VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

Project Report

Topic: SORTING ALGORITHMS

Subject: Data Structures and Algorithms

Students:	*Lecturers:*
Vo Tran Quoc Duy	LT. Bui Huy Thong
CLC07 - 23127359	LT. Nguyen Ngoc Thao

Thursday 27th June, 2024

Contents

List of Algorithms

1 Algorithms Presentation

1.1 Heap Sort

1.1.1 Ideas

Heap sort is comparison-based sorting technique base on Binary Heap data structure - a complete Binary Tree, as we only sort the data in ascending order so the Max Binary Heap, whose the key at the root is maximum among all keys, is taken in to use. It is similar to selection sort where we find the minimum element and place it at the beginning then repeat the same process for the remaining elements.

1.1.2 Descriptions

Using these following steps to perform the heap sort algorithm:

- Step 1: Build a heap data structure from the given input array using heapify.
- Step 2: Swap the root element, which is now the largest element, with the last element of the heap.
- Step 3: Heapify the remaining elements of the heap except for the last element.

Repeat Step 2 and Step 3 until the heap contains only 1 element.

Algorithm 1 Heap Sort

```
function SWAP($a$, $b$)
    $a = a \oplus b$
    $b = a \oplus b$
    $a = a \oplus b$
end function

function HEAPIFY($array$, $n$, $i$)
    $largest \leftarrow i$
    $left \leftarrow 2 * i + 1$
    $right \leftarrow 2 * i + 2$

    if $left < n$ && $array[largest] < array[left]$ then
        $largest \leftarrow left$
    end if

    if $right < n$ && $array[largest] < array[right]$ then
        $largest \leftarrow right$
    end if

    if $largest \neq i$ then
        SWAP(array[largest], array[i])
        HEAPIFY($array$, $n$, $largest$)
    end if
end function
```

```
function heapSort($array$, $n$)
    for $i = 0 \rightarrow n - 1$ do
        Heapify($array$, $n$, $i$)
    end for

    for $i = n - 1 \rightarrow 0$ do
        Swap($array[0]$, $array[i]$)
        Heapify($array$, $i$, $0$)
    end for
end function
```

1.1.3 Time Complexity

The problem size is: n - the number of elements of the array.
The process of converting an array of n elements to a heap data structure takes $O(logn)$ logarithmic time - the $O(logn)$ factor is the height of the binary tree, then the last element will be extracted from the array so the size is now $n-1$ and the converting process will continue until the array only has 1 element left. So the total time complexity is:

$$O(logn) + O(log(n-1)) + ... + O(log1) = O(log(n!))$$

Time complexity: $O(nlogn)$

1.1.4 Space Complexity

Since heap sort is an in-place sorting algorithm, it does not require additional storage.
Space complexity: $O(n)$

1.2 Radix Sort

1.2.1 Ideas

Radix sort is a non comparison-based sorting technique. To achieve the finally sorted order, radix sort distributes the elements into buckets based on each digit's value and repeatedly sorting the elements by their significant digits, from the least significant digit to the most significant digit. Radix sort uses the counting sort algorithm to sort the list considering a certain digit.

1.2.2 Descriptions

Using these following steps to perform the radix sort algorithm:

- Step 1: Find the largest value in the array.
- Step 2: Iterate *exp* times, *exp* is the number of digits of the largest value, once for each significant place. In each iteration, performing counting sort algorithm to sort the elements.

Algorithm 2 Radix Sort

```
function getMax($array$, $n$)
    $Max \leftarrow array[0]$

```

```
    for $i = 0 \rightarrow n - 1$ do
        if $Max < array[i]$ then
            $Max \leftarrow array[i]$
        end if
    end for

    return $Max$
end function

function countSort($array$, $n$, $exp$)
    $output$ is an array of $n$ integers
    $count$ is an array of 10 integers, the value of each element is 0

    for $i = 0 \rightarrow n - 1$ do
        $count[\frac{array[i]}{exp} \mod 10] \leftarrow count[\frac{array[i]}{exp} \mod 10] + 1$
    end for

    for $i = 1 \rightarrow 10$ do
        $count[i] \leftarrow count[i] + count[i - 1]$
    end for

    for $i = n - 1 \rightarrow 0$ do
        $output[count[\frac{array[i]}{exp} \mod 10] - 1] = array[i]$

        $count[\frac{array[i]}{exp} \mod 10] \leftarrow count[\frac{array[i]}{exp} \mod 10] - 1$
    end for

    for $i = 0 \rightarrow n - 1$ do
        $array[i] \leftarrow output[i]$
    end for
end function

function radixSort($array$, $n$)
    $max \leftarrow$ getMax($array$, $n$)

    for $exp = 1 \rightarrow \frac{max}{exp} > 1$ do
        countSort($array$, $n$, $exp$)
    end for
end function
```

1.2.3 Time Complexity

The problem size are:

- k - the number of digits of the largest value.
- n - the number of elements of the array.

The counting sort algorithm used in the radix sort algorithm takes $O(n + b)$ logarithmic time with b is the base of the number system, and since we have to perform the counting sort algorithm k times so the time complexity will be $O(k * (n + b))$. **Time complexity:** $O(k * (n + b))$

1.2.4 Space Complexity

1.3 Flash Sort

1.3.1 Ideas

1.3.2 Descriptions

Algorithm 3 Flash Sort

```
function flashSwap(array, n, bucket, length, maxPos, minPos, c)
    $bucketId \leftarrow length - 1$
    $move \leftarrow 0, i \leftarrow 0, flash \leftarrow 0$

    Swap(array[maxPos], array[0])
    while $move < n - 1$ do
        while $i > bucketId - 1$ do
            $i \leftarrow i + 1$
            $bucketId \leftarrow c * (array[i] - array[minPos])$
        end while

        $flash \leftarrow a[i]$
        while $i \neq bucketID$ do
            $bucketId \leftarrow c * (array[i] - arrray[minPos]$
            $bucketId \leftarrow bucketId - 1$
            Swap(flash, a[bucketId])
            $move \leftarrow move + 1$
        end while
    end while
end function

function flashSort(array, n)
    $length \leftarrow 0.45 * n$
    bucket is an array of length integers, the value of each element is 0
    $maxPos \leftarrow 0, minPos \leftarrow 0$

    for $i = 0 \rightarrow n - 1$ do
        if $array[maxPos] < array[i]$ then
            $maxPos \leftarrow i$
        end if

        if $array[minPos] > array[i]$ then
            $minPos \leftarrow i$
        end if
    end for

    $c \leftarrow \frac{length-1}{array[maxPos]-array[minPos]}$
    for $i = 0 \rightarrow n - 1$ do
```

```
        $k \leftarrow c * (array[i] - array[minPos])$
        $bucket[k] \leftarrow bucket[k] + 1$
    end for

    for $i = 1 \rightarrow length - 1$ do
        $bucket[k] \leftarrow bucket[k] + bucket[k-1]$
    end for

    FLASHSWAP(array, n, bucket, length, maxPos, minPos, c)
    INSERTIONSORT(array, n)
end function
```

1.3.3 Time Complexity

1.3.4 Space Complexity

2 Chart Draw and Comment