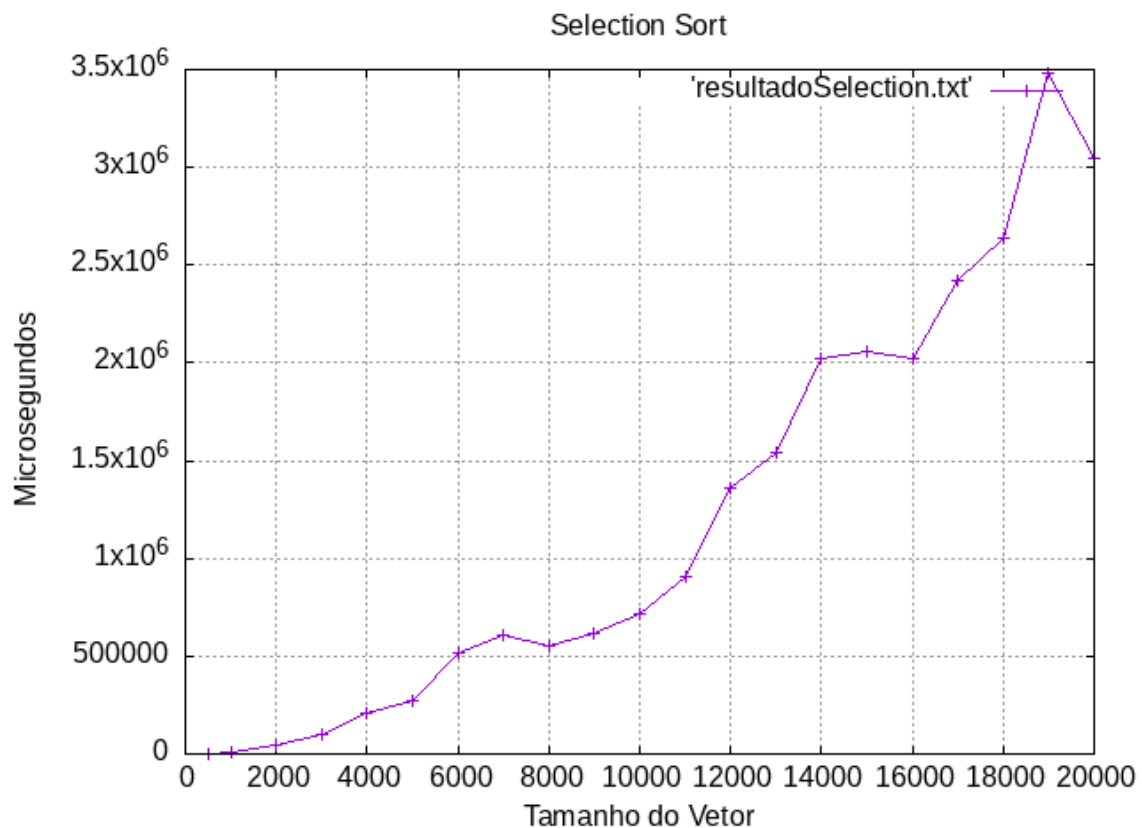


1. SelectionSort

1.1 Iterativo:

```
void SelectionSort(int n, int A[]){  
    for(int i = 0; i < n-1; i++) {  
        int min = i ;  
        for(int j = i+1; j < n; j++)  
            if(A[j] < A[min])  
                min = j;  
        swap(A[i], A[min]);  
    }  
}
```

Gráfico de desempenho: (Gostaríamos de resaltar que todos os gráficos foram gerados por dados resultantes da execução desses algoritmos em um computador com a seguinte configuração: Processador: Intel Celeron CPU N3060 @ 1.60GHz 1.60GHz. Memória RAM: 4,00 GB (utilizável 3,86 GB). SO: Windows 10 de 64 bits. E que o mesmo estava sendo utilizado para outras atividades no momento da execução dos algoritmos.)



1.2 Recursivo:

```
//Utilizada na função SelectionSortRecursiv, no intervalo de A[ind] até A[fim], retorna o índice do menor valor.
int ind_min(int ind, int A[], int fim){
    if (ind == fim)
        return ind;

    int aux = ind_min(ind+1, A, fim);

    if(A[ind] < A[aux])
        return ind;
    return aux;
}

void SelectionSortRecursiv(int n, int A[], int ind){
    if (ind == n)
        return;
    int aux = ind_min(ind, A, n-1);

    if(aux != ind)
        swap(A[aux], A[ind]);

    SelectionSortRecursiv(n, A, ind+1);
}
```

Gráfico de desempenho:

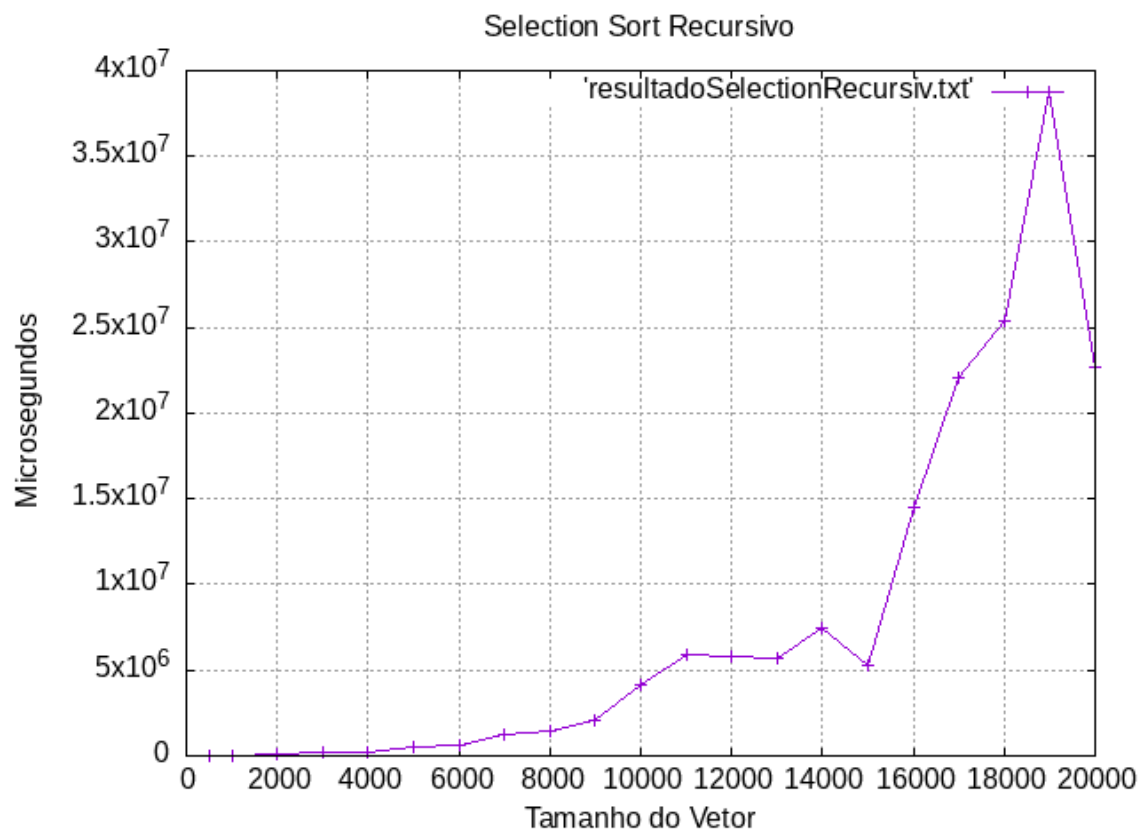
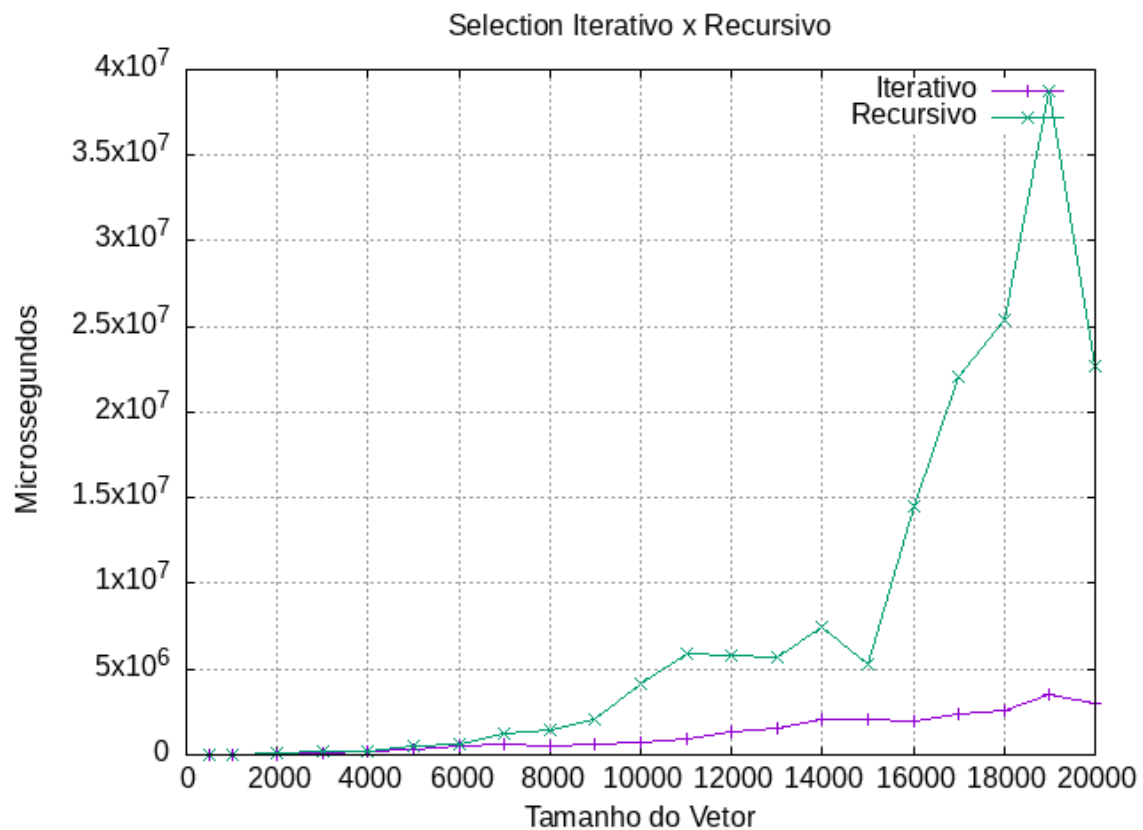


Gráfico que mostra a comparação de desempenho entre as versões iterativa e recursiva do SelectionSort:

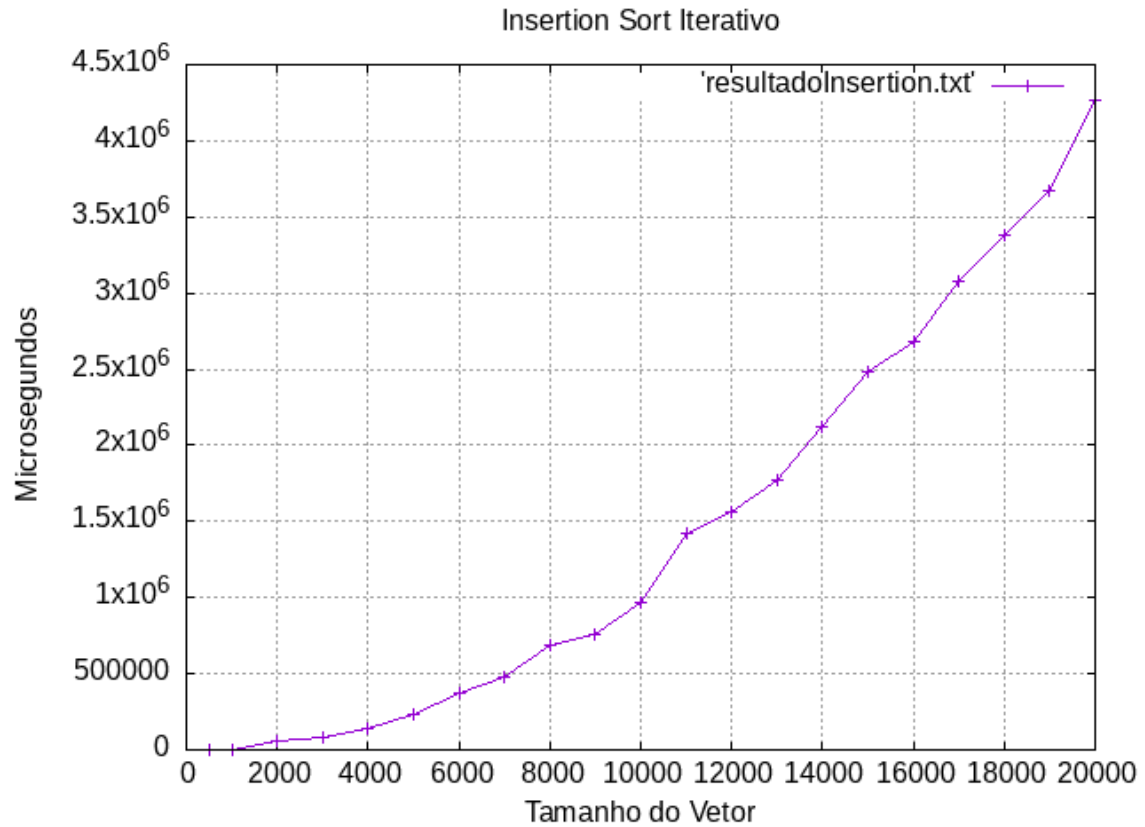


2. InsertionSort

2.1 Iterativo:

```
void InsertionSort(int n, int A[]){
    int i, j, key;
    for (j = 1; j < n; j++) {
        key = A[j];
        i = j-1;
        while (i >= 0 && A[i] > key){
            A[i+1] = A[i];
            i--;
        }
        A[i+1] = key;
    }
}
```

Gráfico de desempenho:



2.2 Recursivo:

```
void InsertionSortRecursiv(int n, int A[]){
    if (n <= 1) return;

    InsertionSortRecursiv(n-1, A);
    int aux = n-2;
    int ultim = A[n-1];
    while(aux >= 0 && A[aux] > ultim){
        A[aux+1] = A[aux];
        aux--;
    }
    A[aux+1] = ultim;
}
```

Gráfico de desempenho:

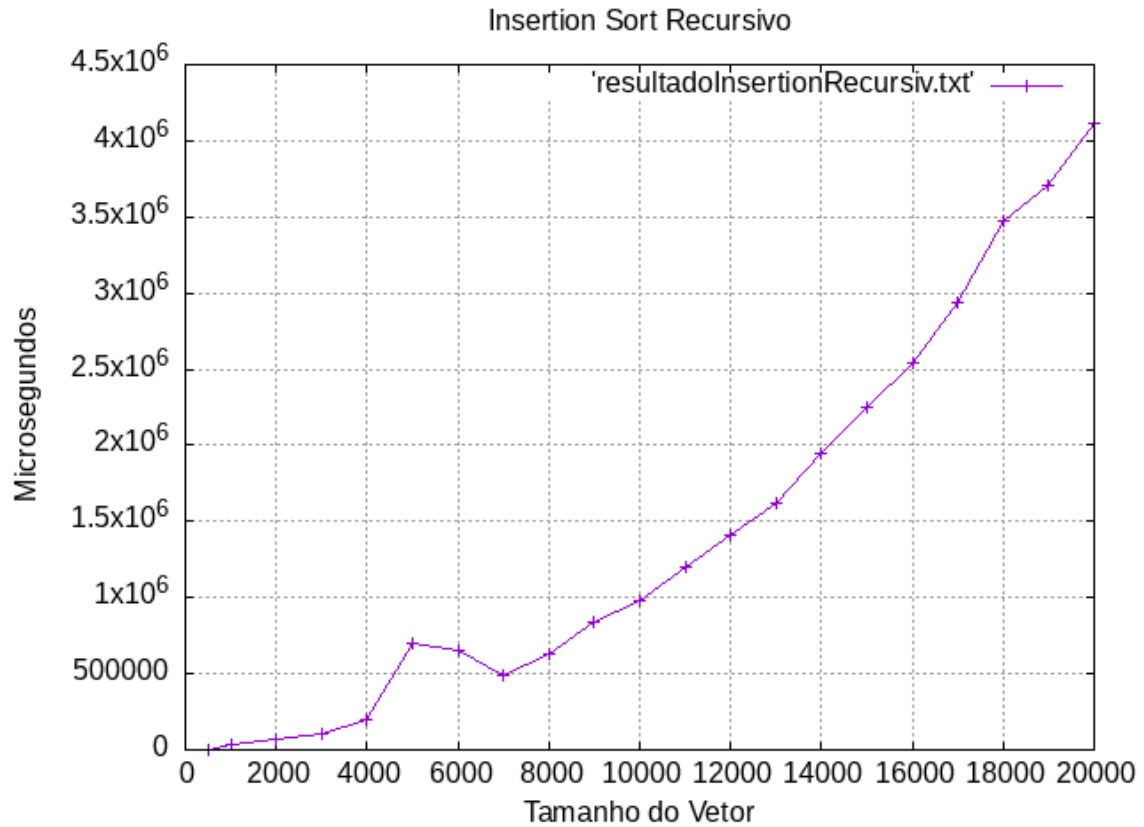
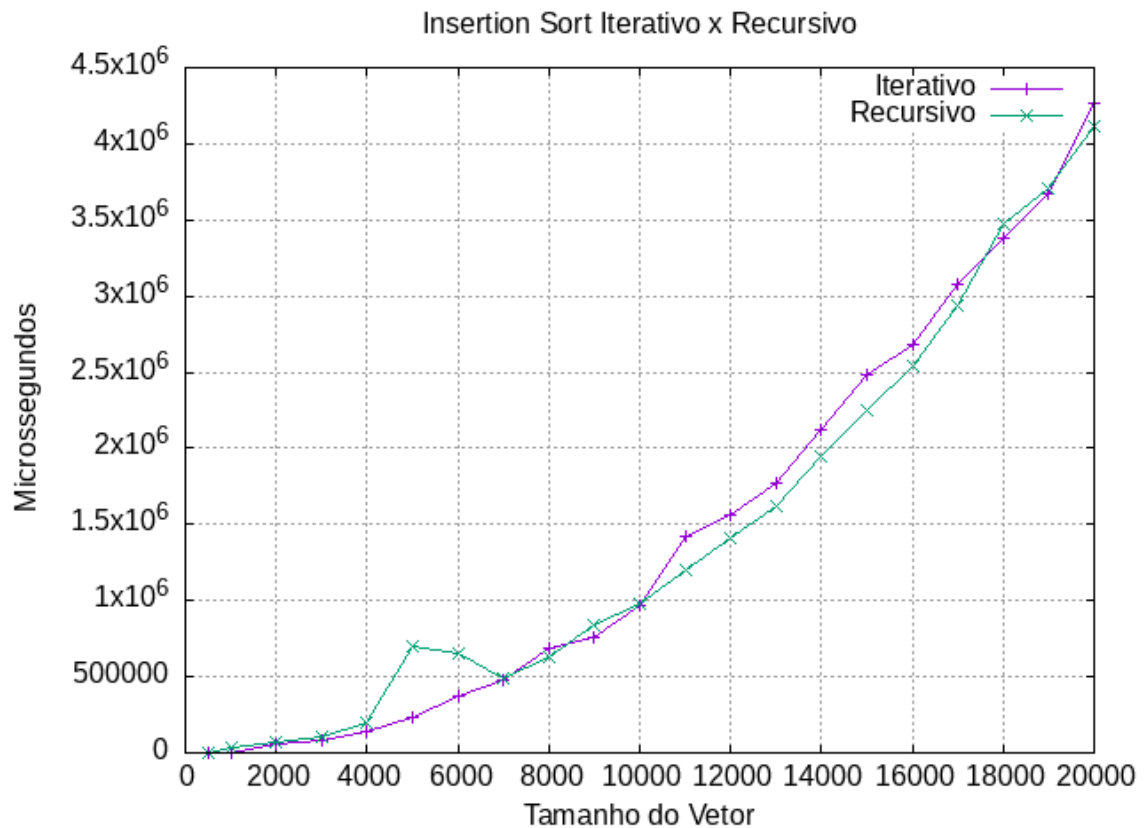


Gráfico que mostra a comparação de desempenho entre as versões iterativa e recursiva do InsertionSort:



3. MergeSort

3.1 Iterativo:

```
//Utilizada na função MergeSort, compara dois inteiros e retorna o menor deles
int min(int x, int y){
    if(x < y)
        return x;
    else
        return y;
}

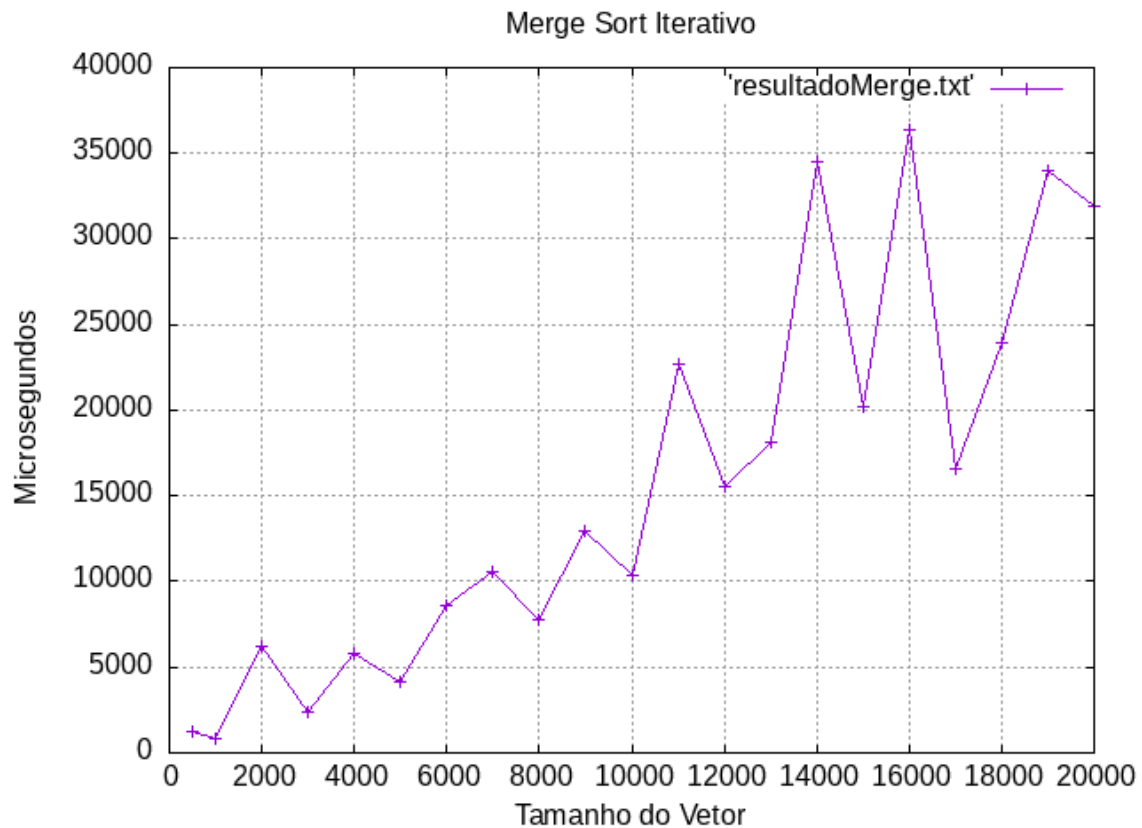
void merge(int A[], int e, int m, int d){
    int i, j, k;
    int n1 = m - e + 1;
    int n2 = d - m;

    int E[n1], D[n2];

    for (i = 0; i < n1; i++)
        E[i] = A[e+i];
    for (j = 0; j < n2; j++)
        D[j] = A[m+1+j];

    i = 0; j = 0; k = e;
    while(i < n1 && j < n2){
        if(E[i] <= D[j]){
            A[k] = E[i];
            i++;
        }
        else{
            A[k] = D[j];
            j++;
        }
        k++;
    }
    while(i < n1){
        A[k] = E[i];
        i++;
        k++;
    }
    while(j < n2){
        A[k] = D[j];
        j++;
        k++;
    }
}
```

Gráfico de desempenho:



3.2 Recursivo:

```
void Intercala(int A[], int p, int q, int r){
    int i, j, k;
    int *W = new int [r - p+1]; // Vetor auxiliar
    i = p;
    j = q+1;
    k = 0;
    // Intercala A[p..q] e A[q+1.. r]
    while(i <= q && j <= r) {
        if(A[i] <= A[j])
            W[k++] = A[i++];
        else
            W[k++] = A[j++];
    }
    while(i <= q) W[k++] = A[i++];
    while(j <= r) W[k++] = A[j++];
    // Copia vetor ordenado W para o vetor A
    for(i = p; i <= r; i++)
        A[i] = W[i-p];
    delete[] W;
}

void MergeSortRecursiv(int A[], int p, int r){
    if (p < r) {
        int q = (p + r) / 2; // Dividir
        // Conquistar
        MergeSortRecursiv(A, p, q);
        MergeSortRecursiv(A, q + 1, r);
        // Combinar
        Intercala(A, p, q, r);
    }
}
```

Gráfico de desempenho:

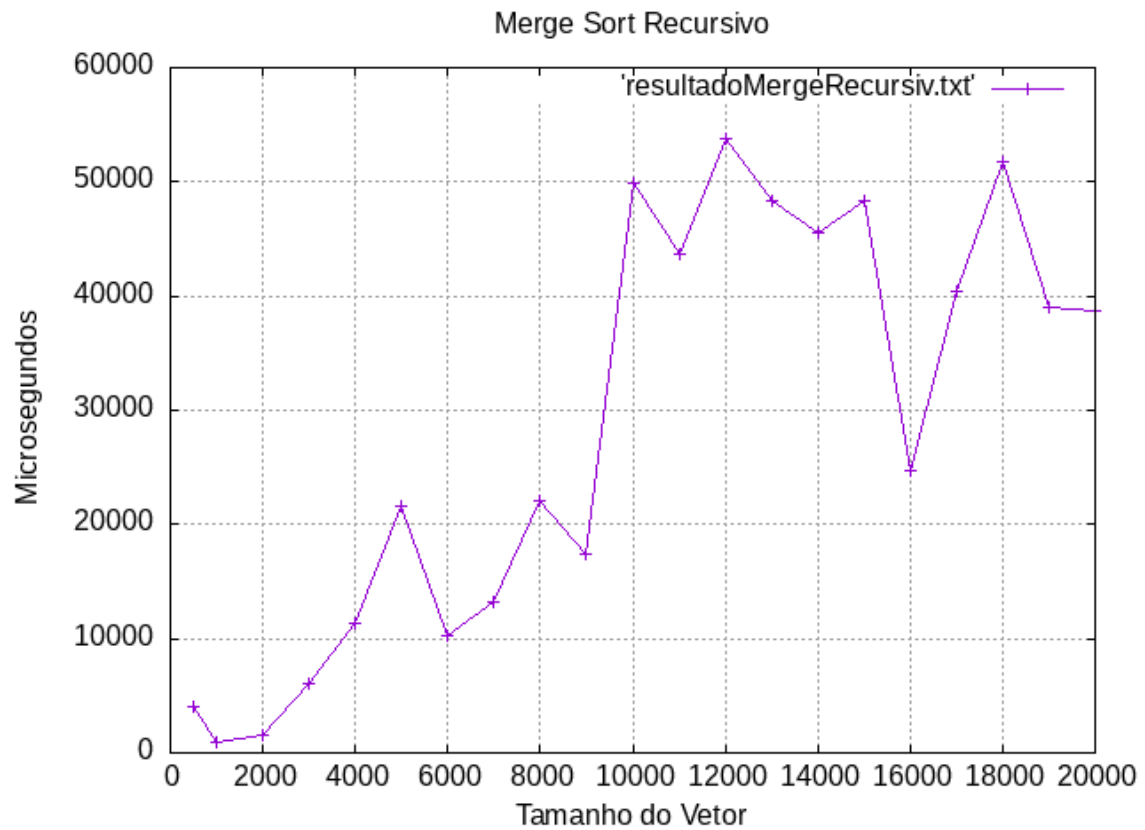
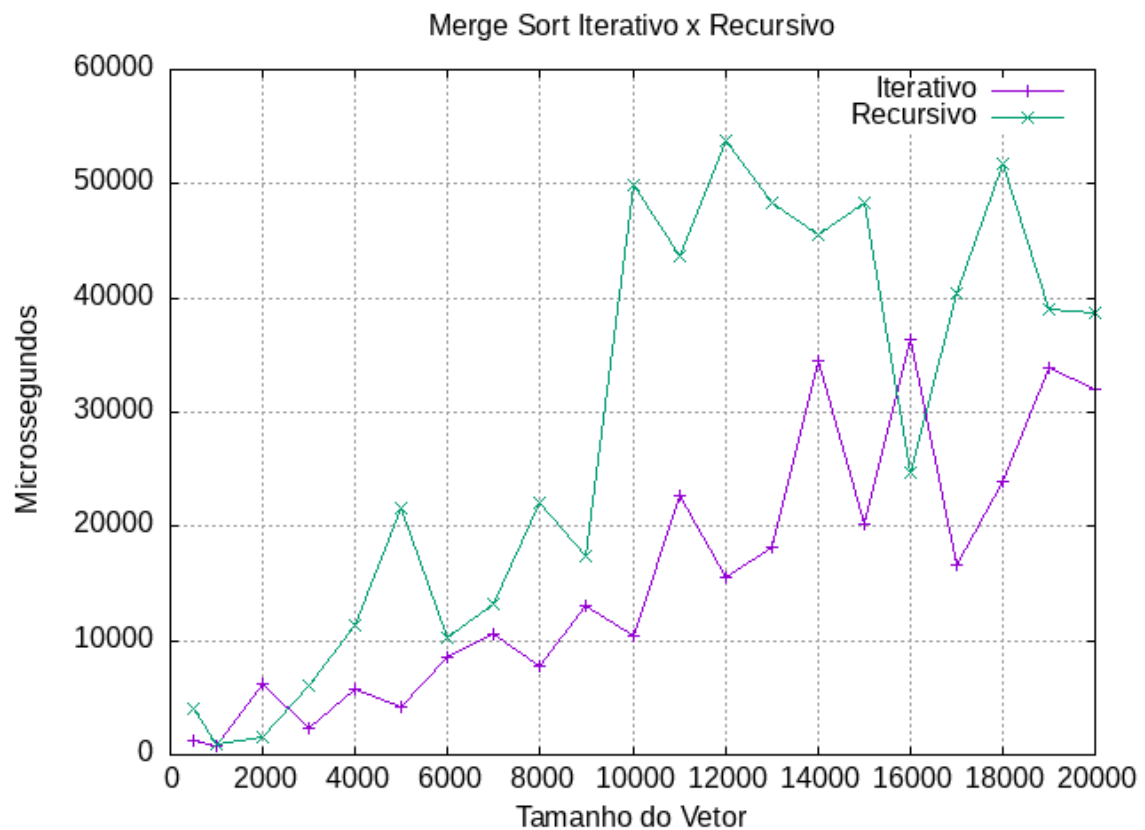


Gráfico que mostra a comparação de desempenho entre as versões iterativa e recursiva do MergeSort:



4. QuickSort

4.1 Iterativo:

```
void QuickSort(int A[], int e, int d){
    int pilha[d - e + 1];
    int top = -1;

    pilha[++top] = e;
    pilha[++top] = d;

    while(top >= 0){
        d = pilha[top--];
        e = pilha[top--];

        int p = separa(A, e, d);
        if (p - 1 > e) {
            pilha[++top] = e;
            pilha[++top] = p - 1;
        }
        if (p + 1 < d) {
            pilha[++top] = p + 1;
            pilha[++top] = d;
        }
    }
}
```

Gráfico de desempenho:

```
/* Recebe um vetor A[p..r] com p <= r.  
2 * Rearranja os elementos do vetor e devolve  
3 * j em p..r tal que A[p..j -1] <= A[j] < A[j +1.. r  
].  
4 */  
int separa(int A[], int p, int r){  
    int c = A[r];  
    int j = p;  
    for(int k = p; k < r; k++){  
        if(A[k] <= c){  
            swap(A[k], A[j]);  
            j++;  
        }  
    }  
    A[r] = A[j];  
    A[j] = c;  
    return j;  
}  
void QuickSortRecursov(int A[], int p, int r){  
    if(p < r){  
        int i = separa(A, p, r);  
        QuickSortRecursov(A, p, i -1);  
        QuickSortRecursov(A, i+1, r);  
    }  
}
```

4.2 Recursi
vo:

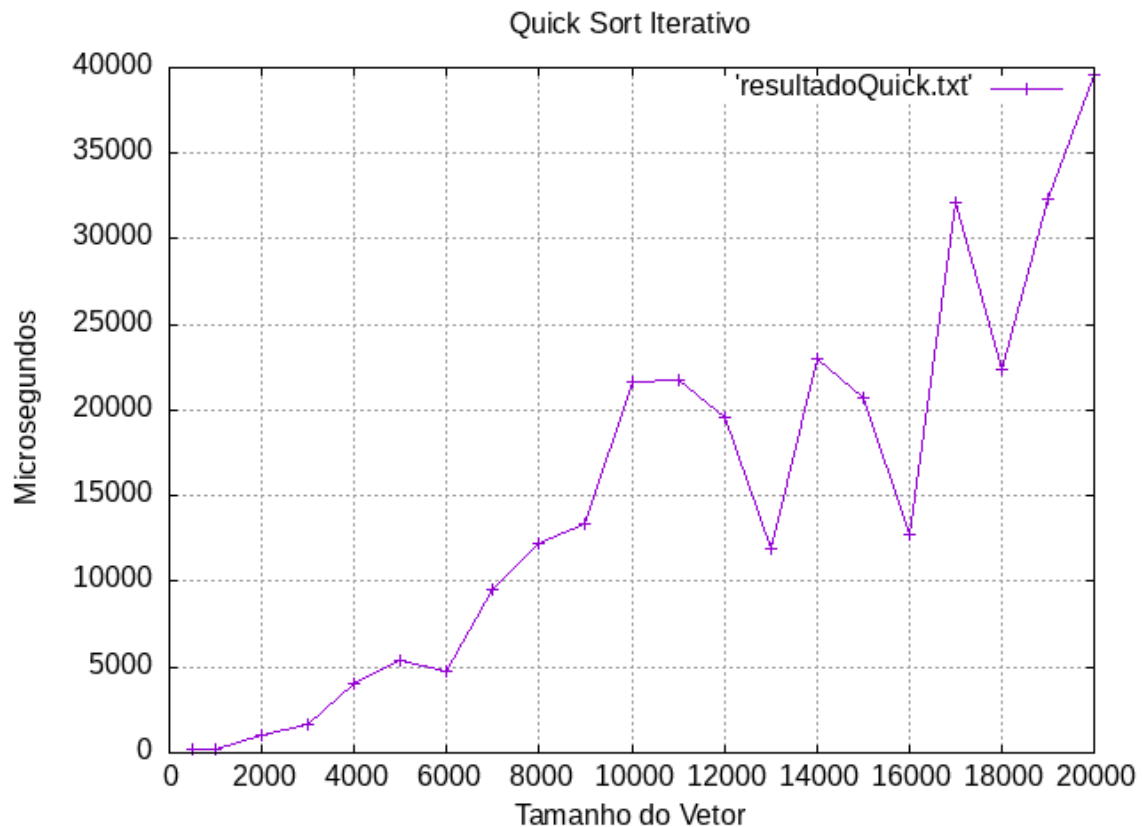
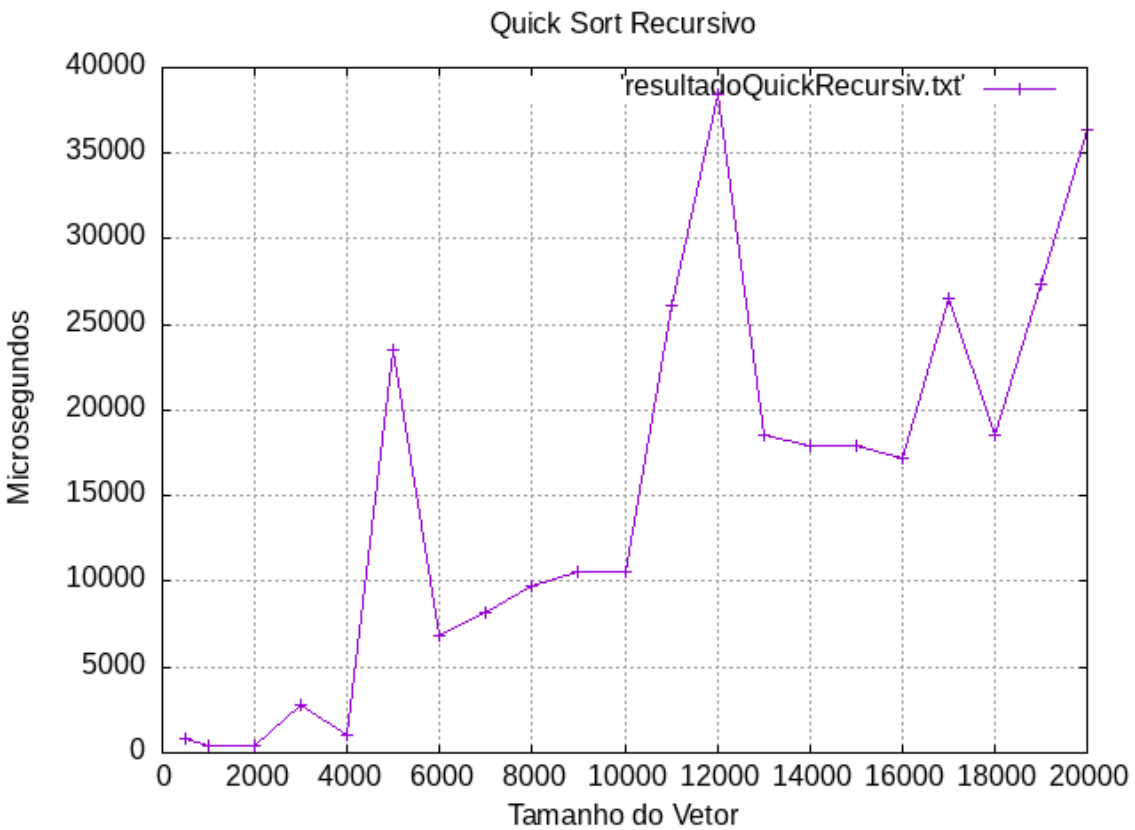
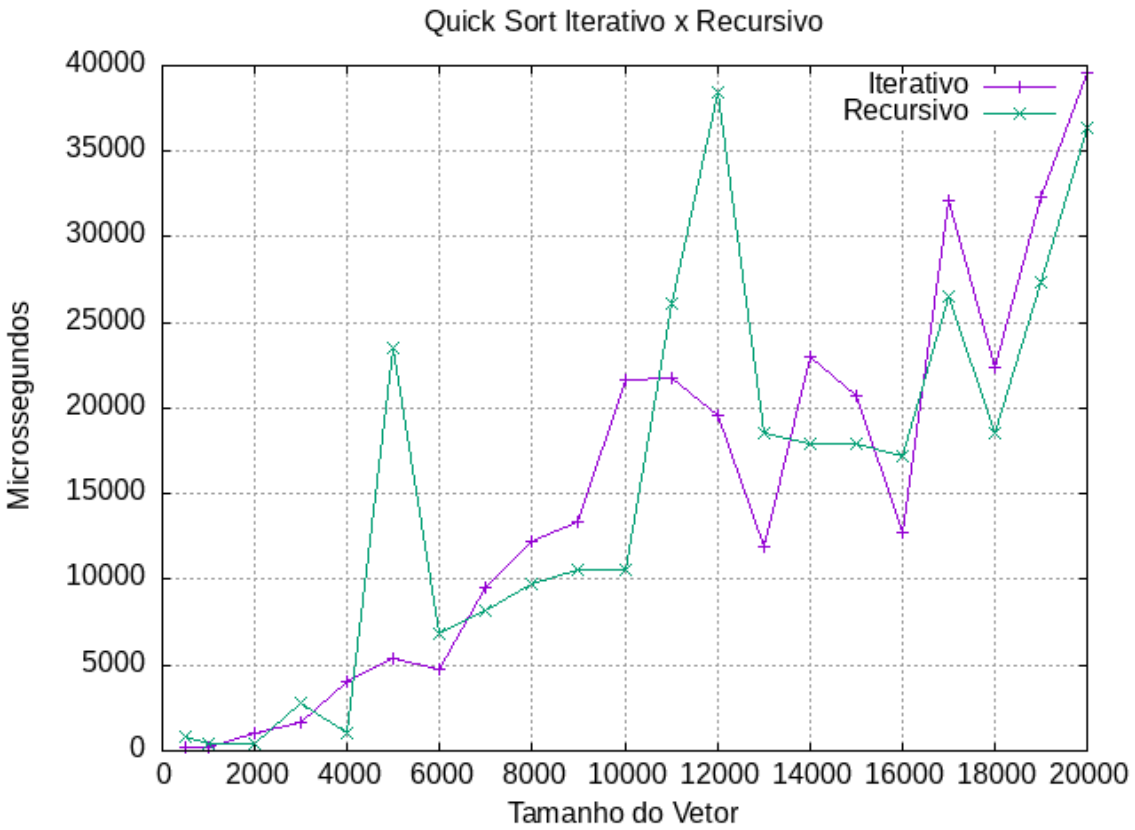


Gráfico de desempenho:
Gráfico que mostra a comparação de desempenho entre as versões iterativa e recursiva do QuickSort:



Divisão do trabalho

Dividimos as tarefas do trabalho de forma bem igualitária, cada um contribuiu tanto com a parte da implementação do código, como na criação dos gráficos.

Dificuldades encontradas

Recursão, em alguns casos com certeza ainda temos dificuldades em encherar esse processo, porém conseguimos implementar os respectivos algoritmos da melhor maneira possível em nossa visão. Sobre a criação dos gráficos, nós primeiro optamos por fazer com python, porém, não estávamos conseguindo realizar uma determinada conversão numérica para que um dos eixos do gráfico ficasse com melhor aspecto de visualização, então descidimos por fazer com o gnuplot mesmo, o que facilitou bastante nosso trabalho.

Concluimos que os algoritmos MergeSort e QuickSort realmente são muito mais rápidos em relação aos dois primeiros, ao executar nosso programa, você perceberá também que os algoritmos CockTailSort e BubbleSort também são de fato muito ruins quando se trata de rapidez.

Referências

<https://www.geeksforgeeks.org/>
<https://stackoverflow.com/>
<https://www.youtube.com/>