VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

REPORT FOR LAB 3: SORTING

DATA STRUCTURES AND ALGORITHMS

TOPIC: SORTING ALGORITHMS

Instructor: Master Bui Huy Thong
Class: 20CTT1TN
Student: 20120131 – Nguyen Van Loc

HO CHI MINH CITY, NOVEMBER 2021

1 Information page

Name: Nguyen Van Loc
Student ID: 20120131
Class: 20CTT1TN
Subject: Data structures and algorithms
Lecturer: Doctor Nguyen Thanh Phuong
Instructor: Master Bui Huy Thong
Topic: Sorting algorithms

2 Introduction page

I have completed 11/11 required algorithms, including: selection sort, insertion sort, bubble sort, shaker sort, shell sort, heap sort, merge sort, quick sort, counting sort, radix sort, and flash sort
For output specifications, I have completed 5/5 commands, 3 for algorithm mode and 2 for comparison mode.
Below is the hardware specifications of the computer I used to run these algorithms:

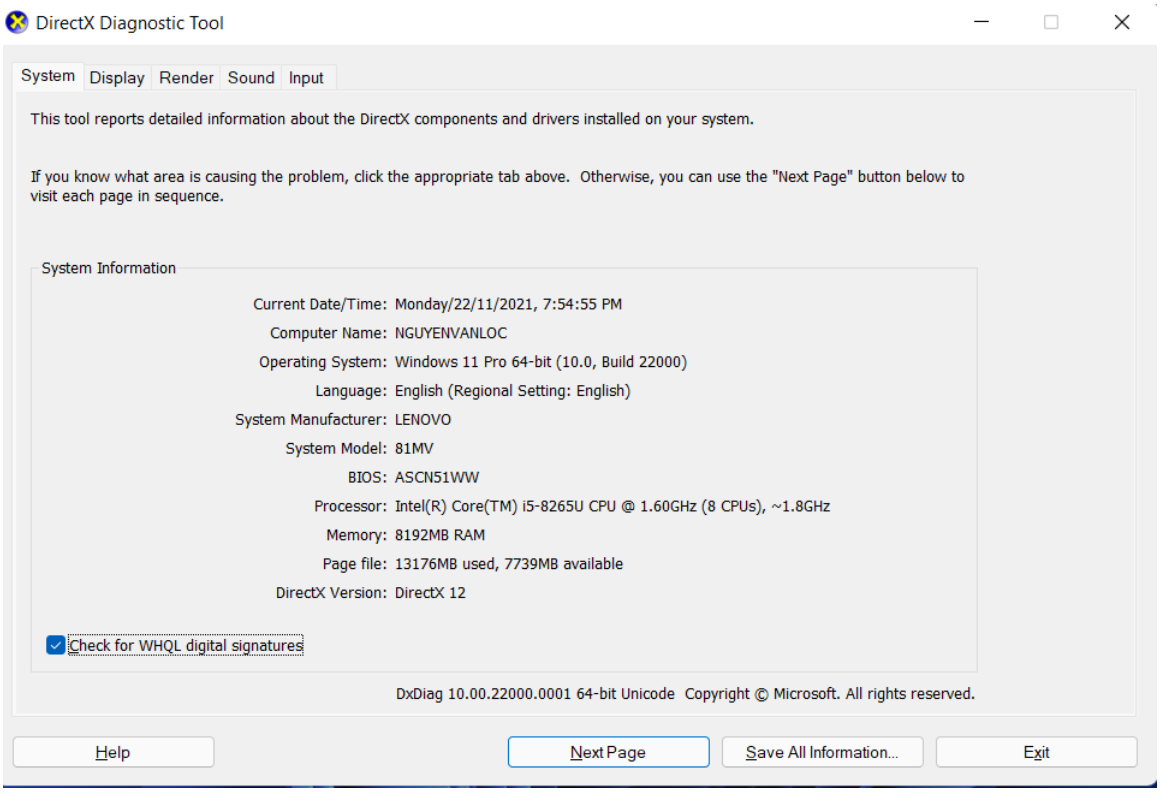

Figure 1 – Hardware specifications

Contents

List of Figures

List of Tables

3 Algorithm presentation

In this section, I will present the algorithms implemented in the project: ideas, step-by-step descriptions, and complexity evaluations. Varians/improvements of an algorithm, if there is any, will be also mentioned.
In this project, sorting algorithms are only used to sort the array in ascending order. Sorting in descending order will be similar.
Most of pseudocodes in this section will be presented in Pascal, with the 1-base array.

3.1 Selection sort

Selection sort is one of the simplest sorting algorithms.
Basic ideas of this algorithm is as followed:

- In the first turn, choose the minimum element in a[1..n], then swap it with a[1], that means a[1] becomes the minimum element of the array.
- In the second turn, choose the minimum element in a[2..n], then swap it with a[2], so that a[2] becomes the second lowest element of the array.
- ...
- In the i-th turn, choose the minimum in a[i..n], then swap it with a[i].
- In the (n-1)-th turn, choose the lower between a[n - 1] and a[n], then swap it with a[n-1].

Pseudocodes: [1]

```
begin
   for i := 1 to n - 1 do
   begin
      jmin := i;
      for j := i + 1 to n do
         if (a[j] < a[jmin]) then jmin := j;
      if (jmin != i) then swap(a[jmin], a[i]);
   end
end
```

Time complexity: [3]

- Worst case: $O(n^2)$.
- Best case: $O(n^2)$.
- Average case: $O(n^2)$.

Space complexity: $O(1)$. [3]

3.2 Insertion sort

Ideas: Consider the array a[1..n].
We see that the subarray with only one element a[1] can be seen as sorted.
Consider a[2], we compare it with a[1], if a[2] ¡ a[1], we insert it before a[1].
With a[3], we compare it with the sorted subarray a[1..2], find the position to insert a[3] to that subarray to have an ascending order.

In a general speech, we will sort the array a[1..k] if the array a[1..k - 1] is already sorted by inserting a[k] to the appopriate position.
Pseudocodes: [1]

```
begin
   for i := 2 to n do
      begin
      temp := a[i];
      j := i - 1;
      while (j > 0) and (temp < a[j]) do
         begin
         a[j + 1] = a[j];
         dec(j);
         end
      a[j + 1] = temp;
      end
end
```

Time complexity: [4]

- Worst case: $O(n^2)$.
- Best case: $O(n)$, in case the array is already sorted.
- Average case: $O(n^2)$.

Space complexity: $O(1)$. [4]
Improvements:

- Binary insertion sort − find the position to insert using binary search, which reduces the number of comparisons. Details at link: `https://www.geeksforgeeks.org/binary-insertion-sor`
- Another improvement of insertion sort is shell sort, which will be presented in section 3.5

3.3 Bubble sort

Ideas: Bubble sort is the simples sorting algorithm, which swaps the adjacent elements if they are in wrong order, repeatedly n times.
After the $i-$th turn, the $i-$th smallest element will be swapped to position i.
Pseudocodes: [1]

```
begin
   for i := 2 to n do
      for j := n downto i do
      if (a[j - 1] > a[j]) then swap(a[j - 1], a[j]);
end
```

Time complexity: $O(n^2)$, not mentioned how the input data is. [1]
Space complexity: $O(1)$. [4]
Variations: There are some variations in the implementation.

- Instead of top-down with `j`, we can iterate from the bottom up, from `i + 1` to `n`.
- Another variation is `j` iterates from `1` to `n - i`. This is the version that I choose in my project.

Improvements: An improvement of bubble sort is shaker sort, which we will research in section 3.4.

3.4 Shaker sort (Cocktail sort)

Ideas: Shaker sort, also called cocktail sort or bi-directional bubble sort, is an improvement of bubble sort. In bubble sort, elements are traversed from left to right, i.e. in one direction only. But shaker sort will traverse in both direction, from left to right and from right to left, alternatively. [6]
Pseudocode: [2]

```
begin
   left := 2;
   right := n;
   k := n;
   repeat
   for j := right downto left do
      if (a[j - 1] > a[j]) then
         begin
         swap(a[j - 1], a[j]);
         k = j;
         end
   left = k + 1; //the last swap position
   for j := left to right do
      if (a[j - 1] > a[j]) then
         begin
         swap(a[j - 1], a[j]);
         k = j;
         end
   right = k - 1;
   until left > right;

end
```

Time complexity: [7]

- Worst case: $O(n^2)$.
- Best case: $O(n)$, in case the array is already sorted.
- Average case: $O(n^2)$.

Space complexity: $O(1)$. [7]

3.5 Shell sort

A drawback of insertion sort is that we always have to insert an element to a position near the beginning of the array. In that case, we use shell sort.
Ideas: Consider an array a[1..n]. For an integer $h : 1 \leqslant h \leqslant n$, we can divide the array into h subarrays:

- Subarray 1: a[1], a[1 + h], a[1 + 2h]...
- Subarray 2: a[2], a[2 + h], a[2 + 2h]...
- ...
- Subarray h: a[h], a[2h], a[3h] ...

Those subarrays are called subarrays with step h. With a step h, shell sort will use insertion sort for independent subarrays, then similarly with $\frac{h}{2}, \frac{h}{4}, ...$ until $h = 1$.
Pseudocodes:

```
begin
   gap := n div 2;
   while (gap > 0) do
      begin
      for i := gap to n do
         begin
         j := i - gap;
         k := a[i];
         while (j > 0 and a[j] > k) do
            begin
            a[j + gap] := a[j];
            j = j - gap;
            end
         a[j + gap] := k;
         end
      gap := gap div 2;
      end
end
```

Time complexity: [8]

- Worst case: $O(n^2)$.
- Best case: $O(n \log n)$.
- Average case: depends on the gap sequence.

Space complexity: $O(1)$. [8]

3.6 Heap sort

Heap sort was invented by J. W. J. Williams in 1981, this algorithm not only introduced an effective sorting algorithm but also built an important data structures to represent priority queues: heap data structure.

3.6.1 Heap data structure

Heap is a special binary tree. A binary tree is said to follow a heap data structure if:

- it is a complete binary tree,
- all nodes in the tree satisfy that they are greater than their children, i.e. the greatest element is the root. Such a heap is called a max-heap. If instead, all nodes are smaller than their childen, it is called a min-heap. [9]

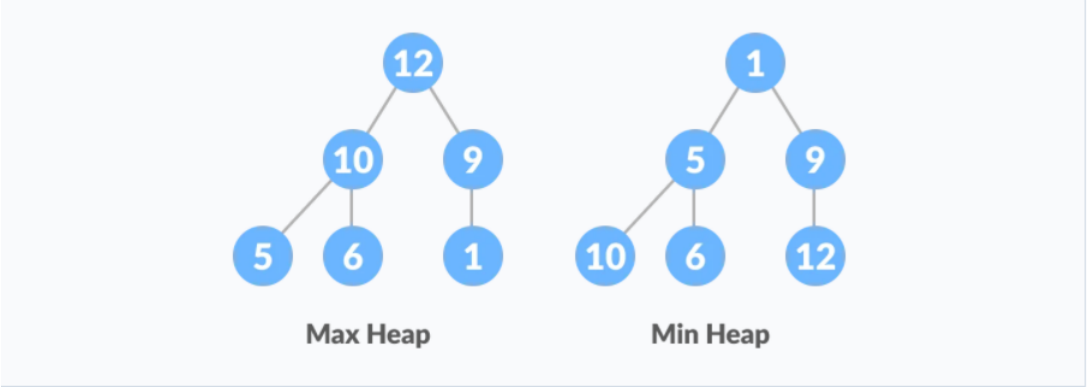

Figure 2 – Max-heap and min-heap. Source: [9]

3.6.2 Build a min-heap

To build a min heap, we: [10]

- Create a new child node at the end of the heap (last level).
- Add the new key to that node (append it to the array).
- Move the child up until we reach the root node and the heap property is satisfied.

To remove/delete a root node in a min heap, we: [10]

- Delete the root node.
- Move the key of last child to root.
- Compare the parent node with its children.
- If the value of the parent is greater than its children, swap them, and repeat until the heap property is satisfied.

3.6.3 Build a max-heap

Building a max-heap is similar to building a min-heap.

3.6.4 Pseudocodes

```
heapify(a[1..n], i)
begin
   max = i;
   left = 2 * i;
   right = 2 * i + 1;
   if (left <= n and a[left] > a[max]) then max = left;
   if (right <= n and a[right] > a[max]) then max = right;
   if (max != i) then
      begin
      swap(a[i], a[max]);
      heapify(a, n, max);
      end
end
```

```
heapsort(a[1..n])
begin
   for i := n div 2 - 1 downto 1 do heapify(a, i);
   for i := n downto 1 do
      begin
      swap(a[0], a[i];
      heapify(a[1..i], 0)
      end
end
```

Time complexity: [9]

- Worst case: $O(n \log n)$.
- Best case: $O(n \log n)$.
- Average case: $O(n \log n)$.

Space complexity: $O(1)$. [9]

3.7 Merge sort

Merge sort is a divide-and-conquer algorithm that was invented by John von Neumann in 1945. This is one of the most popular sorting algorithms.
Ideas:

- Divide the array into two subarrays at the middle position.
- Try to sort both subarrays, if we have not reached the base case yet, continue to divide them into subarrays.
- Merge the sorted subarrays.

Pseudocodes:

```
mergeSort(a[1..n])
begin
   if (n <= 1) do return;
   mid := n div 2;
   left[1..mid] := a[1..mid];
   right[1..n - mid] := a[mid + 1..n];

   mergeSort(left[1..mid]);
   mergeSort(right[1..n - mid]);

   i := 1; j := 1; k := 1;
   while (i <= mid and j <= n - mid)
      begin
      if (left[i] < right[j]) do
         begin
         a[k] := left[i];
         k := k + 1;
         i := i + 1;
         end
```

```
      else
         begin
         a[k] := right[j];
         k := k + 1;
         j := j + 1;
         end
   while (i <= mid) do
      begin
      a[k] := left[i];
      k := k + 1;
      i := i + 1;
      end
   while (j <= n - mid) do
      begin
      a[k] := right[j];
      k := k + 1;
      j := j + 1;
      end
end
```

Time complexity: [11]

- Worst case: $O(n \log n)$.
- Best case: $O(n \log n)$.
- Average case: $O(n \log n)$.

Space complexity: $O(n)$. [11]

References

[1] Le Minh Hoang (2002) *Giai thuat va lap trinh*, Ha Noi University of Education Press

[2] Lectures from Dr. Nguyen Thanh Phuong

[3] https://iq.opengenus.org/time-complexity-of-selection-sort/

[4] https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques

[5] https://www.geeksforgeeks.org/bubble-sort/

[6] https://www.javatpoint.com/cocktail-sort

[7] https://www.geeksforgeeks.org/cocktail-sort/

[8] https://www.tutorialspoint.com/Shell-Sort

[9] https://www.programiz.com/dsa/heap-sort

[10] https://www.educative.io/blog/data-structure-heaps-guide

[11] https://www.programiz.com/dsa/merge-sort