# assignment2

## Individual report for bth004


| | |
|---|---|
| Name: | YuKi |
| Student ID: | xxxxxxxxxxxxx |
| Date: : | December 15, 2023 |

# Content

# 1 Implementation

## 1.1 Bubble Sort

It repeatedly steps through the input list element by element, while element of adjacent elements are compared if a reverse order is found, swaps them. And repeating until no swaps are needed, that means the list has become fully sorted.

Step-by-step looking:

1. Compare adjacent elements, if a reverse order is found, swap them.
2. Repeat the step 1 for each pair of adjacent elements.
3. After step 2 is completed, the final element will be sorted.
4. If none of the pairs of elements in the above steps need to be swapped, the list has become fully sorted. Otherwise, repeat the above steps for all elements except those that are already sorted.

## 1.2 Merge Sort

It continuously divides the input list into sublists, then merges the sorted sublists in order until an ordered list is formed.

Step-by-step looking:

1. Divide the unsorted list into $n$ sublists, each containing one element (a list of one element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

# 2 Pseudo Code

## 2.1 Bubble Sort

**Algorithm 1** Bubble Sort

**Input** Input list $arr$
**for** $i \leftarrow 0; i < n; i++$ **do**
  $flag\_noSwap \leftarrow true$
  **for** $j \leftarrow 0; j < n - i - 1; j++$ **do**
    **if** $arr[j] > arr[j+1]$ **then**
      $swap\ (arr[j], arr[j+1])$
      $flag\_noSwap \leftarrow false$
    **end if**
  **end for**
  **if** $flag\_noSwap$ **then**
    break
  **end if**
**end for**
**Output** Sorted list $arr$

## 2.2 Merge Sort

**Algorithm 2** Merge Sort

**Input** Input list $arr$
**function** $merge(arr_{left}, arr_{right})$
  **while** $arr_{left}$ not empty && $arr_{right}$ not empty **do**
    **if** $arr_{left}[0] < arr_{right}[0]$ **then**
      append $arr_{left}[0]$ to $arr$
      remove $arr_{left}[0]$ from $arr_{left}$
    **else**
      append $arr_{right}[0]$ to $arr$
      remove $arr_{right}[0]$ from $arr_{right}$
    **end if**
  **end while**
  append remaining elements in $arr_{left}$ to $arr$ in order
  append remaining elements in $arr_{right}$ to $arr$ in order
  **return** $arr$
**function** $mergeSort(arr)$
  **if** $n > 1$ **then**
    $mid \leftarrow \lfloor len(arr) \rfloor$
    $arr_{left} \leftarrow mergeSort(arr[:mid])$
    $arr_{right} \leftarrow mergeSort(arr[mid:])$
    $arr \leftarrow merge(arr_{left}, arr_{right})$
  **end if**
$mergeSort(arr)$
**Output** Sorted list $arr$

# 3 Theoretical Analysis

Set that the length of the input list $arr$ is $n$. For the requirement in this assignment, the basic operation is "comparing two numbers".

## 3.1 Bubble Sort

In the best-case, when the list is already sorted, only the first $for$ loop needs to be executed. So the amount of basic operation is $n$.

In the worst-case, when the array is in completely reverse order, two levels of $for$ loop needs to be executed, and every pair of elements need to be compared. So the amount of basic operation is $n(n-1)/2$.

In the average-case, tow levels of $for$ loop needs to be executed, and most pairs of elements need to be compared. So the amount of basic operation is order of magnitude of $n^2$.

So, the bubble sort algorithm's best time complexity is $O(n)$, average/worst time complexity is $O(n^2)$.

## 3.2 Merge Sort

Through the recursion formula, we can know time complexity of merge sort (the time complexity of merging sublists is O(n)):

$$T(n) = 2T(\frac{n}{2}) + O(n)$$

We can use $MasterTheorem$ to calculate

$$T(n) = O(nlog_2(n))$$

# 4 Practical Analysis

## 4.1 Test data of type of Int from $10^1$ to $10^5$

```
-------------Comparison for type of Int 10^1-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Int 10^2-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Int 10^3-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.046875 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Int 10^4-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 4.0 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.015625 seconds

-------------Comparison for type of Int 10^5-------------
Bubble Sort:
Initialization Time: 0.015625 seconds
Sorting Time: 426.859375 seconds
Merge Sort:
Initialization Time: 0.015625 seconds
Sorting Time: 0.25 seconds
```

### 4.2 Test data of type of Float from $10^1$ to $10^5$

```
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Float 10^2-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Float 10^3-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.046875 seconds
Merge Sort:
Initialization Time: 0.0 seconds
Sorting Time: 0.0 seconds

-------------Comparison for type of Float 10^4-------------
Bubble Sort:
Initialization Time: 0.0 seconds
Sorting Time: 5.9375 seconds
Merge Sort:
Initialization Time: 0.015625 seconds
Sorting Time: 0.03125 seconds

-------------Comparison for type of Float 10^5-------------
Bubble Sort:
Initialization Time: 0.046875 seconds
Sorting Time: 635.546875 seconds
Merge Sort:
Initialization Time: 0.046875 seconds
Sorting Time: 0.3125 seconds
```

# 5 Question Answer

**Question**:

At what input size do you consider the time required for initialization to be negligible in relation to the total running time of the algorithm?

**Answer**:

Whether the initialization time is negligible relative to the total running time of the algorithm depends on several factors.

In this assignment, for an input sequence of length $n$, $n$ operations are required. It means the time complexity of initialization is $O(n)$. And the time complexity of bubble sort is $O(n^2)$, the time complexity of merge sort is $O(nlog_2(n))$.

Comparing $O(n)$, $O(n^2)$ and $O(nlog_2(n))$, we can think that when $n$ is relatively small, the difference between $O(n)$, $O(n^2)$ and $O(nlog_2(n))$ is not big, and the time to initialize the data needs to be considered. But when $n$ is large, $O(n)$ is much smaller than $O(n^2)$ and $O(nlog_2(n))$, the time to initialize the data can be negligible.

In the *4 Practical Analysis*(in my PC), we can see when the magnitude of the data comes to $10^5$, the initialization time is dozens or even hundreds of times smaller than the time of sorting. So with this sample, we could consider the time required for initialization to be negligible in relation to the total running time of the algorithm.

# 6 Comparison Of Theoretical And Practical Analysis

## 6.1 Bubble Sort

In *3 Theoretical Analysis*, we can see the time complexity of bubble sort is $O(n^2)$.

In *4 Practical Analysis*, we can see the CPU time of bubble sort increases rapidly after $n$ becomes $10^3$, for type of Int,

$$Time(10^5) = 426s = 106 \cdot Time(10^4) = 106 \times 4s = 10^4 \cdot Time(10^3) = 10^4 \cdot 0.046875s$$

for type of Float,

$$Time(10^5) = 635.5469s = 107 \cdot Time(10^4) = 107 \times 5.9375s = 10^4 \cdot Time(10^3) = 10^4 \cdot 0.0469s$$

This is perfectly consistent with $O(n^2)$ in the theoretical analysis.

## 6.2 Merge Sort

In *3 Theoretical Analysis*, we can see the time complexity of merge sort is $O(nlog_2(n))$.

In *4 Practical Analysis*, the CPU time of merge sort grows very slowly, but grows very fast after $n$ becomes $10^4$, for type of Int,

$$Time(10^5) = 0.25s = 16 \cdot Time(10^4) = 16 \cdot 0.015625s$$

for type of Float,

$$Time(10^5) = 0.3125s = 10 \cdot Time(10^4) = 10 \times 0.03125s$$

And,

$$10^5 \cdot log_2(10^5) \div 10^4 \cdot log_2(10^4) = 12.5$$

This is perfectly consistent with $O(nlog_2(n))$ in the theoretical analysis.

# 7 Python Code

## 7.1 Bubble Sort

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        flag = True
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                flag = False
        if flag:
            break
```

## 7.2 Merge Sort

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

        merge_sort(left_half)
        merge_sort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i] < right_half[j]:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1
```

### 7.3 Sort Test

Code for implementing a comparison of bubble sort algorithm and merge sort algorithm.

```
import time
from mergeSort import merge_sort
from bubbleSort import bubble_sort

def testIntBubbleSort(fileName):
    # Measure initialization time
    start_time = time.process_time()

    # Read data from file
    with open(fileName, 'r') as file:
        data = [int(line.strip()) for line in file]
    # Measure sorting time
    sorting_start_time = time.process_time()
    bubble_sort(data)
    sorting_end_time = time.process_time()

    initialization_time = sorting_start_time - start_time
    sorting_time = sorting_end_time - sorting_start_time
    print(f"Initialization Time: {initialization_time} seconds")
    print(f"Sorting Time: {sorting_time} seconds")

def testIntMergeSort(fileName):
    # Measure initialization time
    start_time = time.process_time()

    # Read data from file
    with open(fileName, 'r') as file:
        data = [int(line.strip()) for line in file]
    # Measure sorting time
    sorting_start_time = time.process_time()
    merge_sort(data)
    sorting_end_time = time.process_time()

    initialization_time = sorting_start_time - start_time
    sorting_time = sorting_end_time - sorting_start_time
    print(f"Initialization Time: {initialization_time} seconds")
    print(f"Sorting Time: {sorting_time} seconds")

def testFloatBubbleSort(fileName):
    # Measure initialization time
    start_time = time.process_time()

    # Read data from file
```

```
    with open(fileName, 'r') as file:
        data = [float(line.strip()) for line in file]
    # Measure sorting time
    sorting_start_time = time.process_time()
    bubble_sort(data)
    sorting_end_time = time.process_time()

    initialization_time = sorting_start_time - start_time
    sorting_time = sorting_end_time - sorting_start_time
    print(f"Initialization Time: {initialization_time} seconds")
    print(f"Sorting Time: {sorting_time} seconds")

def testFloatMergeSort(fileName):
    # Measure initialization time
    start_time = time.process_time()

    # Read data from file
    with open(fileName, 'r') as file:
        data = [float(line.strip()) for line in file]
    # Measure sorting time
    sorting_start_time = time.process_time()
    merge_sort(data)
    sorting_end_time = time.process_time()

    initialization_time = sorting_start_time - start_time
    sorting_time = sorting_end_time - sorting_start_time
    print(f"Initialization Time: {initialization_time} seconds")
    print(f"Sorting Time: {sorting_time} seconds")

if __name__ == "__main__":
    # for power in range(1, 6): # Compare up to 10^5
    #     file_name = f'assignment2/TestData/dataInt_10^{power}.txt'
    #     print(f"-------------Comparison for type of Int 10^{power}-------------")
    #     print("Bubble Sort:")
    #     testIntBubbleSort(file_name)
    #     print("Merge Sort:")
    #     testIntMergeSort(file_name)
    #     print()
    for power in range(1, 6): # Compare up to 10^5
        file_name = f'assignment2/TestData/dataFloat_10^{power}.txt'
        print(f"-------------Comparison for type of Float 10^{power}-------------")
        print("Bubble Sort:")
        testFloatBubbleSort(file_name)
        print("Merge Sort:")
        testFloatMergeSort(file_name)
        print()
```