TIMSORT

ASLI KOÇ 2104010063

Timsort is an efficient sorting algorithm that is a combination of Merge Sort and Insertion Sort.

It was designed to perform well on many different kinds of real-world data, and is used as the default sorting algorithm in Python, Java, and many other programming languages.

Overview of Timsort:

1. Divide the array into small pieces of size "run", and sort these using Insertion Sort.

2. Merge the runs using Merge Sort. The size of the runs is doubled until they reach a certain size, at which point they are merged using the Merge Sort algorithm.

3. Repeat step 2 until the entire array is sorted.

Pseudocode for Timsort:

1. Set the size of the initial run

2. Divide the array into runs of the initial size and sort them using Insertion Sort

3. Merge adjacent runs until the array is sorted

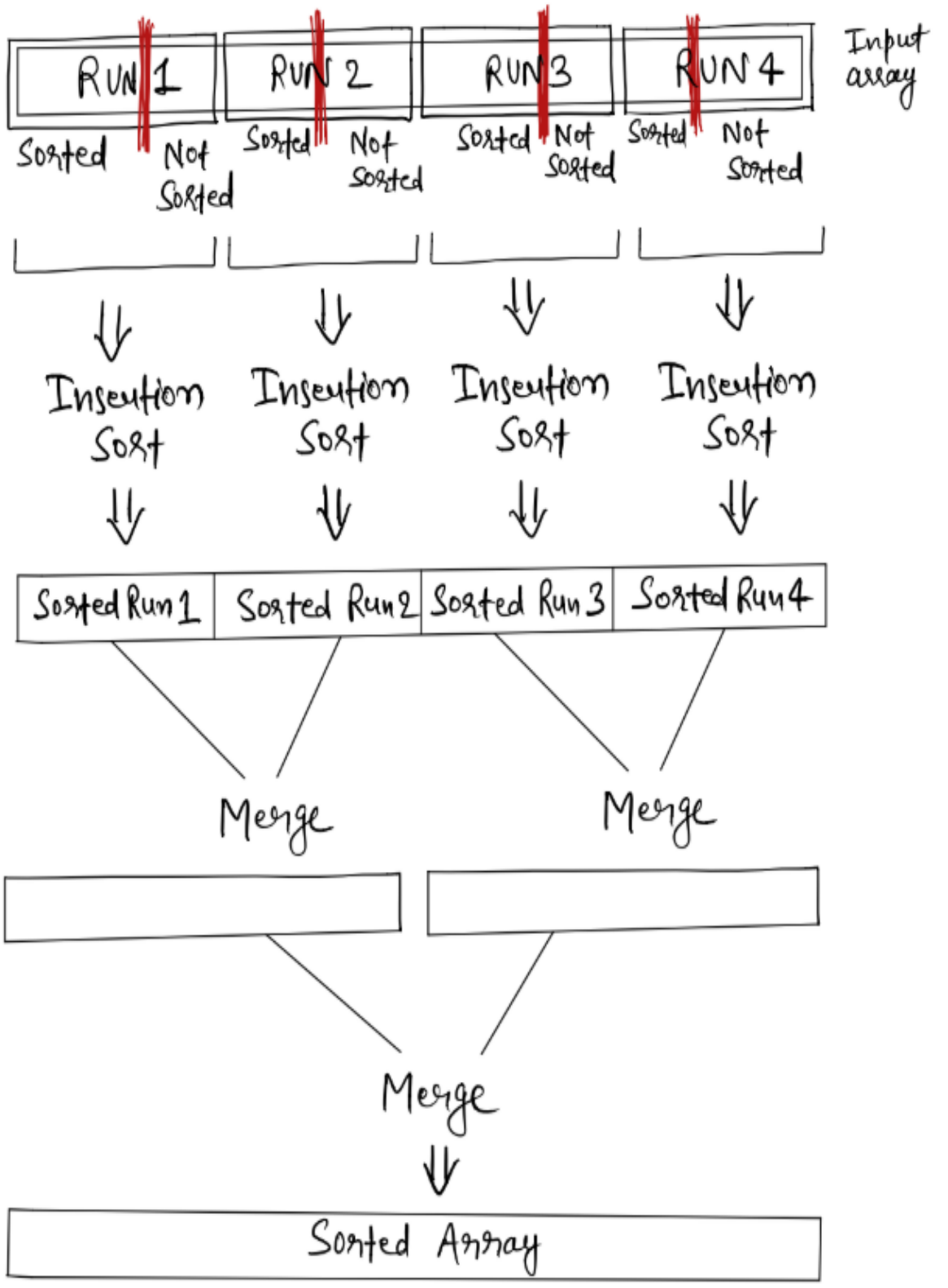
   a. For each pair of adjacent runs, determine the boundaries of the left and right runs
   b. Merge the left and right runs into a single sorted run using Merge Sort

4. Double the size of the runs and repeat step 3 until the array is sorted

5. Return the sorted array

c

```
function timsort(arr):
    // Set the size of the initial run
    run_size = 32

    // Divide the array into runs and sort them using Insertion Sort
    for i from 0 to length(arr) step run_size:
        insertion_sort(arr, i, min(i+run_size, length(arr)))

    // Merge the runs using Merge Sort
    while run_size < length(arr):
        for i from 0 to length(arr) step 2*run_size:
            // Determine the boundaries of the two runs to be merged
            left_start = i
            left_end = min(left_start+run_size, length(arr))
            right_start = left_end
            right_end = min(right_start+run_size, length(arr))

            // Merge the two runs using Merge Sort
            merge(arr, left_start, left_end, right_start, right_end)

        // Double the size of the runs
        run_size *= 2

    return arr
```


RUN 1
RUN 2
RUN 3
RUN 4
Input array
Sorted
Not Sorted
Sorted
Not Sorted
Sorted
Not Sorted
Sorted
Not Sorted
Insertion Sort
Insertion Sort
Insertion Sort
Insertion Sort
Sorted Run 1
Sorted Run 2
Sorted Run 3
Sorted Run 4
Merge
Merge
Merge
Sorted Array

SECOND REVİEW

c++

```
#include <iostream>

void insertionSort(int arr[], int left, int right) {
    for (int i = left + 1; i <= right; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= left && arr[j] > key) {
            arr[j + 1] = arr[j];
            --j;
        }
        arr[j + 1] = key;
    }
}

void merge(int arr[], int left_start, int left_end, int right_start, int right_end) {
    int len1 = left_end - left_start + 1;
    int len2 = right_end - right_start + 1;
    int left[len1];
    int right[len2];
    for (int i = 0; i < len1; ++i) {
        left[i] = arr[left_start + i];
    }
    for (int i = 0; i < len2; ++i) {
        right[i] = arr[right_start + i];
    }
    int i = 0, j = 0, k = left_start;
    while (i < len1 && j < len2) {
        if (left[i] <= right[j]) {
            arr[k++] = left[i++];
        } else {
            arr[k++] = right[j++];
        }
    }
    while (i < len1) {
        arr[k++] = left[i++];
    }
    while (j < len2) {
        arr[k++] = right[j++];
    }
}
```

```
void timSort(int arr[], int n) {
    int run_size = 32;
    for (int i = 0; i < n; i += run_size) {
        insertionSort(arr, i, std::min(i + run_size - 1, n - 1));
    }
    while (run_size < n) {
        for (int i = 0; i < n; i += 2 * run_size) {
            int left_start = i;
            int left_end = std::min(i + run_size - 1, n - 1);
            int right_start = left_end + 1;
            int right_end = std::min(right_start + run_size - 1, n - 1);
            merge(arr, left_start, left_end, right_start, right_end);
        }
        run_size *= 2;
    }
}

int main() {
    int arr[] = {64, 25, 12, 22, 11};
    int n = sizeof(arr) / sizeof(arr[0]);
    std::cout << "Original array: ";
    for (int i = 0; i < n; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    timSort(arr, n);

    std::cout << "Sorted array: ";
    for (int i = 0; i < n; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

1. The insertionSort() function performs insertion sort on a subarray of arr from index left to index right. Insertion sort is used to sort small chunks of the array called "runs" during the initial phase of TimSort.

2. The merge() function performs the merging of two sorted subarrays, one from index left_start to index left_end, and the other from index right_start to index right_end. It uses an auxiliary array to temporarily store the sorted elements during the merging process.

3. The timSort() function is the main implementation of the TimSort algorithm. It starts by performing insertion sort on small runs of size run_size within the array. Then, it repeatedly merges adjacent runs until the entire array is sorted. The size of the runs is doubled after each merge operation.

4. The main() function. It initializes an array arr, prints the original array, calls timSort() to sort the array, and then prints the sorted array

Python

```
def insertion_sort(arr, left, right):
    for i in range(left + 1, right + 1):
        key_item = arr[i]
        j = i - 1
        while j >= left and arr[j] > key_item:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key_item

def merge(arr, left_start, left_end, right_start, right_end):
    len_left = left_end - left_start + 1
    len_right = right_end - right_start + 1
    left = [arr[left_start + i] for i in range(len_left)]
    right = [arr[right_start + i] for i in range(len_right)]
    i, j, k = 0, 0, left_start
    while i < len_left and j < len_right:
        if left[i] <= right[j]:
            arr[k] = left[i]
            i += 1
        else:
            arr[k] = right[j]
            j += 1
        k += 1
    while i < len_left:
```

```
        arr[k] = left[i]
        i += 1
        k += 1
    while j < len_right:
        arr[k] = right[j]
        j += 1
        k += 1

def timsort(arr):
    n = len(arr)
    min_run = 32
    for i in range(0, n, min_run):
        insertion_sort(arr, i, min((i + min_run - 1), n - 1))
    size = min_run
    while size < n:
        for left in range(0, n, 2 * size):
            mid = min(n - 1, left + size - 1)
            right = min(n - 1, mid + size)
            merge(arr, left, mid, right, min(n - 1, right + size - 1))
        size *= 2
    return arr
```

- The algorithm first divides the input array into small chunks of size min_run, and sorts each chunk using insertion sort. Then, it merges the sorted chunks using merge function, which uses the merge step of merge sort. The size of the merged chunks is gradually increased until the entire array is sorted.
    - The implementation uses three helper functions:

1. insertion_sort(arr, left, right): sorts a subarray of arr from left to right using insertion sort.
2. merge(arr, left_start, left_end, right_start, right_end): merges two sorted subarrays of arr from left_start to left_end and from right_start to right_end using the merge step of merge sort.
3. timsort(arr): sorts the input array arr using timsort algorithm.