Bahria University, Islamabad

Department of Software Engineering

Data Structre And Algorithms

(Fall-2024)

Teacher: Engr. Aleem Ahmad

Student : Lotfullah Muslimwal

Enrollment : 01-131232-039

Lab Journal: X

Date:

<table>
<tr><th rowspan="2">Task No:</th><th colspan="2">Task Wise Marks</th><th colspan="2">Documentation Marks</th><th rowspan="2">Total Marks (20)</th></tr>
<tr><th>Assigned</th><th>Obtained</th><th>Assigned</th><th>Obtained</th></tr>
<tr><td>1</td><td>3</td><td></td><td rowspan="5">5</td><td rowspan="5"></td><td rowspan="5"></td></tr>
<tr><td>2</td><td>3</td><td></td></tr>
<tr><td>3</td><td>3</td><td></td></tr>
<tr><td>4</td><td>3</td><td></td></tr>
<tr><td>5</td><td>3</td><td></td></tr>
</table>

Comments:

Signature

LAB

Lab No: 11

Task 1: Sorting Algorithm Performance Comparison

Code:

```
/*
 * This program compares the execution times of three sorting algorithms: Bubble Sort,
Selection Sort, and Binary Sort.
 * It tests the algorithms on arrays of varying sizes (10, 100, 1000, 10000, 100000) and
outputs the time taken
 * for each sorting algorithm in milliseconds. Here's the detailed breakdown:
 *
 * 1. **generateRandomArray(int arr[], int n):**
 *    - This function generates an array of size 'n' with random integer values between 0 and
999.
 *
 * 2. **bubbleSort(int arr[], int n):**
 *    - This function sorts the array using the Bubble Sort algorithm, which repeatedly
compares adjacent elements
 *      and swaps them if they are in the wrong order. The sorting continues until the array is
fully sorted.
 *    - It measures the time taken to complete the sorting using
`chrono::high_resolution_clock`.
 *    - The time is returned in milliseconds.
 *
 * 3. **selectionSort(int arr[], int n):**
 *    - This function sorts the array using the Selection Sort algorithm, which repeatedly
selects the smallest
 *      element from the unsorted part of the array and swaps it with the first unsorted
element.
 *    - It also measures the time taken to perform the sorting and returns the time in
milliseconds.
 *
 * 4. **binarySort(int arr[], int n):**
 *    - This function sorts the array using a modified binary search algorithm for insertion
sorting.
 *    - For each element in the array, it uses binary search to find the appropriate position
where the element should
 *      be inserted, and then shifts the elements to make space for it.
 *    - It measures the time taken and returns the time in milliseconds.
 *
 * 5. **binarySearch(int arr[], int low, int high, int key):**
```

```
 *    - This is the helper function used by `binarySort` to find the position to insert an element using binary search.
 *
 * 6. **printTableRow(int size, double bubbleTime, double selectionTime, double binaryTime):**
 *    - This function prints the results in a formatted table. It displays the array size and the time taken for each
 *      sorting algorithm (Bubble Sort, Selection Sort, and Binary Sort) in milliseconds.
 *
 * 7. **main() function:**
 *    - The main function initializes an array of different sizes (10, 100, 1000, 10000, 100000) and for each size,
 *      it generates a random array and runs all three sorting algorithms: Bubble Sort, Selection Sort, and Binary Sort.
 *    - It then prints the execution time of each algorithm for each array size in a table format.
 *
 * 8. **Dynamic Memory Allocation:**
 *    - Arrays are dynamically allocated for each size in the `sizes[]` array to ensure flexibility in array size.
 *    - After each sorting, the memory is deallocated to avoid memory leaks.
 */


#include <iostream>     // Used for input and output operations like cout, cin, etc.
#include <ctime>        // Provides functions for handling time-related operations such as time(NULL).
#include <cstdlib>      // Provides functions for random number generation like rand() and srand().
#include <iomanip>      // Used for formatting output, such as setting precision or width of printed values.
#include <chrono>       // Provides utilities for measuring time intervals, such as high_resolution_clock and duration.

using namespace std;
using namespace std::chrono;  //namespace contains time - related functionality


void generateRandomArray(int arr[], int n);
double bubbleSort(int arr[], int n);
double selectionSort(int arr[], int n);
double binarySort(int arr[], int n);
int binarySearch(int arr[], int low, int high, int key); // Helper function for Binary Sort to find the correct position
```

```
void printTableRow(int size, double bubbleTime, double selectionTime, double binaryTime);

int main() {
   const int sizes[] = { 10, 100, 1000, 10000, 100000 }; // Different array sizes to test
   const int numSizes = 5;
   cout << setw(10) << "Size" << setw(20) << "Bubble Sort (ms)"
      << setw(20) << "Selection Sort (ms)" << setw(20) << "Binary Sort (ms)" << endl;
   cout << string(70, '-') << endl; // Printing the header of the table

   for (int i = 0; i < numSizes; ++i) { // Loop through each size to perform the tests
      int size = sizes[i]; // Get the current size
      int* arr = new int[size]; // Dynamically allocate memory for the array

      generateRandomArray(arr, size); // Generate random values for the array
      double bubbleTime = bubbleSort(arr, size); // Measure and store Bubble Sort time

      generateRandomArray(arr, size); // Generate new random values for the array
      double selectionTime = selectionSort(arr, size); // Measure and store Selection Sort time

      generateRandomArray(arr, size); // Generate new random values for the array
      double binaryTime = binarySort(arr, size); // Measure and store Binary Sort time

      printTableRow(size, bubbleTime, selectionTime, binaryTime); // Print the results for the current array size

      delete[] arr; // Deallocate the dynamically allocated memory to avoid memory leaks
   }

   return 0;
}

void generateRandomArray(int arr[], int n) {
   srand(static_cast<unsigned int>(time(NULL))); // Seed the random number generator with the current time
   for (int i = 0; i < n; ++i) { // Loop through the array
      arr[i] = rand() % 1000; // Assign a random value between 0 and 999 to each element
   }
}

double bubbleSort(int arr[], int n) {
   auto start = high_resolution_clock::now(); // Start the timer

   for (int i = 0; i < n - 1; ++i) { // Loop through the array elements
```

```
        for (int j = 0; j < n - i - 1; ++j) { // Compare adjacent elements
            if (arr[j] > arr[j + 1]) { // If the current element is greater than the next, swap them
                swap(arr[j], arr[j + 1]);
            }
        }
    }

    auto end = high_resolution_clock::now(); // End the timer
    duration<double> duration = end - start; // Calculate the elapsed time
    return duration.count() * 1000; // Convert the time to milliseconds and return
}

double selectionSort(int arr[], int n) {
    auto start = high_resolution_clock::now(); // Start the timer

    for (int i = 0; i < n - 1; ++i) { // Loop through the array elements
        int minIndex = i; // Assume the current element is the minimum
        for (int j = i + 1; j < n; ++j) { // Compare with the rest of the elements
            if (arr[j] < arr[minIndex]) { // If a smaller element is found, update the minimum index
                minIndex = j;
            }
        }
        swap(arr[i], arr[minIndex]); // Swap the current element with the found minimum
element
    }

    auto end = high_resolution_clock::now(); // End the timer
    duration<double> duration = end - start; // Calculate the elapsed time
    return duration.count() * 1000; // Convert the time to milliseconds and return
}

double binarySort(int arr[], int n) {
    auto start = high_resolution_clock::now(); // Start the timer

    for (int i = 1; i < n; ++i) { // Loop through the array elements starting from the second
element
        int key = arr[i]; // The current element to insert
        int low = 0;
        int high = i; // Binary search range

        int pos = binarySearch(arr, low, high, key); // Find the correct position for the current
element
        for (int j = i; j > pos; --j) { // Shift the elements to make space for the current element
            arr[j] = arr[j - 1];
```

```
        }

        arr[pos] = key; // Insert the element at the correct position
    }

    auto end = high_resolution_clock::now(); // End the timer
    duration<double> duration = end - start; // Calculate the elapsed time
    return duration.count() * 1000; // Convert the time to milliseconds and return
}

int binarySearch(int arr[], int low, int high, int key) {
    while (low < high) { // Binary search loop
        int mid = (low + high) / 2; // Find the middle element
        if (key < arr[mid]) { // If the key is smaller, search the left half
            high = mid;
        }
        else { // If the key is larger, search the right half
            low = mid + 1;
        }
    }
    return low; // Return the position where the key should be inserted
}

void printTableRow(int size, double bubbleTime, double selectionTime, double binaryTime) {
    // Print the results in a formatted row with each sorting algorithm's time
    cout << setw(10) << size
        << setw(20) << fixed << setprecision(4) << bubbleTime
        << setw(20) << fixed << setprecision(4) << selectionTime
        << setw(20) << fixed << setprecision(4) << binaryTime
        << endl;
}
```

GitHub-Link: https://github.com/lotfullahmsl/DSA-Lab-FA2024

Screenshot:

```
      Size     Bubble Sort (ms) Selection Sort (ms)     Binary Sort (ms)
----------------------------------------------------------------------
        10               0.0008              0.0006              0.0007
       100               0.0462              0.0210              0.0143
      1000               3.6516              1.3821              0.6870
     10000             410.8019            126.1045             50.6894
```