Teste o algoritmo com diferentes tamanhos de conjuntos de entrada e verifique sua eficácia.

```
import time
import random

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
```

```
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
    return arr

def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def test_sorting_algorithm(algorithm, arr):
    start_time = time.time()
    sorted_arr = algorithm(arr.copy())
    end_time = time.time()
    return end_time - start_time, sorted_arr

sizes = [10, 100, 1000, 5000, 10000]
algorithms = {
    "Bubble Sort": bubble_sort,
    "Selection Sort": selection_sort,
    "Insertion Sort": insertion_sort,
```

```
    "Merge Sort": merge_sort,
}

results = {size: {} for size in sizes}

for size in sizes:
    arr = random.sample(range(size * 10), size)
    for name, algorithm in algorithms.items():
        execution_time, _ = test_sorting_algorithm(algorithm, arr)
        results[size][name] = execution_time

for size in sizes:
    print(f"\nTamanho do conjunto de entrada: {size}")
    for name in algorithms:
        print(f"{name}: Tempo de execução: {results[size][name]:.6f}
segundos")

binary_search_results = {}

for size in sizes:
    sorted_arr = sorted(random.sample(range(size * 10), size))
    target = sorted_arr[random.randint(0, size - 1)]
    start_time = time.time()
    index = binary_search(sorted_arr, target)
    end_time = time.time()
    binary_search_results[size] = end_time - start_time

print("\nTempos de execução da busca binária:")
for size in sizes:
    print(f"Tamanho do conjunto de entrada: {size}, Tempo de execução:
{binary_search_results[size]:.6f} segundos")
```

Resultado:

```
Tamanho do conjunto de entrada: 10
Bubble Sort: Tempo de execução: 0.000000 segundos
Selection Sort: Tempo de execução: 0.000000 segundos
Insertion Sort: Tempo de execução: 0.000000 segundos
Merge Sort: Tempo de execução: 0.000000 segundos

Tamanho do conjunto de entrada: 100
Bubble Sort: Tempo de execução: 0.000000 segundos
Selection Sort: Tempo de execução: 0.000000 segundos
Insertion Sort: Tempo de execução: 0.000449 segundos
Merge Sort: Tempo de execução: 0.000000 segundos

Tamanho do conjunto de entrada: 1000
Bubble Sort: Tempo de execução: 0.026886 segundos
Selection Sort: Tempo de execução: 0.012040 segundos
Insertion Sort: Tempo de execução: 0.010787 segundos
Merge Sort: Tempo de execução: 0.001000 segundos

Tamanho do conjunto de entrada: 5000
Bubble Sort: Tempo de execução: 0.643393 segundos
Selection Sort: Tempo de execução: 0.258048 segundos
Insertion Sort: Tempo de execução: 0.257021 segundos
Merge Sort: Tempo de execução: 0.005505 segundos

Tamanho do conjunto de entrada: 10000
Bubble Sort: Tempo de execução: 2.568656 segundos
Selection Sort: Tempo de execução: 1.058237 segundos
Insertion Sort: Tempo de execução: 1.105878 segundos
Merge Sort: Tempo de execução: 0.011565 segundos

Tempos de execução da busca binária:
Tamanho do conjunto de entrada: 10, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 100, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 1000, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 5000, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 10000, Tempo de execução: 0.000000 segundos
```

Compare a eficiência do algoritmo de ordenação utilizado com outros algoritmos de ordenação conhecidos.

```python
import time
import random

def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if arr[j] < arr[min_index]:
                min_index = j
        arr[i], arr[min_index] = arr[min_index], arr[i]
    return arr

def insertion_sort(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
```

```
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
    return arr

def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def test_sorting_algorithm(algorithm, arr):
    start_time = time.time()
    sorted_arr = algorithm(arr.copy())
    end_time = time.time()
    return end_time - start_time, sorted_arr

sizes = [10, 100, 1000, 5000, 10000]
algorithms = {
    "Bubble Sort": bubble_sort,
    "Selection Sort": selection_sort,
    "Insertion Sort": insertion_sort,
    "Merge Sort": merge_sort,
}

results = {size: {} for size in sizes}
```

```
for size in sizes:
    arr = random.sample(range(size * 10), size)
    for name, algorithm in algorithms.items():
        execution_time, _ = test_sorting_algorithm(algorithm, arr)
        results[size][name] = execution_time

for size in sizes:
    print(f"\nTamanho do conjunto de entrada: {size}")
    for name in algorithms:
        print(f"{name}: Tempo de execução: {results[size][name]:.6f}
segundos")

binary_search_results = {}

for size in sizes:
    sorted_arr = sorted(random.sample(range(size * 10), size))
    target = sorted_arr[random.randint(0, size - 1)]
    start_time = time.time()
    index = binary_search(sorted_arr, target)
    end_time = time.time()
    binary_search_results[size] = end_time - start_time

print("\nTempos de execução da busca binária:")
for size in sizes:
    print(f"Tamanho do conjunto de entrada: {size}, Tempo de execução:
{binary_search_results[size]:.6f} segundos")
```

Resultado:

```
Tamanho do conjunto de entrada: 10
Bubble Sort: Tempo de execução: 0.000000 segundos
Selection Sort: Tempo de execução: 0.000000 segundos
Insertion Sort: Tempo de execução: 0.000000 segundos
Merge Sort: Tempo de execução: 0.000000 segundos

Tamanho do conjunto de entrada: 100
Bubble Sort: Tempo de execução: 0.000000 segundos
Selection Sort: Tempo de execução: 0.000000 segundos
Insertion Sort: Tempo de execução: 0.000000 segundos
Merge Sort: Tempo de execução: 0.000000 segundos

Tamanho do conjunto de entrada: 1000
Bubble Sort: Tempo de execução: 0.025541 segundos
Selection Sort: Tempo de execução: 0.010025 segundos
Insertion Sort: Tempo de execução: 0.010992 segundos
Merge Sort: Tempo de execução: 0.001000 segundos

Tamanho do conjunto de entrada: 5000
Bubble Sort: Tempo de execução: 0.664843 segundos
Selection Sort: Tempo de execução: 0.264551 segundos
Insertion Sort: Tempo de execução: 0.259803 segundos
Merge Sort: Tempo de execução: 0.006300 segundos

Tamanho do conjunto de entrada: 10000
Bubble Sort: Tempo de execução: 2.759660 segundos
Selection Sort: Tempo de execução: 1.042150 segundos
Insertion Sort: Tempo de execução: 1.077641 segundos
Merge Sort: Tempo de execução: 0.013008 segundos

Tempos de execução da busca binária:
Tamanho do conjunto de entrada: 10, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 100, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 1000, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 5000, Tempo de execução: 0.000000 segundos
Tamanho do conjunto de entrada: 10000, Tempo de execução: 0.000000 segundos
```