SORTING AND SEARCHING ALGORITHM

Department of Computer Engineering
Sem 1, 2012/2013
LAB SESSION 3: Sorting & Searching Algorithm

1.0 Objectives

1. To explain the concept of sorting techniques. [PO1, C2]
2. To formulate suitable sorting algorithms to increase data performance in terms of memory and run time efficiency of the given programming problem. [PO4, C5]
3. To investigate and analyze the efficiency of each sorting technique. [PO3, C4]
4. To discuss the result from this experiment and present technical report. [PO7, A2]
5. To complete every tasks in this experiment effectively as individual or in group. [PO8, A4]

2.0 Theory

Sorting:

One of the common tasks in data processing is sorting. Sorting is the process of sequencing or arranging a list of data items, such as characters, integers, record/structures, according to some order: ascending or descending. The main purpose of sorting is to improve the search or access time. Without sorting, it will be generally take a longer time to access information in a file or database.

The well-known sorting algorithms are Bubble Sort, Selection Sort, Insertion Sort and Quicksort. Each of these algorithms has its own performance, based on the Big-O notation. Big –O notation can be used to measure the computational complexity of a particular sort based on the number of iterations, comparisons and so on. Table 1 summarizes the running times for some of commonly used sorting algorithms.
Table 1: Comparisons of sorting algorithm

<table>
<thead>
<tr>
<th>Sort algorithm</th>
<th>Characteristics</th>
<th>Worst-case</th>
<th>Average-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection sort</td>
<td>Find the smallest element and exchange it with the first element. Then find the second smallest element and exchange it with the second element, and so on until the array is sorted.</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Insertion sort</td>
<td>Scan successive elements for an unordered item and insert it into its proper place among those elements already sorted.</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>Exchange two adjacent elements in the array if they are out of order. Repeat until the array is sorted.</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>Quicksort</td>
<td>Partition the array into 2 parts. First part contains elements that are less than or equal to the pivot while the second part</td>
<td>O(n^2)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Mergesort</td>
<td>Recursively split or partition an array into halves and sort them. When it gets down to one or two integer arrays, it merges them together to form the complete sorted list.</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>Heapsort</td>
<td>Form a tree with the parent tree being larger than its children.</td>
<td>O(n log n)</td>
<td>O(n log n)</td>
</tr>
</tbody>
</table>

In general, the following criteria should be considered when choosing a sorting algorithm:

- The way the data structures are implemented.
- The way the keys or elements are arranged before sorting.
- The amount of time available for programming or coding.
- The size of the application or problem size.
- Time and space trade-off where we need to save computer time and memory space.

**Searching:**

One of the important activities in data processing is searching. Searching is the process of finding the location of a target item in a table or list. It involves comparing items in the table to a given search key and determining whether there is a match. Efficient search techniques are vitally important for accessing information quickly.

Information is usually stored in the form of table or file. A key (or primary key) is used to facilitate searching. The key can be single field or a combination of fields. The key must be unique.
Linear Search:

- This is a very straightforward loop comparing every element in the array with the key. As soon as an equal value is found, it returns. If the loop finishes without finding a match, the search failed and -1 is returned.
- For small arrays, linear search is a good solution because it's so straightforward.
- In an array of a million elements linear search on average will take 500,000 comparisons to find the key. For a much faster search, take a look at binary search.

Binary Search:

- A fast way to search a sorted array is to use a binary search.
- The idea is to look at the element in the middle. If the key is equal to that, the search is finished. If the key is less than the middle element, do a binary search on the first half.
- If it's greater, do a binary search of the second half.
- Binary search repeatedly divides the array in half until it finds the element or there is nothing left to search.

3.0 Equipment List & Requirement

For this lab session, you need:
1. An editor for C++ programming language installed in your system (i.e. Microsoft Visual Studio C++, Borland C++, C-Free, etc.)
2. Fundamental of C++ programming skill

4.0 TASK

Task 1:

Question 1

By using some figures and example, illustrate the sorting process of the following algorithms:

a. Bubble sort
b. Selection sort
c. Insertion sort
d. Quicksort
Question 2

Write the algorithms to illustrate the linear and binary search techniques. Then, Identify and elaborate the differences between both algorithms.

Task 2: Programming

Question 1

Write a program in C++ that applies different sorting techniques to solve integer sorting problem. Your program must operate as follows:

1. Prompts the user to indicate whether they wish to sort in ascending or descending order.
2. Prompts the user to enter size of array and the numbers.
3. Write a function to implement the Bubble sort algorithm. Display the output.
4. Write a function to implement the Selection sort algorithm. Display the output.
5. Write a function to implement the Insertion sort algorithm. Display the output.
6. Write a function to implement the Quicksort algorithm. Display the output.

Note: You can write the program for Task 2 either in one program (using selection/option) or separately for each sorting technique.

Sample output:

```plaintext
Select (1) for ascending, (2) for descending order: 1
Enter list of size: 6
Enter number: 4
Enter number: 2
Enter number: 3
Enter number: 1
Enter number: 6
Enter number: 5

Sorted list for Bubble sort: 1  2  3  4  5  6
```

Or

```plaintext
Select (1) for ascending, (2) for descending order: 2
Enter list of size: 6
Enter number: 4
Enter number: 2
Enter number: 3
```
Enter number: 1
Enter number: 6
Enter number: 5

Sorting techniques:
Select (1) for Bubble
Select (2) for Selection
Select (3) for Insertion
Select (4) for Quicksort
Your choice: 4

Sorted list for Quick sort: 6   5   4   3   2   1

QUESTION 2:

Write a single program to apply linear and binary search which can operate as follows:

a. Prompts the user to enter size of array and the numbers.
b. Write and test a function to implement the linear search algorithm.
c. Write and test a function to implement sorting algorithm.
d. Write and test a function to implement the binary search algorithm.
e. Display the output.

QUESTION 3:

Write the same program as in question 1 but this time you need to write the program in different .cpp file and create header file for each .cpp file that you have created. The program should satisfy the following requirements:

a. Prompts the user to enter size of array and the numbers in main.cpp file.
b. Write a function to implement the linear search algorithm in linear.cpp file.
c. Write a function to implement the sorting algorithm in sort.cpp file.
d. Write a function to implement the binary search algorithm in binary.cpp file.
e. Display the output in the main program.
The integration of the whole files is shown in Figure 1.

```
main.cpp
linear_f()
sort_f()
binary_f()
display

linear.cpp
linear_f()
//linear search algorithm

sort.cpp
sort_f()
//sorting algorithm

binary.cpp
binary_f()
//binary search algorithm
```

**Figure 1**

**Sample output:**

```
Enter list of size: 10
Enter number: 22
Enter number: 101
Enter number: 5
Enter number: 45
Enter number: 32
Enter number: 67
Enter number: 73
Enter number: 99
Enter number: 98
Enter number: 10

Data entered: 22, 101, 5, 45, 32, 67, 73, 99, 98, 10

-- By linear search --
Enter the item you are searching for: 101
The item was found at index location 1

-- After sorting --
Data entered: 5, 10, 22, 32, 45, 67, 73, 98, 99, 101

-- By binary search --
```
Enter the item you are searching for: 101
The item was found at index location 9

5.0 Result

1. Observe and record your result for each programming task.
2. Save all your code in a folder which named by your group number and make the folder as a zip file. Submit the code to yahoo group in folder lab submission 3.

6.0 Discussion / Conclusion

1. Conclude the findings throughout your lab experiment and answer all the questions in this lab sheet.