comportamentos definidos previamente na composição desses dados, entre outras questões. Mas de maneira geral, a partir das análises realizadas anteriormente, é possivel concluir que:

1. Para um vetor cujos elementos estão em ordem crescente, a inserção binária e a inserção direta realizam, empatadas, o menor número de comparações. O Bubblesort, o menor número de movimentações: zero. Porém, o Bubble Sort realiza N^2 comparações no caso minimo, o que nos leva a concluir q a Inserção Binária é a melhor opção para esse caso. Nas palavras de Adam Drozdek, "[...]An advantage of using insertion sort is that it sorts the array only when it is really necessary. If the array is already in order, no substantial moves are performed[...]", que em tradução livre diz: Uma vantagem de usar o método da inserção é que ele ordena o array apenas quando necessário. Se ele já está ordenado, não são feitos tantos movimentos.

2. Já para um vetor com elementos posicionados aleatoriamente, o Selection Sort é o algoritmo que descreve menos movimentações, apesar de seu alto número de comparações. Com isso, se destaca dentro os demais para a tarefa.

3. Para elementos em ordem decrescente, o método mais rápido é o QuickSort. Porém, seu pior caso é n^2 e o método é instável. Por conta disso, dependendo da situação pode valer mais a pena utilizar o método da Fusão ou até mesmo Seleção. Outro ponto quanto ao uso do QuickSort é o uso de memória, que é muito maior devido ao seu caráter recursivo.

4. Vale notar um caso especial. Se os elementos do vetor são grandes entidades compostas, como vetores ou estruturas, então cada atribuição, ou movimentação de chave, pode ser muito custosa em termos de performance. Nessa situação, o Selection Sort pode ser uma boa escolha, devido ao seu baixo número de movimentações.